École Nationale Supérieure des Mines de Saint Étienne

Rapport de projet

GP - Conception des Systèmes Electroniques 1

Application Robotique : Projet Robot

Élèves :
Ibrahim HADJ-ARAB
Gevorg ISHKHANYAN

Enseignants :
Acacio MARQUES
Adil YACOUBI
Yannick MARRIETI

Remerciements

Nous tenons à remercier respectivement **Acacio MARQUES**, **Adil YACOUBI**, ainsi que **Yannick MARRIETI** pour leurs conseils et indications précieuses qui nous ont aidé durant ce projet.

Nous remercions également l'Association de Robotique et d'Electronique des Mines **(AREM)**, d'avoir toujours laissé la porte ouverte aux échanges concernant le robot et d'avoir toujours des astuces précieuses qui ont pu nous faire gaganer du temps pendant ce projet.

Table des matières

I Introduction **1**
1 Cahier des charges . . . 1
1.1 Présentation du sujet . . . 1
1.2 Outils à notre disposition . . . 2
2 Répartition du travail . . . 2

II Conception du Projet **3**
1 Le fonctionnement global . . . 3
2 Surveillance Batterie . . . 5
2.1 Principe de fonctionnement . . . 5
2.2 Convertisseur Analogique Numérique (ADC) . . . 5
2.3 Analog Watchdog . . . 5
2.4 Le Timer . . . 6
2.5 Allumer la LED . . . 7
3 Bouton ON/OFF . . . 8
4 Commmander le robot . . . 9
4.1 Commander les moteurs . . . 9
4.2 Avancer . . . 11
4.3 Tourner . . . 11
5 Le Balayage . . . 12
5.1 Utilisation du SONAR . . . 12
5.2 Le Servo Moteur . . . 14
5.3 Convertir les degrés . . . 16
6 Le Calibrage . . . 18
6.1 Mesure d'angles . . . 18
6.2 Calcul d'erreurs . . . 20
7 La machine d'état . . . 21

III Test du Robot **23**
1 Comparaison avec les valeurs théoriques . . . 23
1.1 Test des valeurs de la PWM des moteurs . . . 23
1.2 Distance sonar . . . 24
2 Améliorations apportées . . . 25
2.1 Filtre anti-rebond . . . 25
2.2 Stratégie de déplacement . . . 26
2.3 Minimiser erreur calibration . . . 26

IV Conclusion **27**
1 Evolution possible du contrat . . . 27
1.1 Filtrage Sonar . . . 27
1.2 Utilisation d'encodeurs pour la calibration . . . 27
2 Applications industrielles du contrat . . . 27

V Annexes **28**
1 Annexe 1 : Code du main.c . . . 28

Table des figures

I.1 Le robot 2
II.1 Périphériques utilisées 3
II.2 IOC Complet 4
II.3 Algorigramme du main 4
II.4 Algorigramme de l'Initialisation des variables (Init()) 4
II.5 Algorigramme de la boucle infinie (Loop()) 5
II.6 Configuration sur IOC pour surveillance batterie 5
II.7 Configuration du Timer 6 6
II.8 Algorigramme du TIMER 6 7
II.9 Algorigramme de l'Analog Watchdog 7
II.10 Algorigramme du Bouton 8
II.11 Algorigramme de la commande en Vitesse du Robot 9
II.12 Configuration du Timer 2 10
II.13 Configuration des PWM pour les moteurs 10
II.14 Algorigramme de la fonction Avancer 11
II.15 Algorigramme de la fonction pour Tourner 12
II.16 LV-MaxSonar-EZ24 13
II.17 Mesure distance SONAR 13
II.18 Algorigramme SONAR 14
II.19 Réglages du TIMER1 14
II.20 SGservo 9g 15
II.21 Algorigramme qui convertit les degré en valeur lisible par le CCR 16
II.22 Réglages du TIMER5 16
II.23 Algorigramme du Timer 5 17
II.24 Configuration du Timer 7 18
II.25 Algorigramme du Timer 7 19
II.26 Calcul de calibration 20
II.27 Algorigramme de la Machine d'état Fini 21
III.1 PWM à 20% 23
III.2 PWM à 50% 24
III.3 PWM à 70% 24
III.4 Variable distance en fonction du temps 25
III.5 Algorigramme pour le filtre anti-rebond 25

Liste des tableaux

1 Prescaler et ARR du TIMER 1 du SONAR 14
2 Valeur lisible par le CCR en degré 15
3 18

I Introduction

Le projet robot est une initiative menée dans le cadre de notre cours sur les systèmes à microcontrôleurs, qui vise à concevoir, programmer et contrôler un robot autonome à l'aide d'une carte Nucleo STM32. Ce projet s'inscrit dans la suite et est une suite logique du cours "Systèmes à microcontrôleurs 1 (SAM1)", où nous avons acquis des compétences en utilisation et programmation des microcontrôleurs STM32.

Le but de ce projet est de mettre en pratique les connaissances acquises dans le cours SAM1 en développant un système embarqué complexe utilisant un microcontrôleur STM32. Nous avons choisi de concevoir un robot autonome qui utilise un capteur sonar pour la navigation.

1 Cahier des charges

1.1 Présentation du sujet

Le cahier des charges du projet est le suivant :

- Concevoir un robot autonome à l'aide d'une carte Nucleo STM32
- Utiliser un capteur sonar pour la navigation
- Suite à la commande start, le robot doit balayer l'espace à la recherche de son objectif
- Une fois l'objectif identifié, le robot doit s'aligner et avancer jusqu'à 20 cm de celui-ci
- Le robot doit suivre l'objectif en gardant toujours cette distance de 20 cm
- Le robot doit fonctionner en boucle ouverte

Les spécifications du projet sont les suivantes :

- PWM : 4 kHz
- Vitesse : 20 cm/s
- Précision : +/- 2 cm

En utilisant les compétences acquises dans le cours SAM1 et en suivant le cahier des charges du projet, nous allons concevoir, programmer et contrôler un robot autonome qui utilise un capteur sonar pour la navigation. Nous allons également documenter le processus de développement du projet et présenter les résultats obtenus. Le robot sera capable de suivre un objectif en maintenant une distance constante de 20 cm, comme un caniche qui suit son maître.

1.2 Outils à notre disposition

FIGURE I.1 – Le robot

Pour ce faire nous avons à notre disposition un robot utilisant un microcontroleur STM32 NUCLEO-L476RG, le même que nous avions utilisé lors de notre participation au Hackathon organisé par AREM (Association de Robotique et d'Electronique des Mines) en début d'année. Le travail s'effectuera sur le logiciel STM32CubeIDE.

Nous avions également à notre disposition des outils comme des oscilloscopes afin de visualiser des signaux (notamment les PWM que nous verrons plus tard). Mais également des multimètres qui nous ont été parfois utiles pour vérifier la tension de la batterie de notre robot sans aller dans le débugger.

2 Répartition du travail

Pour réaliser ce projet, nous nous sommes répartis ce travail de manière aussi équitable que nos emplois du temps nous ont permis. Globalement l'investissement des deux binômes a été proche et complémentaire. Nous avons décidé de commencer le projet sur de bonnes bases en créant directement un **espace de commun** où nous pourrions échanger des fichiers et documenter ce que nous faisons, nous avons choisi de faire une **page Notion** pour faire cela. Nous vous invitons à regarder pour comprendre notre organisation lors du projet :En cliquant ici
Ou en vous rendant sur :
gevorgs.notion.site/1f51df96f14a4bd28614aea562d28665 ?v=02fc9f7c6f2d4c109b59a356f32cc131

II Conception du Projet

1 Le fonctionnement global

Pour faire fonctionner le robot et qu'il puisse répondre à nos exigences, nous allons utiliser plusieurs périphériques.

— Le sonar
— Le servo-moteur
— Les moteurs
— Le bouton Start/Stop
— Le Convertisseur Analogique-Numérique (ADC1)

Les différents timers seront naturellement utilisés pour cadencer le fonctionnement du robot.

FIGURE II.1 – Périphériques utilisées

Le chef d'orchestre de ces périphériques sera une machine d'état fini (FSM), qui commandera les états du robot en fonction de ce qu'il observe avec le sonar (partie qui est détaillée plus tard).

Nous avons configuré notre MCU (microcontrôleur) comme il suit (détails par la suite), ce qui va nous permettre d'allouer les pins pour les périphériques utilisées. Ces choix des pins ont été faits à l'aide de la datasheet donnée dans le cours, indiquant ce que contrôle chaque pin sur la carte électronique du robot.

FIGURE II.2 – IOC Complet

Voici les différents algorigrammes du fonctionnement global de notre projet :

FIGURE II.3 – Algorigramme du main

FIGURE II.4 – Algorigramme de l'Initialisation des variables (Init())

FIGURE II.5 – Algorigramme de la boucle infinie (Loop())

2 Surveillance Batterie

2.1 Principe de fonctionnement

Pour réaliser la survéillance de la batterie, nous allons utiliser plusieurs périphériques du microcontrôleur. Le but sera de lire la valeur de la batterie (qui est un signal analogique), puis de la convertir en valeur numérique grâce au ADC. Une fois la valeur numérique obtenue, un analog watchdog vérifiera que la valeur ne dépasse pas les seuils que nous avons définis, si on n'a pas assez de batterie, la fonction d'interruption allumera une LED pour indiquer que la batterie est faible.

Un Timer sera utilisé afin de faire ces mesures à un instant souhaité.

2.2 Convertisseur Analogique Numérique (ADC)

(a) Configuration de l'ADC (b) Configuration du Watchdog

FIGURE II.6 – Configuration sur IOC pour surveillance batterie

Ayant un ADC qui est sur 12 bits, nous avons 4096 valeurs possibles. La valeur maximale de l'ADC étant 4095 et celle de la batterie étant de 5V, nous trouvons en faisant un produit en croix que pour être à 3.3V il nous faudrait une valeur de l'ADC égale à 3723.

2.3 Analog Watchdog

Un analog watchdog est une fonctionnalité utilisée pour surveiller les valeurs analogiques provenant d'un ADC. Voici comment cela fonctionne en bref :

Deux seuils de tension, haut et bas, sont définis. Ces seuils déterminent les limites entre lesquelles la tension d'entrée doit se situer. Le microcontrôleur surveille en continu la tension d'entrée de l'ADC. Si la tension d'entrée dépasse le seuil haut ou descend en dessous du seuil bas, une condition de watchdog est déclenchée. Lorsque la condition de watchdog est déclenchée, une interruption est générée, le code rentre alors dans la fonction Callback qui correspond.

Les étapes de configuration sont :

- Initialisation de l'ADC : Configurer l'ADC pour lire les valeurs analogiques des capteurs ou des broches spécifiques. (Voir images au dessus)
- Définition des Seuils : Configurer les registres de l'ADC pour définir les valeurs de seuil haut et bas. Ici on a 0 et 3723.
- Activation de l'Analog Watchdog : Activer la fonctionnalité de watchdog pour les canaux ADC concernés.
- Gestion des Interruptions : Écrire une fonction HAL_ADC_LevelOutOfWindowCallback() pour gérer les cas où les valeurs de l'ADC dépassent les seuils définis.

2.4 Le Timer

On utilise le TIMER_6, en activant l'option NVIC pour autoriser les interruptions.

FIGURE II.7 – Configuration du Timer 6

Calculs : Nous voulons que le timer soit configuré à T = 1ms, donc f = 1000 Hz.

$$\boxed{PSC = \frac{80.10^6}{2^{16}.f} = \frac{80.10^6}{2^{16}.1000} = 1.22 \approx 2 - 1} \tag{1}$$

$$\boxed{ARR = \frac{80.10^6}{f.PSC} = \frac{80.10^6}{1000.2} = 40000} \tag{2}$$

A chaque coup de clock du Timer_6, donc tous les 1ms, on incrémente la valeur de T_Batt (voir la fonction HAL_TIM_PeriodElapsedCallback).
A T_Batt = 500, car on veut faire une mesure tous les 500ms, on fait une surveillance de la batterie.

FIGURE II.8 – Algorigramme du TIMER 6

2.5 Allumer la LED

Nous avons utilisé la **LED2** se trouvant sur la carte STM32. Dans l'IOC nous avons configuré le port **PA5** en **GPIO_Output**, que nous allons nommer **LED_Batt**. Nous avons pris cette pin car c'était celle qui commandait la LED d'après la datasheet.

La LED s'allume uniquement lorsque la fonction d'interruption du watchdog est appelée.

```
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    // Allumer la LED si le seuil est depasse
    HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin, GPIO_PIN_SET);
  }
}
```

Listing 1 – Code d'interruption de l'analog Watchdog

FIGURE II.9 – Algorigramme de l'Analog Watchdog

3 Bouton ON/OFF

Nous avons utilisé le **Bouton Bleu** se trouvant sur la carte STM32 pour démarrer et arrêter le robot. Dans l'IOC nous avons configuré le port **PC13** en **GPIO_EXTI13** (en interruption), que nous allons nommer Bouton_Bleu. Nous avons pris cette pin car c'était celle qui commandait le bouton.

Nous l'avons configuré en interruption afin qu'elle interrompt le programme si on appuie dessus pour faire un arrêt d'urgence.

Nous avons décidé d'utiliser une variable globale **Start_value** que nous mettons à **1** pour indiquer que le robot peut rouler, et à **0** pour indiquer que le robot doit rester immobile. Ainsi, avec une condition dans la boucle while du main() et dans la FSM, le robot ne fait une action que si la valeur de **Start_value** vaut **1**.

Dans ce cas, Start_value = 0 au début, c'est à l'utilisateur d'appuyer sur le bouton pour le faire passer à 1 et lancer le robot. Pour les prochains appuis, cela dépendra de la valeur de Start_value, si elle est active alors on arrête et on initialise le robot, sinon on lance le robot. Le comportement est détaillé dans l'algorigramme qui suit :

FIGURE II.10 – Algorigramme du Bouton

4 Commmander le robot

4.1 Commander les moteurs

Afin de cadencer les moteurs, nous avons utilisé le Timer 2.

Pour mettre en place la commande de l'embase nous avons utilisé 4 pins, 2 pour les canaux de **PWM du timer 2**, et 1 pour les commandes en sortie pour chaque côté du robot qui seront des GPIO_Output. Nous utilisons la Pin PC8 nommé DIR1 qui correspond à la partie droite, et la Pin PB2 appelée DIR2 pour la partie gauche.

Pour faire tourner les moteurs, nous allons le faire en indiquant les valeurs des rapport cycliques que nous souhaitons avoir pour les PWM. Voici la fonction qui nous permet de commander la PWM ;

```
void CommandeMoteur(int percent) {
    if (percent < 0 || percent > 100) {
        // Pourcentage invalide, on peut mettre une gestion d'erreur ici
        return;
    }
    // Calcul du rapport cyclique num rique (CCPR)
    uint32_t ccr_value = (htim2.Init.Period + 1) * percent / 100;

    // Mettre    jour les registres CCR des canaux PWM des moteurs
   gauche et droit
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, ccr_value);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, ccr_value);
}
```

Listing 2 – Fonction de setup du rapport cyclique

FIGURE II.11 – Algorigramme de la commande en Vitesse du Robot

Pour répondre à la contrainte sur la PWM qui doit être égale à **4000 Hz**, nous allons configurer le Timer 2 en fonction. Voici le calcul pour la configuration du timer 2 :

$$\boxed{PSC = \frac{80.10^6}{4.10^3.2^{16}} \implies PSC = 1 - 1} \tag{3}$$

$$\boxed{ARR = \frac{80.10^6}{4.10^3.1} \implies ARR = 20\ 000} \tag{4}$$

La configuration du Timer 2 dans l'IOC, **en mode génération de PWM** :

Counter Settings	
Prescaler (PSC - 16 bits value)	1-1
Counter Mode	Up
Counter Period (AutoReload Re...	20000
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

FIGURE II.12 – Configuration du Timer 2

PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (32 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 4	
Mode	PWM mode 1
Pulse (32 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High

FIGURE II.13 – Configuration des PWM pour les moteurs

4.2 Avancer

Poue faire avancer le robot, il suffit de mettre la valeur des deux Pins qui contrôlent les moteurs à 1, car il faut que les deux chenilles avancent. Nous avons la possibilité d'indiquer le rapport cyclique directement en appelant la fonction Avancer().

FIGURE II.14 – Algorigramme de la fonction Avancer

```
void Avancer(int percent) {
    // Mettre les broches DIRD et DIRG    1 pour avancer
    HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, GPIO_PIN_SET);

    // Commande du moteur avec le rapport cyclique donn
    CommandeMoteur(percent);
}
```

Listing 3 – Fonction Avancer

Etant donné la contrainte de vitesse de 20cm/s, nous avons fait des mesures afin de voir pour quelle valeur du rapport cyclique nous aurions cette vitesse.
Pour un rapport cyclique de 100%, nous avons observé une vitesse de 30cm/s.
Par simple produit en croix, nous avons trouvé qu'il faudrait mettre un rapport cyclique de $100 \times \frac{2}{3} = 66.6$. Nous appellerons donc toujours la fonction Avancer(66) afin d'avoir la vitesse exigée.

4.3 Tourner

Pour faire tourner le robot à gauche ou à droite, nous avons utilisé la fonction "Tourner" qui prend en paramètre la direction souhaitée et le pourcentage en vitesse. Cette fonction utilise les broches DIR1 et DIR2 pour contrôler la direction des moteurs du robot.

Ainsi, lorsque la fonction est appelée avec la direction "D" ou "d", le robot tourne à droite. Pour cela, nous réglons la broche DIR1 à l'état bas (GPIO_PIN_RESET) et la broche DIR2 à l'état haut (GPIO_PIN_SET).

De la même manière, lorsque la fonction possède en argument la direction "G" ou "g", le robot tourne à gauche. Pour cela, nous réglons la broche DIR1 à l'état haut (GPIO_PIN_SET) et la broche DIR2 à l'état bas (GPIO_PIN_RESET).

Dans les deux cas, on termine la fonction en appliquant la vitesse souhaitée aux roues du robot.

FIGURE II.15 – Algorigramme de la fonction pour Tourner

5 Le Balayage

La phase de balayage est une étape essentielle dans le fonctionnement du robot puisqu'elle permet de détecter la présence de l'objet cible dans son environnement. Le sonar monté sur le servo-moteur permet au robot de balayer une zone spécifique en émettant des impulsions ultrasonores et en mesurant le temps qu'il faut pour que ces impulsions soient réfléchies vers le capteur. En analysant les données de temps de retour, le robot peut déterminer la présence et la distance d'objets dans son environnement. Cette capacité de balayage permet au robot de se déplacer efficacement et de réagir aux changements de son environnement.

5.1 Utilisation du SONAR

Le capteur sonar utilisé dans ce projet est le LV-MaxSonar-EZ24. Ce capteur fonctionne en émettant un signal sonore et en mesurant le temps qu'il faut pour que le signal soit réfléchi vers le capteur. Ce temps de mesure est utilisé pour calculer la distance entre le capteur et l'objet qu'il a rencontré.

FIGURE II.16 – LV-MaxSonar-EZ24

FIGURE II.17 – Mesure distance SONAR

Ainsi, le fonctionnement du capteur sonar implique deux broches : le Trig sonar et l'Echo sonar. Premièrement, le Trig sonar émet un impulsion ultrasonore. Cette dernière est une onde sonore de haute fréquence inaudible pour les humains. Elle est générée en envoyant une courte impulsion électrique (50 ms) à la broche Trig du capteur sonar.

Deuxièmement, l'Echo sonar reçoit l'impulsion ultrasonore réfléchie et mesure le temps qu'il faut pour que l'impulsion revienne. Cela est fait en utilisant deux canaux sur la broche

Echo du capteur sonar. Le premier canal émet l'impulsion ultrasonore (remet donc le Timer à 0) et le deuxième canal écoute l'impulsion retournée (lit le front descendant). En mesurant la durée de l'impulsion sur la broche Echo (différence de temps entre celui du canal 2 et du canal 1), la distance à l'objet peut être calculée.

L'utilisation de deux canaux pour l'Echo Sonar est nécessaire puisque le capteur doit être capable d'émettre l'impulsion ultrasonore et d'écouter l'impulsion retournée en même temps. L'utilisation d'un seul canal pour les deux fonctions entraînerait un retard ou une interférence ce qui entraînerait des mesures de distance inexactes.

FIGURE II.18 – Algorigramme SONAR

On configure le TIMER 1 pour le SONAR tel que :

PSC	46-1
ARR	65535

TABLE 1 – Prescaler et ARR du TIMER 1 du SONAR

FIGURE II.19 – Réglages du TIMER1

5.2 Le Servo Moteur

Le servo-moteur DGServo 9g est un moteur électrique qui permet de contrôler la position angulaire d'une charge. Il est utilisé dans ce projet pour contrôler le mouvement

du sonar monté dessus. Le servo-moteur fonctionne en recevant un signal PWM (Pulse Width Modulation) qui détermine la position angulaire de la charge.

FIGURE II.20 – SGservo 9g

D'après le cahier des charges, la période du signal PWM doit être de 37.5 ms. Ainsi, la position angulaire de la charge est déterminée par le temps d'impulsion que l'on mesure. Par exemple, un signal d'1.5 ms correspond à une position angulaire de 0 degré, tandis qu'un signal de 2 ms correspond à une position angulaire de +90 degrés.

$$\boxed{T(\theta) = \frac{\theta}{180} + 1 \ et \ x = T(\theta) \cdot \frac{ARR}{T_{PWM}}} \tag{5}$$

avec :

— $T(\theta)$: Le temps d'impulsion à appliquer
— θ : la position angulaire du servo-moteur
— x : la valeur du CCR qui définit la largeur d'impulsion
— T_{PWM} : Durée totale d'une période du PWM (en ms)
Ainsi on obtient les valeurs suivantes 0°, 45° et -45° :

Degré	Valeur lisible par le CCR
0	2556
45	3408
-45	1703

TABLE 2 – Valeur lisible par le CCR en degré

ce qui donne donc pour 1° :

$$\boxed{x = 18.93} \tag{6}$$

5.3 Convertir les degrés

Afin que le qu'on puisse commander le servo-moteur en degré une fonction dont voici l'algorigramme est nécessaire :

FIGURE II.21 – Algorigramme qui convertit les degré en valeur lisible par le CCR

C'est tout simplement la relation linéaire qui relie x et la position angulaire.

On utilise également un autre 2 (TIMER 5) afin de faire le balayage du sonar toutes les 100 ms :

$$\boxed{PSC = \frac{80.10^6.100.10^{-3}}{2^{16}} \implies PSC = 123 - 1} \tag{7}$$

$$\boxed{ARR = \frac{80.10^6.100.10^{-3}}{123} \implies ARR = 65040} \tag{8}$$

FIGURE II.22 – Réglages du TIMER5

FIGURE II.23 – Algorigramme du Timer 5

Ainsi, la première condition de l'algorigramme vérifie si le balayage est en cours. Si c'est le cas, le code effectue les opérations suivantes :

— Si c'est le début du balayage, la distance minimale est réinitialisée à 1000 et l'angle minimal est réinitialisé à 0 (expliqué plus tard, voir III.2.3).
— Le code calcule ensuite la distance minimale mesurée par le sonar. Si la distance minimale est supérieure à la distance mesurée actuellement, la distance minimale est mise à jour avec la distance mesurée actuelle et l'angle minimal est mis à jour avec l'angle correspondant à la position actuelle du servo-moteur.
— Le code fait ensuite tourner le servo-moteur pour effectuer le balayage. Si la position actuelle du servo-moteur est inférieure à 3408 (correspond à 45°, voir Table 2 pour plus d'explication), la position est incrémentée de 50 (i.e. 2.64° environ d'après ce qui suit) et la nouvelle position est envoyée au servo-moteur. Si la position actuelle est supérieure ou égale à 3408, la position est réinitialisée à 1703 et le balayage est arrêté en réglant la variable "balayage" à 0.
— Enfin, le code calcule l'angle correspondant à la position actuelle du servo-moteur à l'aide de la fonction "PtToDeg".

6 Le Calibrage

L'étape de calibration est une étape importante dans le fonctionnement du robot. En effet, elle permet de régler les roues du robot pour qu'il soit aligné en face de l'objet cible. Pour ce faire, le robot effectue une mesure de distance toutes les 0.25 ms en utilisant le capteur sonar.

6.1 Mesure d'angles

Selon le degré mesuré sur le sonar, le robot tourne un certain temps proportionnel au nombre de périodes de 1 ms nécessaires.

On a réglé le robot tel qu'il tourne lors de la calibration à sa vitesse maximale :

Degré	seconde
360	4.01857
y	$\frac{4.01857 \cdot y}{360}$

Table 3

Par exemple, si le robot doit tourner de y degrés, il tournera pendant $\frac{4.01857 \cdot y}{360}$ s. Il faut donc :

$$\boxed{\text{round}\left(\frac{4.01857}{360 \times 0.0005} \times |y|\right)} \tag{9}$$

périodes de 0.5 ms nécessaires pour calibrer le robot à l'objet (avec `round` l'entier le plus proche).

Ainsi, on obtient les réglages pour le TIMER7 :

$$\boxed{PSC = \frac{80.10^6.0,5.10^{-3}}{2^{16}} \implies PSC = 1 - 1} \tag{10}$$

$$\boxed{ARR = \frac{80.10^6.0,5.10^{-3}}{2} \implies ARR = 40000} \tag{11}$$

Figure II.24 – Configuration du Timer 7

FIGURE II.25 – Algorigramme du Timer 7

Tout d'abord, le servo-moteur est remis à 0°. Ensuite, la fonction "CommandeMoteur" est utilisée pour arrêter les moteurs.

Le code vérifie ensuite l'angle minimal mesuré lors du balayage précédent. Si l'angle minimal est inférieur à 0, cela signifie que l'objet est à droite du robot. Dans ce cas, la fonction "CommandeMoteur" est utilisée pour faire tourner les moteurs à une vitesse de 100% et la fonction "Tourner" est utilisée pour faire tourner le robot à droite. À l'inverse, si l'angle minimal est supérieur à 0, cela signifie que l'objet est à gauche du robot. Dans ce cas, le robot tourne à gauche avec les mêmes fonctions que pour tourner à droite.

La variable "c_calib" est elle incrémentée à chaque itération de calibrage (i.e. toutes les 0.5 ms).

Si la variable "calibrage" est à 0, le code arrête les moteurs en utilisant la fonction "CommandeMoteur" et réinitialise la variable "c_calib" à 0.

Enfin, si la variable "avance" est à 1, le code réinitialise la distance minimale et l'angle minimal à 0, utilise la fonction "Avancer" pour faire avancer le robot à une vitesse d'environ 20 cm/s et réinitialise la variable "y_min" à 0.

6.2 Calcul d'erreurs

Cependant, avec cette méthode, il y a une erreur dans le mouvement du robot qui peut être calculée et potentiellement corrigée pour assurer une navigation précise.

$$\boxed{\theta = \frac{0.5 \cdot 0.5ms \cdot 360}{4.01857s} = 0.022 \text{ degrés}} \tag{12}$$

FIGURE II.26 – Calcul de calibration

Ainsi comme,

$$\boxed{h = d \cdot tan(\theta) \approx 0.5mm} \tag{13}$$

On obtient si on laisse le robot avancer une erreur angulaire et de position de :

$$\boxed{\phi = arctan(\frac{20cm}{h}) \approx 89.86 \text{ degrés}} \tag{14}$$

$$\boxed{erreur_distance_robot = \sqrt{h^2 + (20cm)^2} \approx 20cm} \tag{15}$$

Ainsi, il n'y a théoriquement aucune erreur de position dû au TIMER 7 pour le calibrage.

7 La machine d'état

La machine d'état du robot est composée de cinq états : `ETAT_ATTENTE`, `ETAT_BALAYAGE`, `ETAT_CALIBRAGE`, `ETAT_AVANCER` et `ETAT_ARRET`. Le robot commence par l'état `ETAT_ATTENTE` et avance à travers les états en fonction des actions effectuées.

FIGURE II.27 – Algorigramme de la Machine d'état Fini

Avant chaque état, le robot vérifie si le bouton bleu est activé. Si c'est le cas, le robot passe à l'état suivant. À l'inverse, si le bouton bleu est désactivé, le robot reste dans l'état actuel.

Dans l'état `ETAT_ATTENTE`, le robot attend que le bouton bleu soit activé pour passer à l'état suivant.

Ensuite, dans l'état `ETAT_BALAYAGE`, le robot effectue un balayage de la zone en utilisant le capteur sonar monté sur le servo-moteur. Ainsi, le servo-moteur balaie entre -45 et 45 degrés et détermine si un objet est à une distance inférieure à 160 cm.

Dans l'état `ETAT_CALIBRAGE`, le robot ajuste les roues pour s'aligner en face de l'objet cible. Comme décrit précédemment, selon le degré mesuré sur le sonar, le robot tourne un certain temps proportionnel au nombre de périodes de 100 ms nécessaires.

Dans l'état `ETAT_AVANCER`, le robot avance jusqu'à une distance de 20 cm de l'objet cible. Si le robot ne détecte plus d'objet dans cet état, cela signifie qu'une erreur s'est produite dans le calibrage. Dans ce cas, le robot repart en état `ETAT_BALAYAGE` pour effectuer un nouveau balayage de la zone.

Enfin, dans l'état `ETAT_ARRET`, le robot s'arrête et attend que le bouton bleu soit activé avant de revenir en état `ETAT_ATTENTE`.

III Test du Robot

1 Comparaison avec les valeurs théoriques

1.1 Test des valeurs de la PWM des moteurs

Afin de vérifier que la PWM est bien génerée, nous avons branche une sonde de l'oscilloscope sur la pin **"PWM"** qui se trouve sur la carte électronique.
Sous chaque oscillogramme, nous pouvons lire la valeur de **"Duty cycle"** qui correspond au rapport cyclique observé.
Nous avons à chaque fois des valeurs très proches aux rapports cycliques que nous avons renseigné sur la fonction Avancer du robot, ce qui montre que notre PWM fonctionne parfaitement.

Figure III.1 – PWM à 20%

FIGURE III.2 – PWM à 50%

FIGURE III.3 – PWM à 70%

1.2 Distance sonar

Pour vérifier que le sonar détecte bien la distance d'un objet devant lui, nous avons mis en place un protocole :
Lancer une capture de distance du sonar en mode débug. Observer la variable "distance", qui correspond à la distance de l'objet en cm, à traver le SWV Graph Viewer.
Une fois commencé, on approche une main du capteur, et on observe l'évolution de la variable distance dans le graphique du SWV. On remarque que plus on rapproche la main, plus la variable distance diminue, ce qui est correct. Les valeurs des distances cor-

respondent bien après vérification à la règle, une tolérance de +/**-2cm** nous était donné dans le cahier des charges et nous sommes bien dans ces ordres là.

Nous remarquons cependant des irrégularités dans le graphique, des piques se trouvent apparaître alors que la main n'a pas bougé. Ceci peut s'expliquer par le manque de fiabilité de ce genre de capteur, ou par une perturbation de la mesure par des éléments extérieurs comme la poussière.

FIGURE III.4 – Variable distance en fonction du temps

2 Améliorations apportées

2.1 Filtre anti-rebond

La fonction check_button_state() vérifie l'état du bouton pour gérer le rebond :

(a) Algorigramme de check_button_state()

(b) Algorigramme du rebond du Bouton bleu

FIGURE III.5 – Algorigramme pour le filtre anti-rebond

Si debounce_flag est égal à 1 et que la différence entre le temps actuel (HAL_GetTick()) et debounce_time est supérieure à DEBOUNCE_DELAY, alors : debounce_flag est remis à 0, indiquant que le délai de rebond est passé.

La fonction HAL_GPIO_EXTI_Callback est un rappel appelé lorsqu'une interruption sur le GPIO est détectée, lorsqu'on appuie sur un bouton :

Si l'interruption provient du bouton bleu (GPIO_Pin == Bouton_Bleu_Pin) et que debounce_flag est égal à 0, alors debounce_flag est mis à 1 pour indiquer qu'un rebond potentiel est en cours et debounce_time est mis à jour avec le temps actuel pour commencer le délai de rebond.

Si le bouton est détecté comme étant appuyé :
(HAL_GPIO_ReadPin(Bouton_Bleu_GPIO_Port, Bouton_Bleu_Pin) == GPIO_PIN_SET), alors Start_value = 1 comme expliqué dans la partie concernant le Bouton.

2.2 Stratégie de déplacement

Insister ici sur le fait qu'on utilise que des Timer pour gérer le temps de calibration non pas des HAL_Delay ni de boucles while/for

2.3 Minimiser erreur calibration

Au lieu de balayer avec le sonar et d'essayer de trouver directement l'objet, nous avons décidé de faire un balayage complet et de garder la distance minimale. Une fois cette distance minimale obtenue, nous calibrons le robot afin qu'il se place en face, pour que nous n'ayons qu'a avancer sur l'objet sans tourner, ce qui minimisera l'erreur de position.

IV Conclusion

Pour conclure, notre contrat a été conçu et réalisé avec succès. Ainsi, il est bien capable de suivre un objectif en maintenant une distance constante de 20 cm, comme un caniche qui suit son maître. Cependant, il existe quelques évolutions que l'on peut évoquer.

1 Evolution possible du contrat

1.1 Filtrage Sonar

En effet, il existe des situations où le capteur sonar peut mesurer des distances de moins de 5 cm, ce qui ne devrait pas être le cas. Pour résoudre ce problème, il serait possible d'ajouter un filtre pour éliminer les mesures erronées.

1.2 Utilisation d'encodeurs pour la calibration

De plus, bien que le robot utilise actuellement un capteur sonar pour la navigation, il serait possible d'ajouter des encodeurs aux roues pour améliorer la précision de la calibration. Cela permettrait au robot de mieux suivre l'objectif et de réduire les erreurs de navigation.

2 Applications industrielles du contrat

Enfin, ce projet robot a des applications potentielles dans les divers domaines industriels suivants :

- Surveillance de Sécurité : Equipé de caméras et de capteurs, un robot chien pourrait patrouiller des propriétés privées ou des entreprises, détecter des intrusions, et alerter les propriétaires ou la sécurité.
- Détecteur de Fuites ou d'humidité : Un robot chien équipé de capteurs spécialisés pourrait détecter des fuites d'eau, ou d'autres substances dangereuses dans des environnements industriels ou domestiques, alertant les propriétaires ou les responsables de la sécurité en cas de problème. Il suffirait alors de l'embarquer d'un capteur d'humidité.
- Inspection de sites dangereux : Un robot équipé de caméras et de capteurs pourrait être utilisé pour inspecter des sites dangereux, tels que des centrales nucléaires, des mines ou des sites de stockage de déchets dangereux, réduisant ainsi les risques pour les travailleurs.

V Annexes

1 Annexe 1 : Code du main.c

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
    LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "math.h"
#include "stdlib.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum {
    ETAT_ATTENTE,
    ETAT_BALAYAGE,
    ETAT_CALIBRAGE,
    ETAT_AVANCER,
    ETAT_ARRET
} Etat;
/* USER CODE END PTD */

/* Private define ------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DEBOUNCE_DELAY 200
/* USER CODE END PD */

/* Private macro -------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------*/
ADC_HandleTypeDef hadc1;

```

```c
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim5;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;

/* USER CODE BEGIN PV */
int balayage; // permet de faire le balayage du sonar
int x;
int y;
int y_min;
int calibrage; // permet de faire le calibrage roues / sonar
int c_calib = 0; // Compteur de p riodes pour le calibrage des roues
int avance;
Etat etat_actuel = ETAT_ATTENTE;
uint32_t distance ;
uint32_t distance_min = 1000;
uint32_t valeur_sonar;
uint32_t pulse_length;
uint32_t T_Batt = 0;
volatile uint8_t Start_value = 0;
int debounce_flag;
int debounce_time;
uint16_t V_batt;
/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM5_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM6_Init(void);
static void MX_TIM7_Init(void);
/* USER CODE BEGIN PFP */
void SetServoPosition(TIM_HandleTypeDef *htim, uint32_t Channel, int
   angle);
void Tourner(char direction);
void CommandeMoteur(int percent);
void Avancer(int percent);
int PtToDeg(int x);
void GererEtat();
void check_button_state();
/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
```

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
   ----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
   Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM1_Init();
  MX_TIM5_Init();
  MX_ADC1_Init();
  MX_TIM6_Init();
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */
  balayage = 0;
  calibrage = 0;
  avance = 0;
  x = 1703;

  // Pour le SONAR
  HAL_TIM_Base_Start_IT(&htim1);
  HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_2); // Reception
  HAL_GPIO_WritePin(TRIG_GPIO_Port, TRIG_Pin , GPIO_PIN_SET); //Emission
    du SONAR
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4); // Servomoteur

  // On remet le servo moteur    0
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 2556);

  CommandeMoteur(0);

  // Pour les chenilles
  HAL_TIM_Base_Start(&htim2);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);

  //Pour le balayage
  HAL_TIM_Base_Start_IT(&htim5);
```

```
  //Pour le calibrage
  HAL_TIM_Base_Start_IT(&htim7);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
     if (Start_value){
       if (T_Batt >= 500){
         HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin,
   GPIO_PIN_RESET);
         HAL_ADC_Start(&hadc1);
         HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
         V_batt = (HAL_ADC_GetValue(&hadc1)*3.3)/4095;
         T_Batt = 0;

       }
    else{
      CommandeMoteur(0);
    }
    GererEtat();


     }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
   HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the specified
   parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
```

```
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK
   )
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_MultiModeTypeDef multimode = {0};
  ADC_AnalogWDGConfTypeDef AnalogWDGConfig = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
```

```
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc1.Init.OversamplingMode = DISABLE;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure the ADC multi-mode
  */
  multimode.Mode = ADC_MODE_INDEPENDENT;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analog WatchDog 1
  */
  AnalogWDGConfig.WatchdogNumber = ADC_ANALOGWATCHDOG_1;
  AnalogWDGConfig.WatchdogMode = ADC_ANALOGWATCHDOG_SINGLE_REG;
  AnalogWDGConfig.Channel = ADC_CHANNEL_15;
  AnalogWDGConfig.ITMode = ENABLE;
  AnalogWDGConfig.HighThreshold = 3723;
  AnalogWDGConfig.LowThreshold = 0;
  if (HAL_ADC_AnalogWDGConfig(&hadc1, &AnalogWDGConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_15;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
```

```
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_SlaveConfigTypeDef sSlaveConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 46-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 0xFFFF;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
  sSlaveConfig.InputTrigger = TIM_TS_TI1FP1;
  sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sSlaveConfig.TriggerPrescaler = TIM_ICPSC_DIV1;
  sSlaveConfig.TriggerFilter = 0;
  if (HAL_TIM_SlaveConfigSynchro(&htim1, &sSlaveConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_1) !=
   HAL_OK)
```

```
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
  sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
  if (HAL_TIM_IC_ConfigChannel(&htim1, &sConfigIC, TIM_CHANNEL_2) !=
   HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_4) !=
   HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
  sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
  sBreakDeadTimeConfig.Break2Filter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 1-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 20000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) !=
   HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_4) !=
   HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */
```

```
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM5 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM5_Init(void)
{

  /* USER CODE BEGIN TIM5_Init 0 */

  /* USER CODE END TIM5_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM5_Init 1 */

  /* USER CODE END TIM5_Init 1 */
  htim5.Instance = TIM5;
  htim5.Init.Prescaler = 123-1;
  htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim5.Init.Period = 65040;
  htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM5_Init 2 */

  /* USER CODE END TIM5_Init 2 */

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{
```

```
  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 2-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 40000;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief TIM7 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM7_Init(void)
{

  /* USER CODE BEGIN TIM7_Init 0 */

  /* USER CODE END TIM7_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM7_Init 1 */

  /* USER CODE END TIM7_Init 1 */
  htim7.Instance = TIM7;
  htim7.Init.Prescaler = 1-1;
  htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim7.Init.Period = 40000;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) !=
   HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM7_Init 2 */

  /* USER CODE END TIM7_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, DIR2_Pin|TRIG_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : Bouton_Bleu_Pin */
  GPIO_InitStruct.Pin = Bouton_Bleu_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Bouton_Bleu_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LED_Batt_Pin */
  GPIO_InitStruct.Pin = LED_Batt_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_Batt_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : DIR2_Pin TRIG_Pin */
  GPIO_InitStruct.Pin = DIR2_Pin|TRIG_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : Tension_Batt_Pin */
  GPIO_InitStruct.Pin = Tension_Batt_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Tension_Batt_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : DIR1_Pin */
  GPIO_InitStruct.Pin = DIR1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(DIR1_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

void Tourner(char direction) {
    if (direction == 'D' || direction == 'd') {
        // Direction droite
        HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, GPIO_PIN_SET);
    } else if (direction == 'G' || direction == 'g') {
        // Direction gauche
        HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, GPIO_PIN_RESET);
    } else {
        // Direction non reconnue, vous pouvez ajouter une gestion d'
    erreur ici
        return;
    }
}

void CommandeMoteur(int percent) {
    if (percent < 0 || percent > 100) {
        // Pourcentage invalide, on peut mettre une gestion d'erreur ici
        return;
    }

    // Calcul du rapport cyclique num rique (CCPR)
    uint32_t ccr_value = (htim2.Init.Period + 1) * percent / 100;

    // Mettre    jour les registres CCR des canaux PWM des moteurs
    gauche et droit
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, ccr_value);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, ccr_value);
}

void Avancer(int percent) {
```

```
    // Mettre les broches DIRD et DIRG    0 pour avancer
    HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, GPIO_PIN_SET);

    // Commande du moteur avec le pourcentage donn
    CommandeMoteur(percent);
}

int PtToDeg(int x){
  return ((int) (x-2556)/18.93);
}

// Fonction principale pour g rer les  tats
void GererEtat() {
    switch (etat_actuel) {
        case ETAT_ATTENTE:
            // Logique pour l' tat  d'attente
          if (Start_value){
            etat_actuel = ETAT_BALAYAGE;
          }
            break;

        case ETAT_BALAYAGE:
            // Logique pour l' tat  de balayage
          if (Start_value){
        balayage = 1; // Activer le balayage
        if (x >= 3408 && distance_min < 160) { //Balayage termin
          balayage = 0;
          etat_actuel = ETAT_CALIBRAGE;
        }
          }
            break;

        case ETAT_CALIBRAGE:
          if (Start_value) {
            calibrage = 1;
        if (c_calib >= round((4.01857/(360*0.001))*abs(y))) { // Le
    calibrage est termin
          calibrage = 0;
          etat_actuel = ETAT_AVANCER;
        }
          }
            break;

        case ETAT_AVANCER:
            // Logique pour l' tat  d'avancer
          if (Start_value){
        x = 1703; // On r initialise la valeur du SONAR
        avance = 1;
        if (distance > 160) {
          avance = 0;
          x = 1703; // On r initialise la valeur du SONAR
          CommandeMoteur(0); // On arr te d'avancer
          etat_actuel = ETAT_BALAYAGE;
        }
        if (distance <= 20) {
          avance = 0;
```

```
          etat_actuel = ETAT_ARRET;
        }
          }
            break;

        case ETAT_ARRET:
            // Logique pour l' tat  d'arr t
          if (Start_value){
            CommandeMoteur(0);
        // distance inf rieur    5 permet de debugger le robot (D s fois il calcule la distance    2cm)
        if (distance >= 22){
          etat_actuel = ETAT_ATTENTE;
        }
          }
            break;
    }
}

void check_button_state()
{
    if (debounce_flag == 1 && (HAL_GetTick() - debounce_time) > DEBOUNCE_DELAY)
    {
        debounce_flag = 0;
    }
}


void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == Bouton_Bleu_Pin && debounce_flag == 0)
  {
      debounce_flag = 1;
      debounce_time = HAL_GetTick();
  }

  if (GPIO_Pin == Bouton_Bleu_Pin)
  {
      if (HAL_GPIO_ReadPin(Bouton_Bleu_GPIO_Port, Bouton_Bleu_Pin) == GPIO_PIN_SET)
      {
        if (Start_value){
        //On remet    l' tat  initial le robot
            etat_actuel = ETAT_ATTENTE;
        CommandeMoteur(0);
        balayage = 0;
        calibrage = 0;
        avance = 0;
        x = 1703;
        // On remet le servo moteur    0
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 2556);
        }
          Start_value = !Start_value; // Basculer la variable Start_value entre 0 et 1


```

```
      }
  }
}


//Timer 5 utilis  pour la Calibration
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {

  if (htim->Instance == TIM5) {
    // On mesure les distances toutes les 100 ms
    if (balayage){
      // Si d but de balayage on RAZ la distance mininale
      if (x == 1703){
        distance_min = 1000;
        y_min = 0;
      }

      // Calcul de la distance minimale :
      if (distance_min > distance){
        distance_min = distance;
        y_min = PtToDeg(x);
      }
      //Faire le balayage du servo moteur
      if (x < 3408){
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, x);
        x += 50;
      }
      else {
        x = 1703;
        balayage = 0;
      }
      // On calcule l'angle de la distance mesur e
      y = PtToDeg(x);
    }
  }
  if (htim->Instance == TIM6)
  {
    T_Batt ++;
  }
  if (htim->Instance == TIM7){
    if (calibrage) {
            // Code de calibrage des roues

          // On remet le servo moteur    0
          __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 2556);
          CommandeMoteur(0);

          // On calibre les roues en fonction du pr c dent angle du servo (minimal)
          if (y_min < 0) {
            // Tourner    droite
            CommandeMoteur(100);
            Tourner('d');
          }
          else if (y_min > 0) {
            // Tourner    gauche
            CommandeMoteur(100);
```

```
            Tourner('g');
          }
          c_calib++;
        }

    else {
      CommandeMoteur(0);
      c_calib = 0;

    }

    if (avance){
      // ON RAZ la dist min
      distance_min = 1000;
      y_min = 0;
      Avancer(66); // Vitesse d'environ 20cm/s
    }
  }
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2){

    valeur_sonar = (uint32_t) HAL_TIM_ReadCapturedValue(&htim1,
   TIM_CHANNEL_2);
    distance =  (uint32_t) valeur_sonar/101;
    }
  }

void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    // Allumer la LED si le seuil est d pass
    HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin, GPIO_PIN_SET);
  }
}


/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
   state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Listing 4 – Code main.c