Mines Saint-Etienne

Rapport Projet Robot

Encadré par M. Acacio Marques

MINES Saint-Étienne

Institut Mines-Telecom

Par Yasser EL KOUHEN et Max DUBOIS

Avril-Juin 2024

Remerciements

Je souhaite exprimer ma profonde gratitude pour M. Acacio Marques, le responsable de l'UP Projet Robot et du GP Conception des Systèmes Electroniques 1, pour nous avoir encadré et conseillé tout au long de notre projet robot. Ainsi que pour avoir mis à notre disposition le laboratoire C102 pour pourvoir travailler le Main du code de notre projet robot - et les autres fonctionnalités demandés - en dehors des séances planifiés. Par ailleurs, nous voudrions plus particulièrement lui exprimer notre reconnaissance pour avoir pris en compte, dans la notation de la démonstration de notre robot, les différentes améliorations que nous avions apportées par rapport aux exigences premières de notre contrat, le contrat numéro 8.

Nous tenons à également témoigner notre reconnaissance à M. Adil Yacoubi et M. Yannick Marrieti pour les précieux conseils techniques donnés au début du projet Robot qui nous ont permis d'aborder le contrat robot n°8 sur une bonne lancée.

Sommaire

Table des matières

Tables des figures

1. Cahier des Charges et Spécifications

1.1. Introduction

Dans le contexte de l'UP Projet Robot, chaque groupe s'est vu assigné un contrat robot numéroté de 1 à 12. Chacun de ces contrats correspond à des objectifs et des fonctions spécifiques à implémenter dans un robot commun :

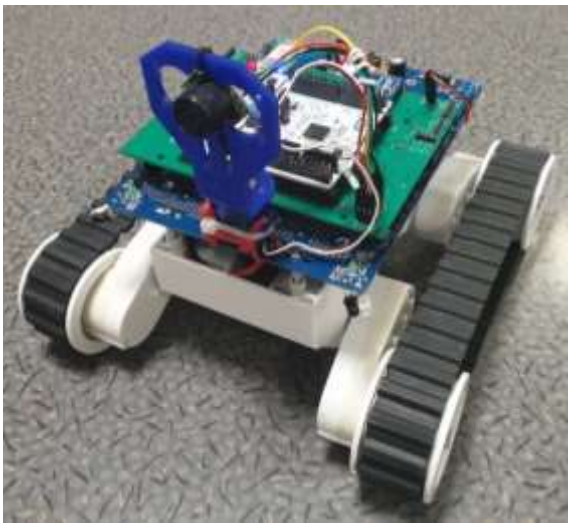

Figure 1 Robot utilisé pour le projet robot

Les nombreux contrats utilisent des caractéristiques spécifiques du robot mis à disposition parmi celles-ci-dessous :

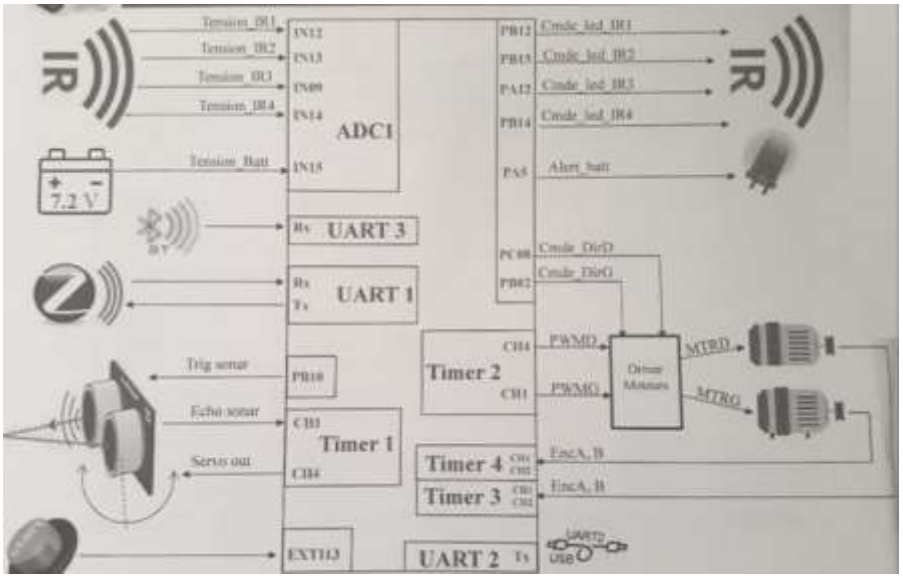

Figure 2 Synoptique du robot utilisé

1.2. Cahier des charges

Quant à nous, le contrat qui nous a été assigné est le numéro 8. L'objectif général de ce projet est de coder notre robot de telle sorte que : à la suite de la commande start, le robot via son sonar monté sur servo-moteur balaie l'espace à la recherche de son objectif. Une fois identifié et la distance entre l'objectif et le robot déterminé, le robot s'aligne et avance jusqu'à 20cm de celui-ci à une vitesse de 20 cm/s. Le tout avec une PWM des moteurs des chenilles de 5 KHz.

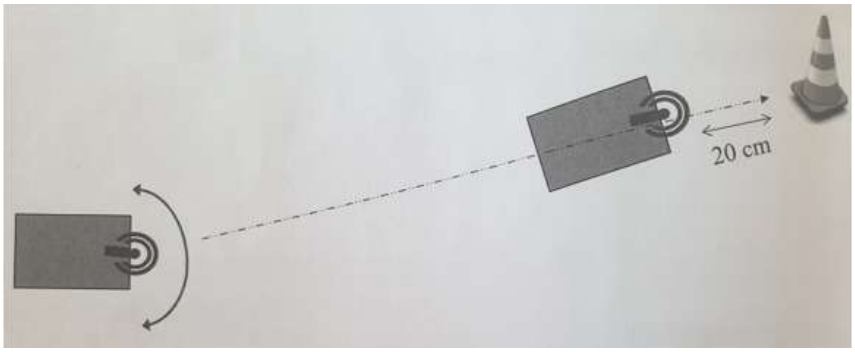

Figure 3 Trajectoire voulue du robot

En des termes plus concis, le tableau ci-dessous résume toutes les spécifications demandées :

Critères	Niveaux
Vitesse du robot	20 cm/s (± 1 cm/s)
Distance finale entre l'obstacle et le robot	20 cm (± 2 cm)
Vitesse angulaire	Vitesse maximale possible
Angle du servo-moteur par rapport au robot	0°
Fréquence du signal PWM	5 kHz
Envoi de la vitesse, phase et tension de la batterie	Transmission par UART
Surveillance de la batterie	A une fréquence d'au moins 20Hz
Start & Stop	Implémentation du bouton start & stop et usage d'un filtre anti-rebond

Figure 4 Cahier des Charges Contrat Robot n°8

2. Conception

2.1. Algorigrammes

2.1.1. Main

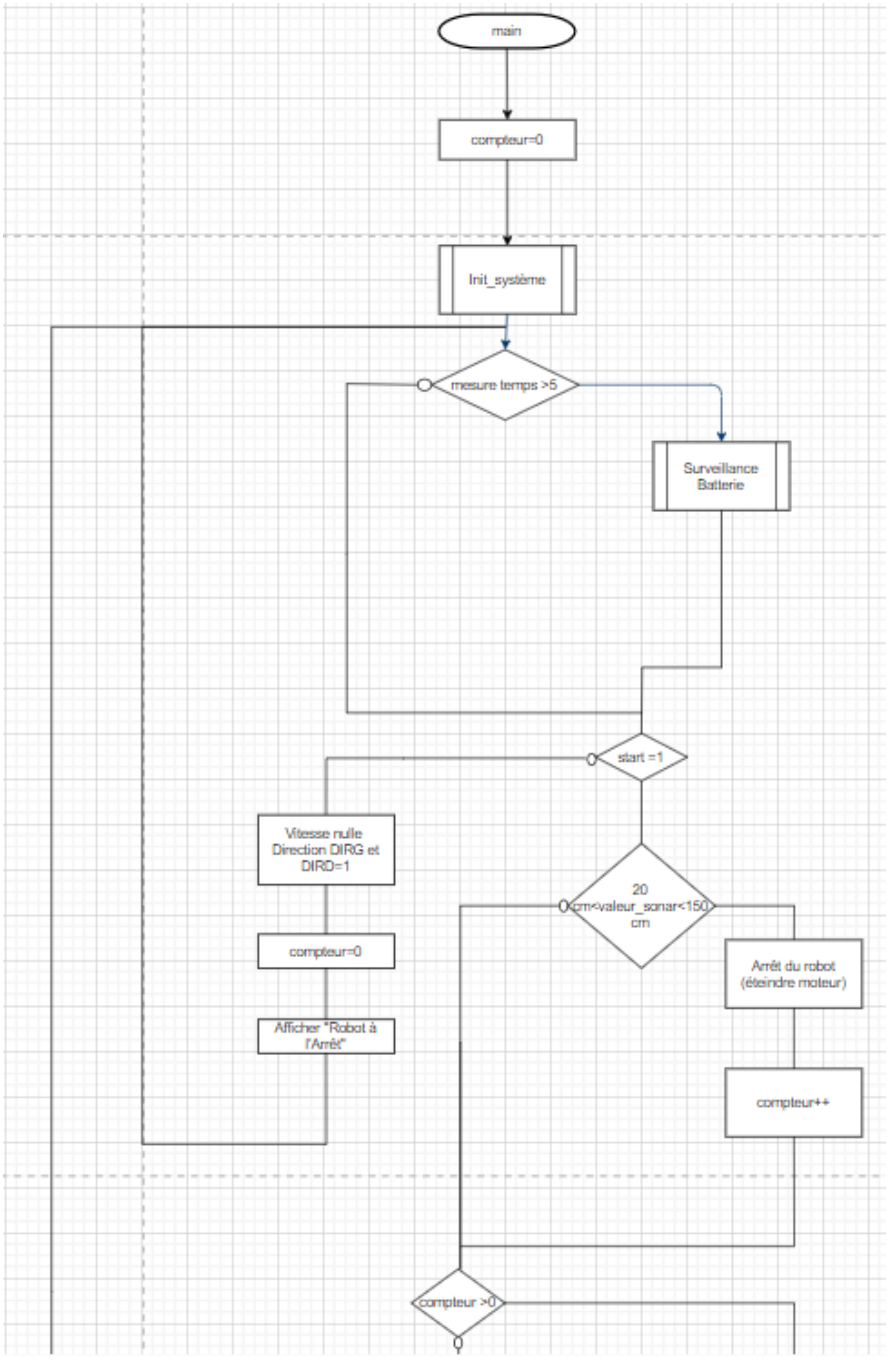

Figure 5 Algorigramme du Main - Partie 1

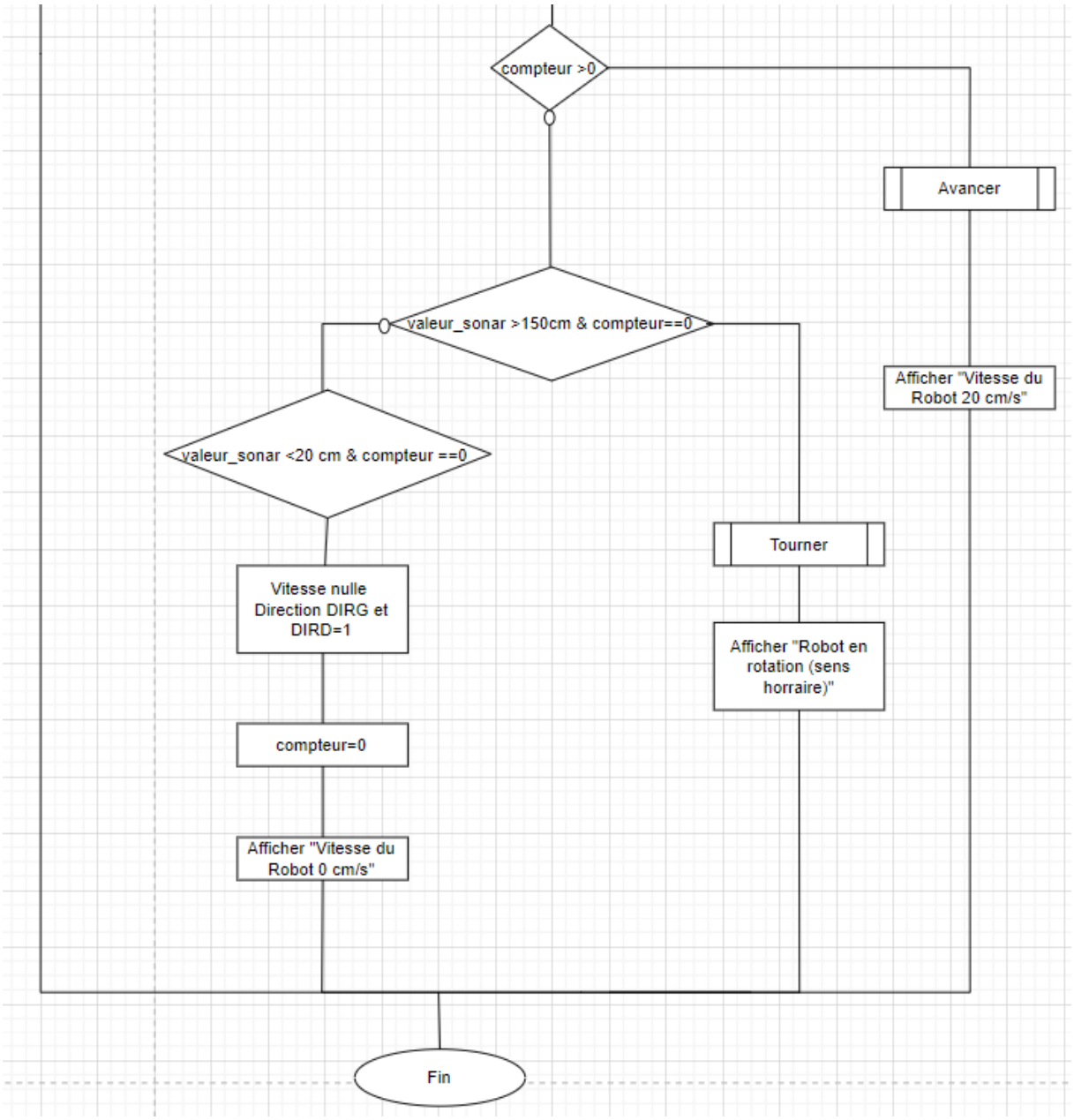

Figure 6 Algorigramme du Main - Partie 2

2.1.2 Interruptions et initialisations

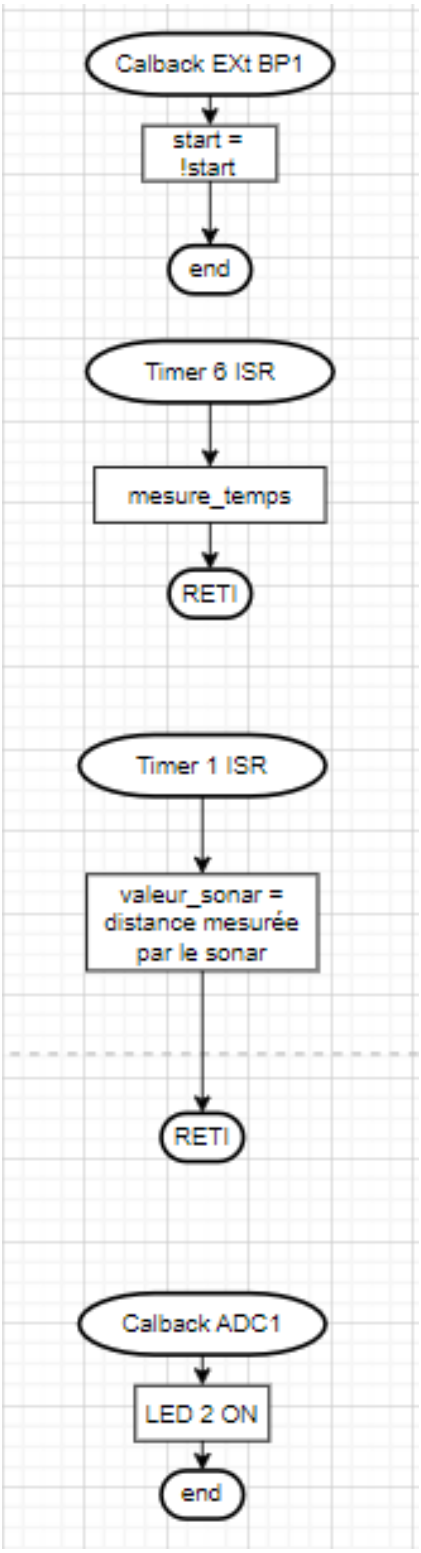

Figure 8 Algorigramme des différentes interruptions

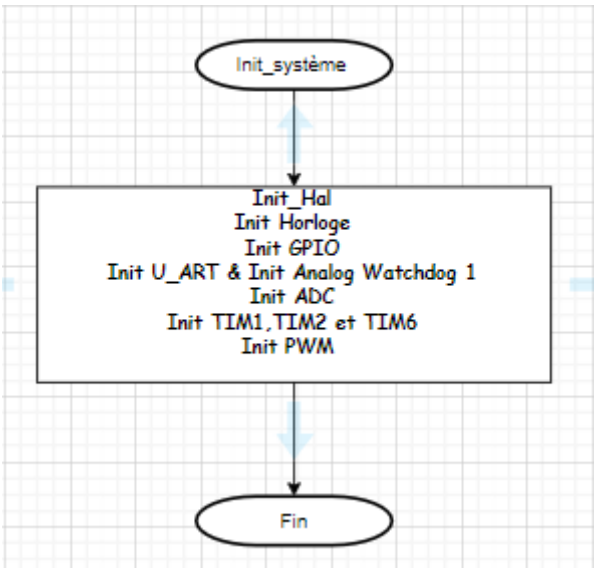

Figure 7 Algorigramme de l'Initialisation

2.1.3. Surveillance batterie

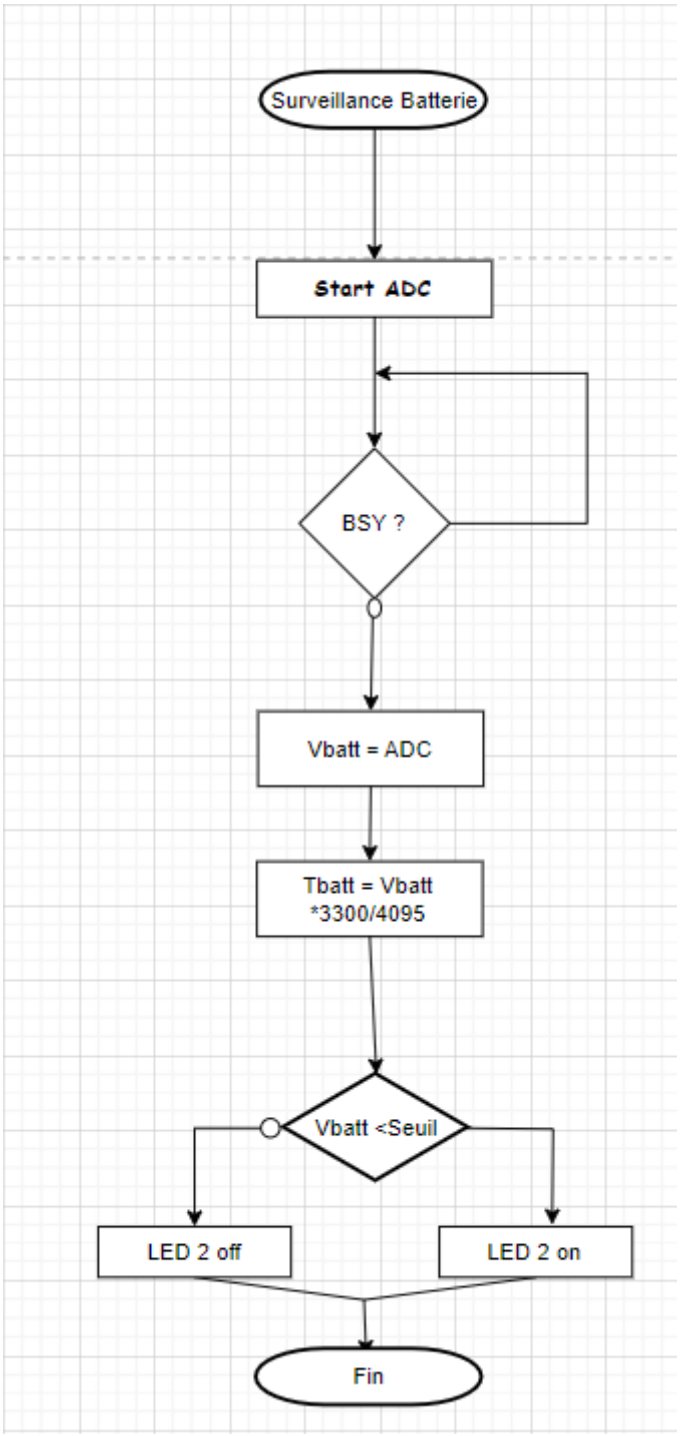

Figure 9 Algorigramme de la Surveillance batterie

2.1.4. Avancer

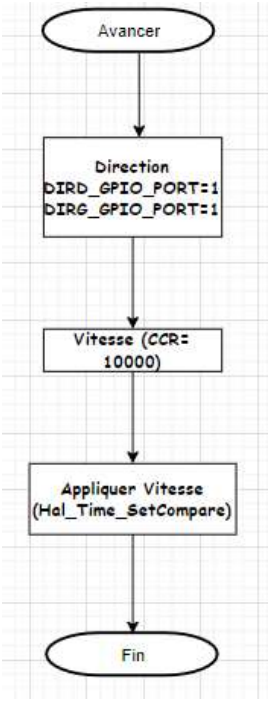

Figure 10 Algorigramme de la fonction Avancer

2.1.5. Tourner

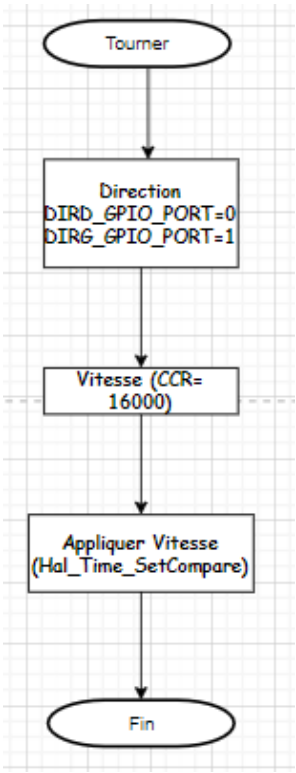

Figure 11 Algorigramme de la fonction Tourner

2.2. Configuration

2.2.1. Analog Watchdog

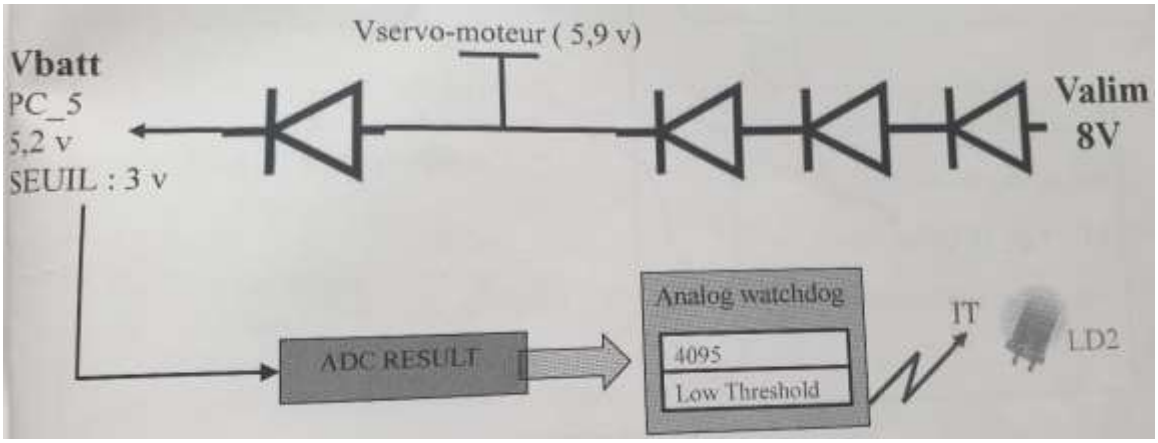

Figure 12 Configuration théorique de l'Analog Watchdog

L'analog watchdog est une fonctionnalité intégrée dans certains microcontrôleurs, notamment ceux de la famille STM32. Il est utilisé pour surveiller les niveaux de tension analogiques et générer une interruption si ces niveaux dépassent des seuils définis. Interruption qui se trouve être le callback :

```
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc)
```

Figure 13 Interruption Analog watchdog

Dans notre situation, nous utilisons l'analog watchdog pour surveiller si la batterie n'est pas déchargée. De manière plus pratique, la batterie est considérée déchargée lorsque sa tension est inférieure à 3V.

Figure 14 Configuration pratique de l'Analog Watchdog

En pratique, cette configuration définit une fenêtre délimitée par une limite basse (Low threshold) et une limite haute (High threshold). Et l'interruption Figure 11 se déclenche lorsque la valeur de la tension convertie par l'ADC est en dehors de cette fenêtre. Sachant que la relation entre la tension de la batterie et la valeur numérique de l'ADC est :

$$Vbatt(décimal) = \frac{Tbatt(V)}{3.3} \cdot 4095$$

Figure 15 Relation Tension ADC

A noter que dans la figure 12, le high threshold est à 4095 avec la relation figure 13 car l'ADC est sur 3.3 V et 12 bits. Donc la valeur maximale de l'ADC est de 4095 et correspond à toutes les tensions supérieures ou égales à 3.3V. Par le procédé de produit en croix utilisé figure 13, le low threshold est de 3723 car la batterie est considérée déchargée lorsque la tension Tbatt est inférieure à 3.3 V.

2.2.2 Servomoteur

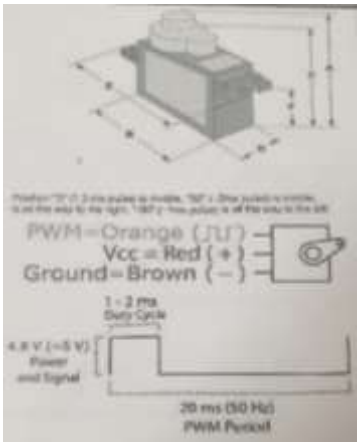

Figure 16 Fonctionnement d'un servomoteur de période 50ms

Le servomoteur permet de positionner le sonar à un angle spécifique. Un servomoteur reçoit un signal de contrôle sous forme de PWM (Pulse Width Modulation). Ce signal détermine la position du servomoteur. Comme on le voit figure 14, l'angle de rotation du servomoteur est contrôlé de telle manière que pour un angle de référence de 0°, le PWM a un rapport cyclique de 5% et pour un angle de 180° le rapport cyclique est de 10%.

En pratique, notre servomoteur est configuré de cette manière :

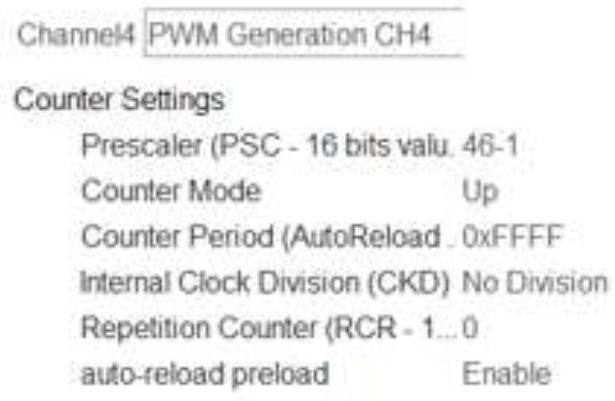

Figure 17 Configuration Pratique du servomoteur

Sachant que l'on a une fréquence d'horloge de la carte de fclock= 80MHz, une période du timer associé à notre PWM de 37.5ms, car on souhaite avoir un PWM de 37.5ms. Et que le timer associé à la PWM du servomoteur est le timer 1 qui a une résolution de 16 bits.

$$PSC = PES\left(fclock \cdot \frac{Ttimer}{2^{résolution}}\right)$$

- **PES : Partie Entière Supérieure**
- **PEI : Partie Entière Inférieure**

Figure 18 formule du préscalaire

D'après la formule du préscalaire (PSC) figure 17, on obtient bien le PSC de 46 présenté figure 16.

2.2.3 Sonar

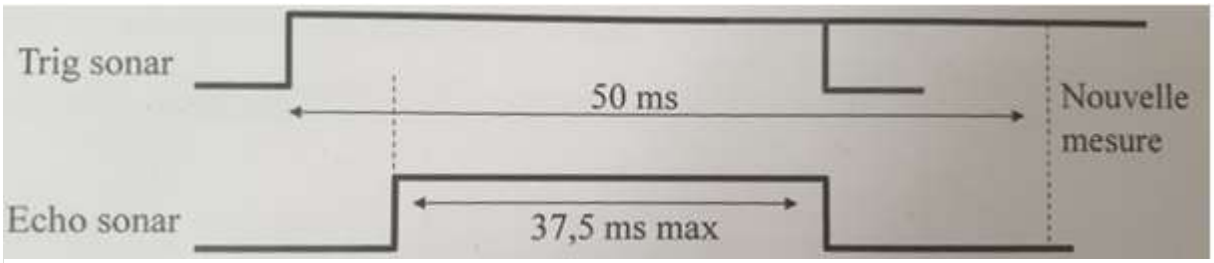

Figure 19 Fonctionnement du sonar

Le sonar permet de détecter et mesurer la distance avec un obstacle qu'il aurait capté. Lorsqu'un signal de déclenchement (Trig sonar) est envoyé au module, l'émetteur envoie une

brève rafale d'ondes ultrasonores à une fréquence spécifique. Echo sonar est le signal reçu lorsque le sonar reçoit les ondes réfléchies.

Dans notre projet, le sonar peut détecter un objet jusqu'à une distance maximale de 645 cm avec pour largeur d'impulsion maximale associée d'Echo sonar de 37.5 ms. Et l'émetteur du sonar via Trig sonar envoie la rafale d'ondes ultrasonores à une fréquence de 20 KHz (période de 50 ms).

D'autre part, on sait qu'un timer peut être compris comme étant un compteur qui s'incrémente à chaque période de PSC/fclock jusqu'à atteindre l'ARR puis se réinitialise à 0. Les interruptions se déclenchant lors de la réinitialisation du timer qui se fait toutes les $2^{\text{résolution}}$ x PSC/fclock. Partant de ce principe, la largeur d'une impulsion de Echo avec x, stocké dans la variable valeur_sonar, l'entier affiché par la fonction : $x \cdot \frac{PSC}{fclock}$

```
valeur_sonar = (uint16_t) HAL_TIM_ReadCapturedValue(&htim1, TIM_CHANNEL_2);
```

Figure 20 fonction lecture sonar

Etant donné que dans notre code, nous utilisons le timer 1 en mode Input Capture avec l'élément de configuration ci-dessous :

Figure 21 Timer 1 et le mode input capture

Il se trouve qu'utiliser la fonction figure 18 de cette manière permet d'initialiser le timer à 0 juste après avoir pris une capture de la valeur de IC1 (front montant du signal, TIM_CHANNEL_1), ce qui permet d'avoir la largeur d'impulsion de notre signal echo x en prenant une capture de la valeur de IC2 (front descendant du signal, TIM_CHANNEL_2).

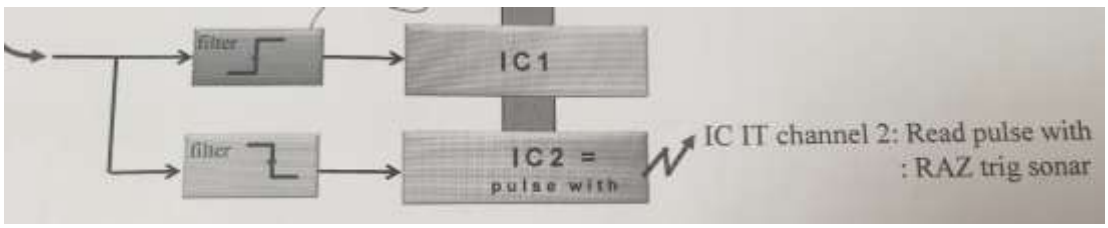

Figure 22 Input capture et channel associé

Par produit en croix, on trouve que la distance entre l'obstacle et le sonar est :

$$distance = x \cdot \frac{PSC}{fclock} \cdot \frac{645\ cm}{37.5\ ms} = x \cdot 0.00989\ cm = \frac{x}{101}\ cm$$

Figure 23 Relation de la distance captée par le sonar

2.4 Vue globale du MCU

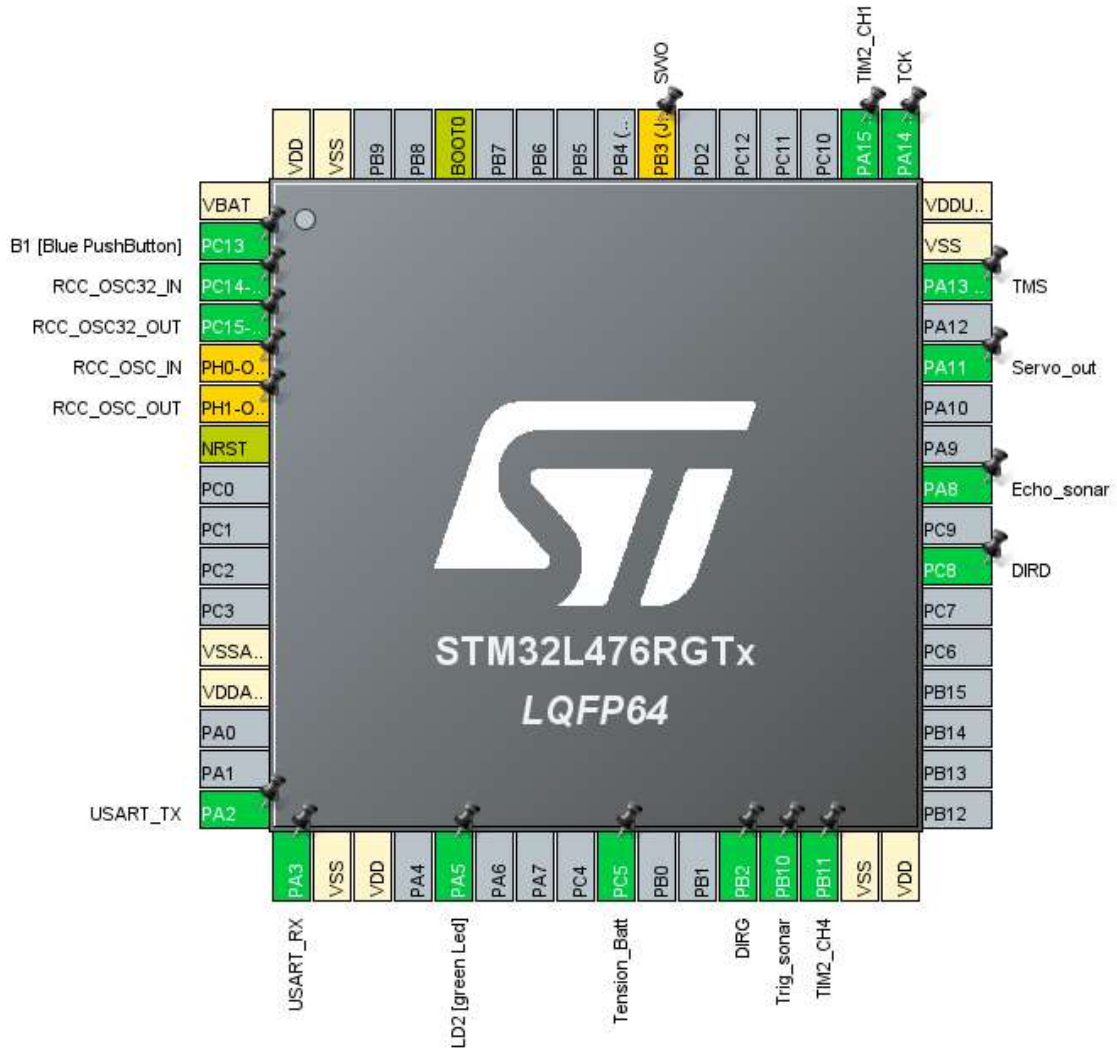

Figure 24 MCU de notre projet robot

A travers le MCU présenté ci-dessus, nous comprenons comment sont configurés les différentes pins permettant de réaliser les fonctions précédemment évoquées. A cela s'ajoute d'autres pins que nous avions configuré à d'autres fins :

- Les GPIO_output DIRG et DIRD qui permettent de contrôler respectivement le sens de rotation des chenilles gauches et droites du robot.

- Le GPIO_output LD2 qui permet de contrôler l'état de la LED associée à la surveillance de la batterie.
- Le GPIO_EXTI13 B1 qui permet de recevoir les changements d'états du bouton poussoir associé à la fonction de start & stop.
- Les pins USART_RX et USART_TX sont des broches utilisées pour la communication série via l'interface USART (Universal Synchronous/Asynchronous Receiver/Transmitter) respectivement pour la réception et la transmission des données séries via l'USART2 comme ci-dessous :

```
HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
```

Figure 25 Transmission de données séries via l'USART2

3. Réalisation/test

3.1. Comparaison entre valeurs théoriques et valeurs pratiques

Nous avons d'abord ajusté le CCR des 2 moteurs à 10000 de telle sorte à avoir un robot avançant à 20 cm/s. C'ést-à-dire un rapport cyclique de 62.5% car l'ARR est de 16000. Ce qui nous donne en arrêtant le robot toutes les 1s dans le code de la carte les mesures :

Temps (s)	**Position (cm)**
0	0
1	19.7
2	40.4
3	60.3
4	80.8
5	101
6	120.8
7	140.3
8	159.7
9	180.2

Figure 26 Tableau test de l'avancement du robot

Ce qui nous confirme bien que notre robot a une vitesse linéaire de 20 cm/s (± 1 cm/s) avec pour code de la carte, dans la boucle infinie :

```
255     //test moteur robot
256     /*avancer_test(20, 1); // 20 cm
257     HAL_Delay(5000);*/
```

Figure 27 Code test avancement robot

Numéro de l'essai	Position mesurée (cm)
0	19.4
1	18.6
2	19.3
3	20.0
4	19.9
5	18.2
6	19.1
7	19.8
8	18.5

Figure 28 Tableau test de l'arrêt du robot

D'autre part, il est également nécessaire de respecter le critère d'arrêt devant l'obstacle de 20 cm (± 2 cm). Partant de ce principe, il faut confirmer que cette exigence est respectée. Pour cela, nous avons simplement start le robot pour le premier essai puis stop et réinitialiser avant de le start pour l'essai suivant. Il ne restait plus qu'à mesurer la distance finale entre l'obstacle et le robot, ce qui a été fait ci-dessus. On peut voir que notre exigence d'arrêt est bien respectée avec le robot distant de l'obstacle de 20 cm (± 2 cm) à chaque essai.

Il est intéressant de noter qu'il n'y a aucune valeur supérieure à 20 cm car notre robot évaluant la distance avec l'objet de manière quasi-continue ne s'arrête qu'après que cette distance en question soit descendue en dessous de 20cm.

```
//test Sonar
//Calibrage du sonar
  /*for (int i=0;i<180;i++){
      set_servo_angle(i);
  }*/
```

Figure 29 Code calibrage servomoteur

Le dernier élément qu'il faut calibrer est le servomoteur, étant donné que chaque sonar a un angle de référence 0° différent, en effectuant un point d'arrêt à la ligne set_servo_angle(i) et en exécutant en mode debug. Le calibrage se fait visuellement en essayant d'incrémenter i jusqu'à aligner le sonar par rapport au robot.

Il est pertinent de souligner que la vitesse du robot doit être elle aussi calibrée avant l'utilisation d'un robot de sorte à respecter le critère de vitesse linéaire. Tandis que ça n'est pas le cas pour l'exigence de distance finale entre le robot et l'obstacle.

3.2. Chronogrammes

Pour garantir le bon fonctionnement des signaux PWM générés par notre microcontrôleur STM32, nous avons effectué une série de tests et de vérifications à l'aide d'un oscilloscope. Les signaux PWM sont essentiels pour le contrôle précis des actionneurs du robot, tels que les moteurs et les servomoteurs. En configurant l'oscilloscope pour mesurer à la fois la fréquence et le cycle de travail des signaux PWM, nous avons pu confirmer que les paramètres programmés dans le code correspondent aux spécifications requises. Les captures d'écran suivantes montrent les différentes configurations de PWM testées, illustrant la stabilité des signaux et la conformité aux valeurs attendues en termes de fréquence et de cycle de travail. Ces vérifications visuelles nous ont permis de valider empiriquement que les signaux PWM générés sont corrects et fiables pour une utilisation dans notre application robotique.

D'autre part, les timers jouent un rôle essentiel dans la génération des signaux PWM (Pulse Width Modulation) utilisés pour contrôler les différents éléments. Les timers sont configurés pour compter à une certaine fréquence, déterminée par l'horloge du microcontrôleur et le prescaler (PSC) fonction de la période du PWM, qui est également celle du timer. En mode PWM, le timer module la largeur des impulsions dans un cycle périodique, contrôlant ainsi la puissance moyenne délivrée aux composant. En ajustant le rapport cyclique de la PWM, nous pouvons contrôler le comportement de ces différents éléments.

De cette manière, dans notre situation :

- Tester la fréquence des PWM des chenilles correspond à tester la fréquence du timer 2
- Tester la fréquence du sonar correspond à tester la fréquence du timer 1

3.2.1. PWMD

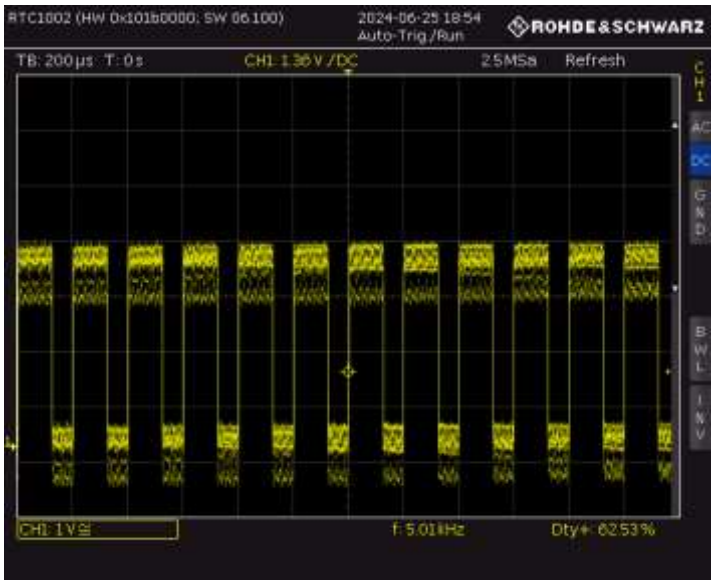

Figure 30 Signal PWM du moteur Droit (PWMD)

Comme on peut le voir, la PWM du moteur associée à la chenille droite du robot est bien de fréquence 5 kHz (± 0.2 %) comme on l'avait initialement configuré avec le timer 2 associé aux 2 moteurs de chenilles.

3.2.2. PWMG

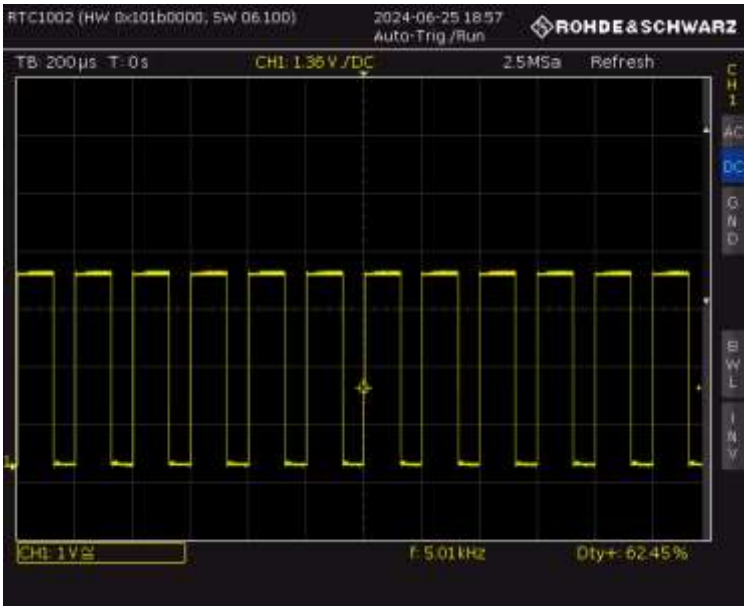

Figure 31 Signal PWM du moteur gauche (PWMG)

De même, La PWM du moteur associé à la chenille gauche du robot a également une fréquence

de 5 kHz (± 0,2 %), conformément à la configuration initiale réalisée avec le timer 2, qui est associée aux deux moteurs des chenilles.

3.2.3. Servo out

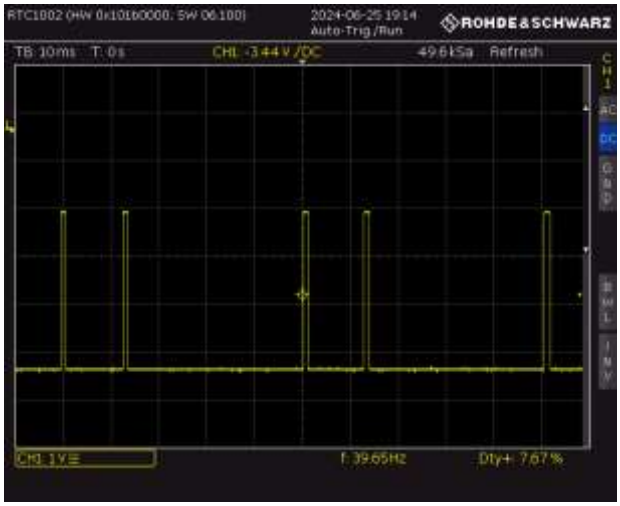

Figure 33 Signal PWM du sonar (Rapport cyclique bas)

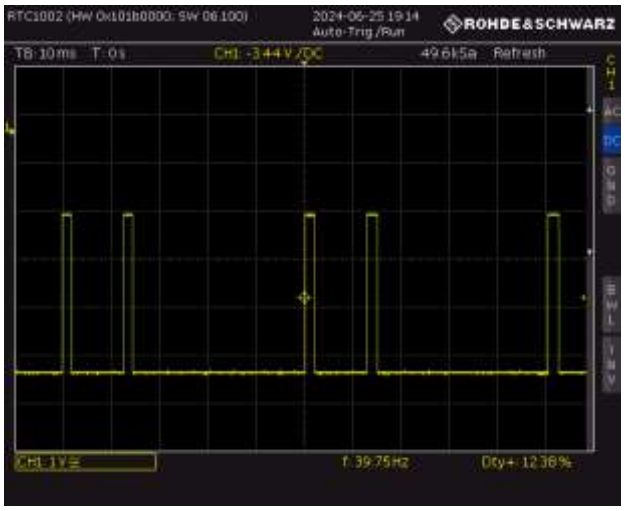

Figure 32 Signal PWM du sonar (Rapport cyclique haut)

En actionnant le servomoteur de cette manière dans la boucle infinie du code de la carte :

```
angle_robot++;
set_servo_angle(angle_robot);
if (angle_robot==180){
    angle_robot=0;
}
```

Figure 34 Code test servomoteur

On observe bien que le rapport cyclique de la PWM servomoteur est de 5% pour un angle de référence de 0° et de 10% pour un angle de 180°. Bien que dans la figure 28, le rapport cyclique est supérieur à 12%, en pratique on observe que le servomoteur s'immobilise pour des rapports cycliques supérieurs à 10% ou inférieurs à 5%.

De plus on semble avoir un rebond du signal qui pourrait s'expliquer par des interférences électriques ou une mauvaise connexion des fils ce qui perturbe le pic unique à une fréquence de 20 kHz recherché. 20 kHz étant la fréquence

3.2.4. Trig sonar et Echo sonar

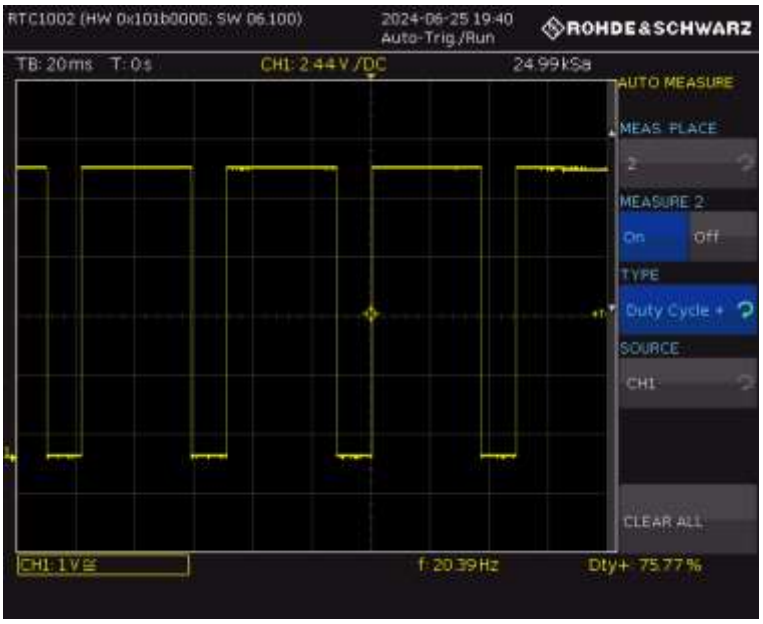

Figure 35 Signal Echo Sonar

Quant au sonar, On remarque que le signal Echo Sonar à l'oscilloscope a une fréquence de 20 kHz, ce qui correspond à la fréquence voulue de Trig Sonar figure 19. De plus, pour une distance avec l'objet supérieure à 645 cm, le signal affiché a un rapport cyclique de 75% par rapport à Trig Sonar, soit une période 50*0.75=37.5 ms, ce qui correspond également aux attentes théoriques de Trig sonar expliquées figure 19.

Rapport cyclique évoluant de manière proportionnelle à la distance comme on s'y attendait théoriquement. Avec par exemple un rapport cylique de de 5.2% par rapport à Trig sonar pour une distance à l'objectif de 45 cm :

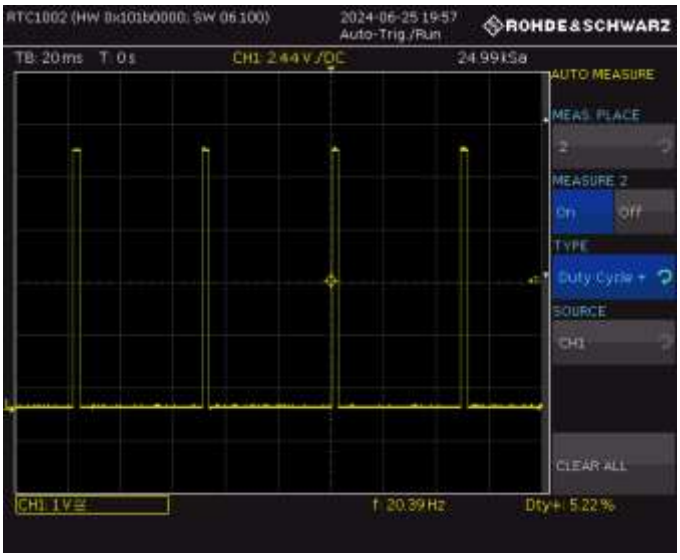

Figure 36 Signal Echo Sonar (objet à 45 cm)

3.3. Mesures via SWV

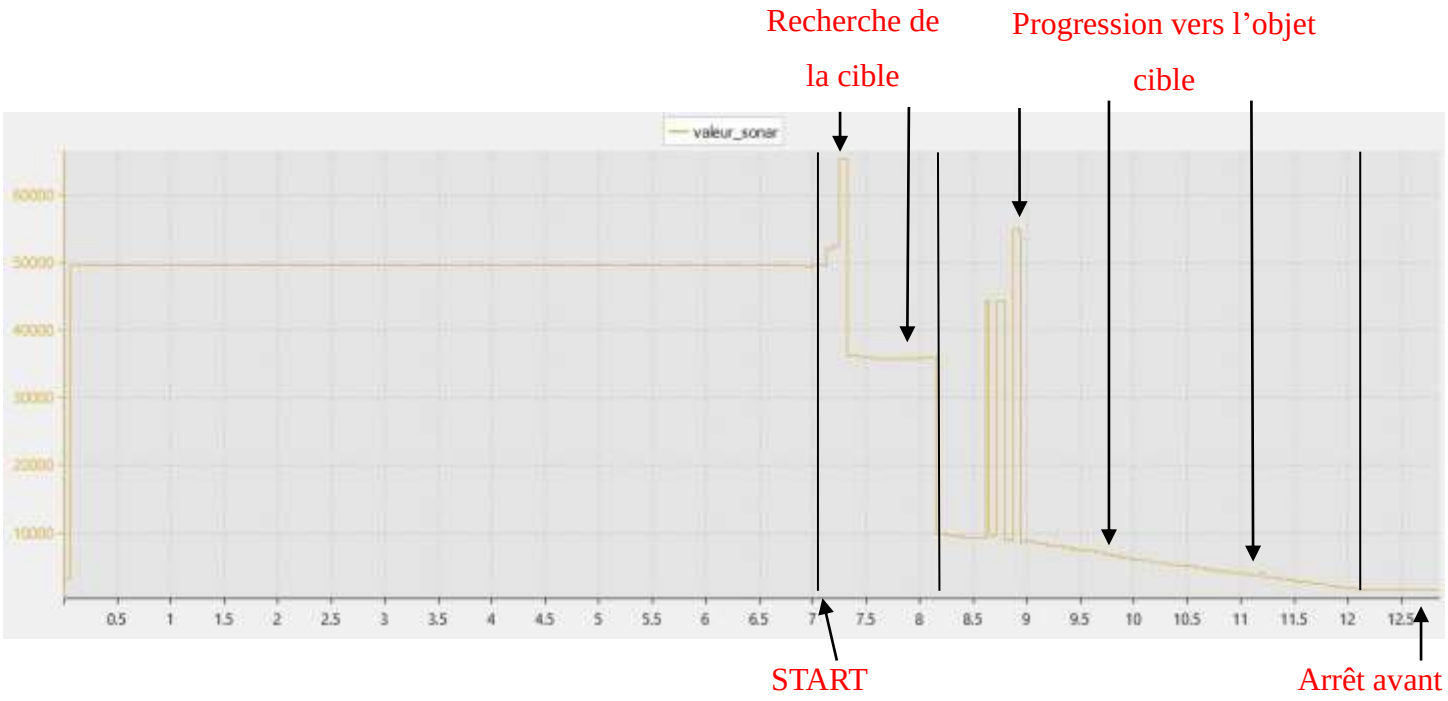

Figure 37 Graphique SWV du comportement de notre robot

A travers le graphique ci-dessus de valeur_sonar (101 fois la distance en cm entre le robot et l'objet détecté) en fonction du temps (en s), on comprend l'étendue du comportement de notre robot. Comportement qui se décrit selon plusieurs phases :

- **Avant le start :** Le robot ne fait que détecter la distance avec l'obstacle aligné le plus proche de lui et effectue d'autres fonctions intermédiaires qui ne sont pas le cœur du projet, comme la surveillance de la batterie. On le voit très bien avec une distance à l'obstacle constante.
- **Recherche de la cible :** Une fois le sonar calibré et aligné par rapport au robot, et le robot démarré grâce au START&STOP. Pour détecter un obstacle autour de lui, le robot effectue une rotation dans le sens horaire avec ses chenilles. Puis s'arrête de tourner lorsqu'il détecte un obstacle à **moins d'1m50**. On le remarque par les différentes valeurs que prend valeur_sonar et les différentes (quasi) discontinuités présentes. Discontinuités issues du fait que le robot détecte des objets différents.
- **Progression vers l'objet cible :** Une fois notre objet présent à moins d'1m50 détecté. Le robot s'approche à une vitesse quasiment constante de l'obstacle. Vitesse que l'on

peut déterminer par la pente de la courbe (ici 25 cm/s au lieu de 20 cm/s car le robot utilisé pour les mesures SWV n'a pas été calibré en vitesse au préalable). Les pics présents dans cette phase seront expliqués dans la suite car témoignent de notre réponse face aux difficultés rencontrées et aux améliorations souhaitées.

- **Arrêt avant obstacle :** Une fois le robot proche de l'obstacle, comme exigé dans le cahier des charges, on voit que le robot s'arrête à 20 cm en obstacle. Distance issu initialement d'un fonctionnement en boucle ouverte et qui est maintenant le résultat du robot asservi en position. En d'autres termes, après améliorations, le robot asservi en position fait en sorte de continuellement resté à 20 cm de l'obstacle.

Il est important de noter que dans l'utilisation de notre robot, nous avons délimité la zone de détection à un rayon d'1m50 du robot pour faire en sorte que le robot détecte bien l'objet cible voulu et ne soit pas perturbé par un obstacle à une distance de 4 m par exemple.

3.4. Résultats obtenus

Notre projet de robot a été conçu et développé pour répondre aux spécifications détaillées dans le cahier des charges. Après une série de tests rigoureux, nous avons obtenu les résultats suivants, qui confirment que notre robot respecte les exigences :

- **Vitesse du robot** : La vitesse moyenne du robot a été mesurée à 20 cm/s, avec une précision de ± 1 cm/s, ce qui est en conformité avec les spécifications.
- **Distance finale entre l'obstacle et le robot** : Le robot s'arrête à une distance de 20 cm de l'obstacle, avec une tolérance de ± 2 cm, respectant ainsi les critères définis.
- **Vitesse angulaire** : Le robot atteint la vitesse angulaire maximale possible, à l'aide d'un rapport cyclique de 100%, optimisant ainsi son temps de réponse et sa maniabilité.
- **Angle du servo-moteur par rapport au robot :** Le servo-moteur a été réglé à un angle de 0° par rapport au robot, assurant une détection précise et fiable de l'obstacle par le sonar et un alignement simple du robot et de l'obstacle avec le sonar.
- **Fréquence du signal PWM** : La fréquence du signal PWM utilisée pour les moteurs est fixée à 5 kHz (± 2%), assurant un contrôle précis de la vitesse et du positionnement.

- **Transmission des données** : Les valeurs de vitesse, phase, et tension de la batterie sont correctement transmises via UART, permettant une surveillance et un contrôle en temps réel.
- **Surveillance de la batterie** : La tension de la batterie est surveillée à une fréquence de 20 Hz, assurant ainsi une surveillance constante de l'état de la batterie et la prévention des décharges profondes.
- **Start & Stop** : Un bouton start & stop a été implémenté avec succès, bien qu'un filtre anti-rebond n'a pas été réalisé pour garantir un fonctionnement fiable et sans erreurs.

En résumé, notre robot non seulement satisfait, mais dépasse dans certains cas les attentes du cahier des charges, démontrant une performance robuste et fiable. Ces résultats confirment la réussite de notre projet et la pertinence de nos choix techniques et méthodologiques. L'oubli du filtre anti-rebond a été l'unique manquement au cahier des charges. Bien que les améliorations apportées au robot, présentées ci-dessous, permettent d'ajouter une nouvelle dimension au robot, notamment pour le fonctionnement du système initialement en boucle ouverte.

3.5. Difficultés rencontrées et solutions apportées

Lors du calibrage de l'alignement du sonar qui précède l'utilisation du robot comme présenté figure 29, il est en réalité très difficile d'aligner parfaitement le sonar et le robot.

$$deviation_{robot}\,(cm) = deviation_{sonar}\,(degré) \cdot \frac{2 \cdot \pi \cdot distance_{parcourue}}{360}$$

Figure 38 Relation déviation sonar et déviation robot

Le problème qui se crée est qu'alors une simple déviation de 5° du sonar par rapport au robot crée un décalage de 13 cm à l'arrivée devant l'obstacle comme on peut le voir avec la formule ci-dessus valable pour des petits angles.

La solution a donc été un asservissement en position du robot afin qu'il ne dévie pas de sa cible initiale. Cet asservissement consistait à utiliser le sonar en continu pour détecter l'obstacle. Et dès que l'obstacle était perdu de vue, autrement dit la présence des **pics dans le**

graphique figure 37 lors de la phase de la progression vers l'objet cible, le robot tournait à droite (sens horaire).

Un nouveau problème s'est alors posé à nous. Si le robot perd de vue l'objet cible mais que cet objet se trouve à gauche de lui, il effectuera quasiment un tour complet pour se repositionner correctement. Cette solution, étant extrêmement peu efficace, crée en plus un comportement chaotique pour le robot asservi en position. Pour palier à cette issue, la solution logique est de faire en sorte que l'obstacle se trouve toujours à droite du robot lorsque le robot ne le détecte plus. En d'autres termes, le robot doit toujours au pire des cas avancer à gauche de l'obstacle. Ce qui se traduit par un sonar qui doit détecter l'obstacle à droite du robot. Dans un souci de parcours d'une distance minimale, cet angle vers la droite du sonar par rapport au robot doit être minime, comme on peut le voir pour le robot avec sonar calibré ci-dessous :

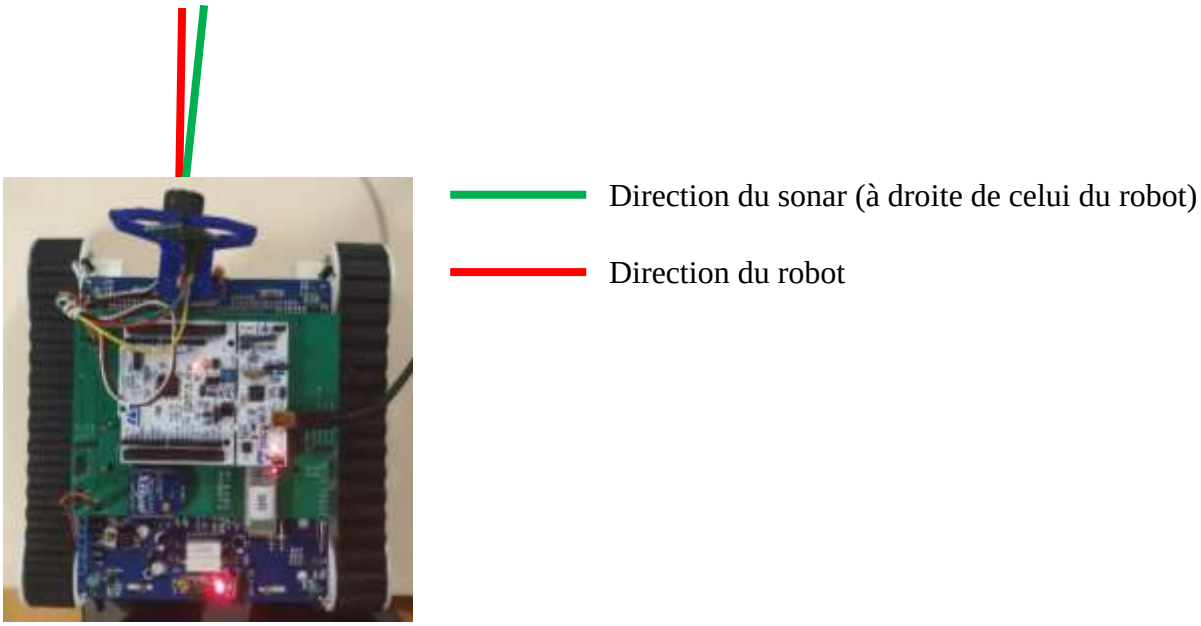

Figure 39 Correction de l'angle du sonar

Ainsi on passe d'une trajectoire initialement rectiligne à une trajectoire légèrement courbée vers la droite comme l'image ci-dessous (la trajectoire est bien moins courbée en réalité) :

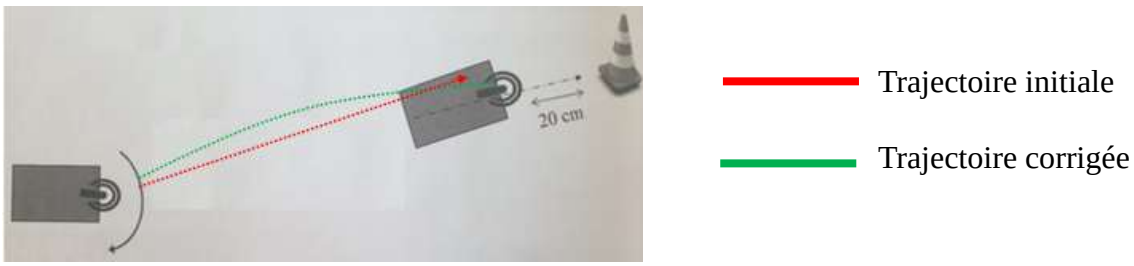

Figure 40 Trajectoire corrigée du robot

3.6. Améliorations apportées

À l'origine, ce projet visait à programmer notre robot de manière qu'après avoir reçu la commande de démarrage, il utilise son sonar monté sur un servo-moteur pour balayer l'espace à la recherche de sa cible. Une fois identifiée et la distance entre le robot et la cible déterminée, le robot s'aligne et avance vers celle-ci jusqu'à une distance de 20 cm, à une vitesse de 20 cm/s. Les améliorations portées ont été multiples :

- **Suivi de l'objet cible :** Une fois l'objet atteint par le robot, l'étape suivante est de faire en sorte que le robot suive la cible même si cet objet est en mouvement. C'est pourquoi nous avons implémenté cette fonction dans notre code à l'aide de la boucle infinie
- **Arrêt du robot lors de la rencontre d'un obstacle :** Si nous voulons nous assurer que l'objet atteint sa cible, il est nécessaire de prendre en compte les changements dans l'environnement du robot entre lui-même et son objectif. En ce sens, le robot doit pouvoir s'arrêter s'il rencontre un obstacle intermédiaire avant qu'il puisse atteindre sa cible. Dans ce but, le code du robot a été modifié afin que le robot s'arrête s'il rencontre un obstacle intermédiaire à moins de 20 cm.

L'implémentation de ces 2 fonctions a dû passer par un objet cible que l'on ne détectait qu'une fois, à l'aide du sonar, à un objet cible que l'on détecte en continu. En d'autres termes, nous avons dû passer d'un fonctionnement en boucle ouverte à un système asservi en position.

4. Conclusion

4.1. Evolutions possibles du contrat

Il est possible d'améliorer et d'étendre les fonctionnalités de notre robot pour le rendre plus polyvalent et performant. Parmi les évolutions envisageables, on peut inclure :

- **Implémentation d'un filtre anti-rebond pour le bouton start & stop :** Lorsqu'on souhaite démarrer le robot, il n'est pas rare de cliquer plusieurs fois sur le bouton poussoir, du fait de sa sensitivité, alors qu'on souhaite simplement appuyer une fois. Lors de ces situations, le robot a un comportement imprévisible, on ne sait pas si on a start ou stop. En d'autres termes, il n'est pas rare s'y reprendre à plusieurs fois pour start ou stop le robot. Implémenter un filtre anti-rebond permettrait au robot de ne prendre en compte que le premier appui et ainsi éviter à l'utilisateur de s'y reprendre plusieurs fois pour start ou stop le robot.
- **Suivi d'objets en mouvement :** Actuellement, le robot est capable de détecter et d'avancer vers une cible fixe et de la suivre si elle est en mouvement. Suivi possible tant que ne se trouve pas un objet intermédiaire entre le robot et sa cible que le robot suivrait au lieu de sa cible initiale. Il serait bénéfique d'implémenter un système avancé de suivi d'objets en mouvement pour des applications plus dynamiques. Système de suivi avancé qui se caractériserait par un robot qui suit sa cible tout en ne soyant pas perturbé par un objet intermédiaire qu'il finirait par suivre au lieu de sa cible initiale. De manière pratique, le robot ignorerait les objets détectés en dehors d'un certain rayon autour de sa cible.
- **Évitement d'obstacles :** Ajouter une capacité avancée d'évitement d'obstacles permettrait au robot de naviguer dans des environnements plus complexes sans intervention humaine. C'est-à-dire avoir la capacité de contourner un obstacle pour atteindre son objet cible.
- **Amélioration de l'autonomie :** Optimiser la gestion de l'énergie pour prolonger l'autonomie du robot, en particulier par une surveillance plus intelligente de la batterie.
- **Communication sans fil :** Utiliser les modules de communication sans fil (Wi-Fi, Bluetooth) pour une interaction à distance plus flexible et en temps réel.

4.2. Application industrielle possible de votre contrat

Les applications industrielles potentielles pour notre robot, une fois les fonctionnalités de base codées et validées, sont nombreuses :

- **Logistique et entrepôts** : En ajoutant des composants permettant de transporter des poids, le robot peut être utilisé pour transporter des marchandises dans des entrepôts, en optimisant les trajets et en évitant les obstacles automatiquement.
- **Surveillance et sécurité** : Déployer le robot pour des missions de patrouille et de surveillance dans des zones industrielles, grâce à ses capacités de détection et de mouvement autonomes.
- **Agriculture** : Modulo l'ajout de capteurs spécifiques et l'implémentation d'un code additionnel d'identification des anomalies. Le robot peut être adapté pour surveiller les cultures, détecter les anomalies, et naviguer entre les rangées de plantations.
- **Maintenance industrielle** : Utiliser le robot pour inspecter et diagnostiquer les équipements dans des environnements difficiles d'accès, en fournissant des données en temps réel sur l'état des machines, le tout grâce à l'ajout de certains capteurs spécifiques.

4.3. Ce que nous avons retenu

Au terme de ce projet, nous avons acquis des compétences essentielles en programmation de systèmes robotiques. Nous avons appris les principes de fonctionnements et la programmation de capteurs comme les sonars et d'actionneurs comme les servomoteurs pour des tâches spécifiques. La mise en œuvre de boucles de contrôle pour le déplacement et l'arrêt du robot nous a permis de mieux saisir de manière pratique les principes de l'asservissement et de la commande en temps réel. La transmission des données via UART et la surveillance de la batterie nous ont sensibilisés à l'importance des communications et de la gestion de l'énergie dans les systèmes embarqués. L'implémentation des fonctionnalités demandées tout en respectant les spécifications nous a enseigné la rigueur et l'importance des tests dans le développement logiciel. Ces compétences nous permettront de mieux nous adapter à divers projets robotiques à venir, en fournissant des solutions efficaces et adaptées répondant aux exigences industrielles.

5. Code

```
/* 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 "stdio.h"
#include "string.h"
#include "stdlib.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */
```

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

/* USER CODE END PM */

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

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */

// A utiliser après la partie démarrage des timers
void tourner();
void avancer();
void avancer_test(float avancement,int sens);
void set_servo_angle(float angle);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
*/
/* USER CODE BEGIN 0 */
#define NB_char 60
char msg[NB_char];
char msg2[NB_char];

#define SERVO_MIN_PULSE_WIDTH 670
#define SERVO_MAX_PULSE_WIDTH 4030
```

```
volatile uint16_t CCR;
volatile uint16_t Vbatt;
volatile uint16_t Tbatt;
volatile uint16_t distance;
volatile uint16_t angle_robot;

volatile uint16_t valeur_sonarF;
volatile uint16_t valeur_sonar;
volatile uint16_t compteur=0;

volatile uint16_t avancement=0;
volatile uint16_t temps=0;
volatile uint16_t start=0;
volatile uint16_t mesure_temps=0;


void set_servo_angle(float angle)
{
  uint16_t pulse_width = SERVO_MIN_PULSE_WIDTH + ((SERVO_MAX_PULSE_WIDTH -
SERVO_MIN_PULSE_WIDTH) * angle) / 180;
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, pulse_width);
}


void tourner() //a utiliser apres la partie demarrage timers

{
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
  HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  CCR=16000;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

void avancer() //a utiliser apres la partie demarrage timers

{
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
  HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  CCR=10000;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

void avancer_test(float avancement,int sens) //a utiliser apres la partie
demarrage timers
// temps en ms, avancement en cm, avancer : sens=1 sinon reculer : sens=-1
{
  CCR=10000;
```

```
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);

  int temps = 5000 * avancement/100 ;

  if(sens==1){
    HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
    HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
  }
  else if (sens==-1){
    HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_RESET);
  }

  HAL_Delay(temps);

  CCR=0;
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,CCR);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,CCR);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
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_USART2_UART_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_TIM1_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */

  // démarrage des timers

  // Pour le SONAR
  HAL_TIM_Base_Start(&htim1);
  HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_2); // Reception
  HAL_GPIO_WritePin(Trig_sonar_GPIO_Port, Trig_sonar_Pin , GPIO_PIN_SET);
//Emission du SONAR
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_Base_Start_IT(&htim6);

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
  // fin de démarrage des timers

  //test Sonar
  //Calibrage du sonar
    /*for (int i=0;i<180;i++){
      set_servo_angle(i);
    }*/
  angle_robot=86; //regarder par rapport au capteur et non pas à la tête,
mettre sonar légèrement vers la droite dans le pire des cas
  set_servo_angle(angle_robot);
  HAL_Delay(1000);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
    {

  if(mesure_temps>=5){ //surveillance batterie toutes les 50 ms
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    Vbatt=HAL_ADC_GetValue(&hadc1);
    Tbatt=Vbatt*3300/4095;
```

```
    if(Vbatt==4095){
      sprintf(msg,"Tension de la batterie : au moins %d mV\r\n",Tbatt);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    else {
      sprintf(msg,"Tension de la batterie : %d mV\r\n",Tbatt);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }

    mesure_temps=0;
  }

    //test moteur robot
    /*avancer_test(100, 1);  //avancer d'1m en 5s puis attendre 5s
    HAL_Delay(5000);*/

    if(start){

    if (20*101<valeur_sonar && valeur_sonar<=150*101 && compteur==0){
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      compteur++;
    }

    if (compteur>0){
      avancer();
      compteur=0;
      sprintf(msg,"Vitesse du robot : 20 cm/s  \r\n");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }

    else if (valeur_sonar>150*101 && compteur==0) {
      tourner();
        compteur=0;
        sprintf(msg,"Robot en rotation (sens horaire) \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    else if (valeur_sonar<=20*101 && compteur==0){
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
      compteur=0;
      sprintf(msg,"Vitesse du robot : 0 cm/s \r\n");
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    }

    else{
```

```
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,0);
      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,0);
      HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin,GPIO_PIN_SET);
      HAL_GPIO_WritePin(DIRG_GPIO_Port, DIRG_Pin,GPIO_PIN_SET);
      sprintf(msg,"Robot à l'arrêt\r\n");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    /* 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_14;
  AnalogWDGConfig.ITMode = ENABLE;
  AnalogWDGConfig.HighThreshold = 4095;
  AnalogWDGConfig.LowThreshold = 3723;
  if (HAL_ADC_AnalogWDGConfig(&hadc1, &AnalogWDGConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_14;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_640CYCLES_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_ENABLE;
  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 = 5;
  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 = 5;
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
  */
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 = 16000;
  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 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 = 13-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 61538;
  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 USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_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_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, DIRG_Pin|Trig_sonar_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DIRD_GPIO_Port, DIRD_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : DIRG_Pin Trig_sonar_Pin */
  GPIO_InitStruct.Pin = DIRG_Pin|Trig_sonar_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 : DIRD_Pin */
  GPIO_InitStruct.Pin = DIRD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(DIRD_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 HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc)
{
  /* Prevent unused argument(s) compilation warning */
  if(hadc->Instance==ADC1){
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
  }
}


void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2){
      valeur_sonar = (uint16_t) HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_2);
      distance =  (uint16_t) valeur_sonar/101;
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM6){
    mesure_temps; //TIM6 de période 10 ms
  }
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == B1_Pin){
    start=!start;
  }
}

/* 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 */
```