MINES
Saint-Étienne

Projet ROBOT

GP - Conception des Systèmes Électroniques 1

CONTRAT 3

Baron Alexandra & Brière Yann || EI23

Table des matières

Table des figures

Introduction

L'objectif de ce projet est principalement de se familiariser avec les cartes STM32 et au logiciel STM32CubeIDE. Un objectif central est également le débug qu'il est indispensable de maîtriser pour travailler en autonomie.

Cahier des charges et spécifications

Dans le cadre du projet, notre binôme s'est vu attribuer le contrat numéro 3. L'objectif est le suivant : après la commande start, le robot doit obéir à 2 commandes Bluetooth de la tablette : avancer et tourner à droite. Le robot ne peut tourner que si la commande avancer a été faite. Un second appui sur avancer arrête le robot. Lorsque le robot avance, la commande tourner augmente l'angle de rotation de 45 degrés pour chaque appui. Un appui prolongé sur tourner fait tourner le robot sur place. Dans tous les cas de figure, le robot avance après la rotation. Pour l'arrêter il y a deux solutions : bouton d'arrêt d'urgence B1 ou un deuxième appui sur avancer.

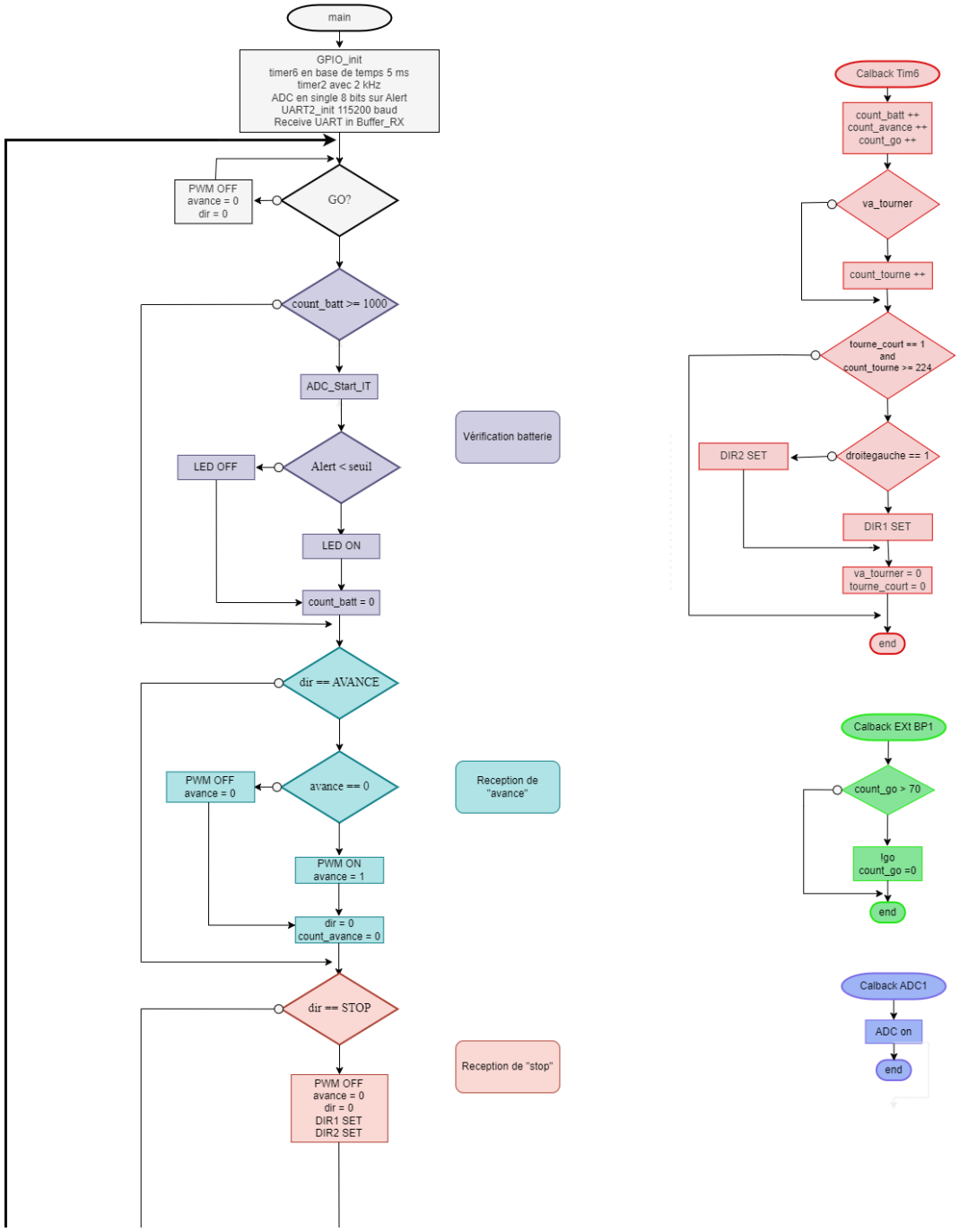
Pour notre contrat, le PWM est fixé à une fréquence de 2 kHz. La base de temps utilisé est de 5ms. Le robot doit avancer à une vitesse de 20 cm/s. La précision de l'angle de rotation est de +/- 5° et le robot fonctionne en boucle ouverte. L'horloge interne du microcontrôleur est fixée à 80 MHz. Pour notre contrat, nous avons uniquement besoin des 2 moteurs à courant continu, ainsi que du module Bluetooth pour communiquer avec la tablette.

De plus, pour piloter le robot nous avons dû installer l'application "Bluetooth RC Controller" sur une tablette. "Connect to car" permet d'appairer notre robot caractérisé par son numéro. "Settings" permet de voir le code ASCII envoyé.

Il existe également des attentes communes à tous les contrats. En effet, le robot ne peut avancer que si le bouton *go* (Bouton bleu B1) est activé. Ce bouton sert également d'arrêt d'urgence. Nous avons fait le choix d'utiliser un filtre anti-rebond qui était fortement recommandé pour ce bouton, afin d'assurer sa fiabilité lors d'un appui. De plus, la tension de la batterie doit également être surveillée de manière régulière pour détecter les chutes de tension à des valeurs inférieures à 3V. Pour cela, on utilise la valeur que l'ADC convertit et on fixe un seuil à ne pas dépasser. Si c'est le cas, la LED 2 s'allume.

Enfin, nous avons effectué quelques démarches supplémentaires au contrat de base car nous trouvions que des ajouts pouvaient être pertinents. Dans un premier temps, nous avons implémenté un filtre anti-rebond sur le bouton avancer. En effet, comme notre contrat stipule qu'un appui sur "avancer" devait arrêter le robot si celui-ci était en marche, les appuis n'étaient parfois pas fiables. Si on appuyait un peu trop longtemps, le robot n'obéissait pas forcément à la commande car il recevait deux fois la commande (s'il avançait, il continuait et pareil s'il était arrêté). Puis, nous avons implémenté un bouton d'arrêt sur la tablette. En effet, nous nous sommes dit qu'il était pratique de pouvoir arrêter le robot s'il avançait ou s'il tournait directement depuis la tablette. D'autant plus que cet ajout a été effectué avant le filtre anti rebond sur le bouton avancer. Ainsi, ce bouton nous permettait d'arrêter le robot quoi qu'il arrive. Cependant, il n'est pas totalement équivalent au bouton B1. En effet, il n'y a pas besoin de deuxième appui pour que le robot puisse fonctionner à nouveau. Ce bouton sert simplement à arrêter le robot. Enfin, le contrat demande que les commandes pour tourner de 45° et pour tourner tant que l'appui est enclenché soit faites vers la droite. Nous avons décidé d'ajouter également ces commandes pour le côté gauche afin que le robot soit plus maniable.

Conception


main
GPIO_init
timer6 en base de temps 5 ms
timer2 avec 2 kHz
ADC en single 8 bits sur Alert
UART2_init 115200 baud
Receive UART in Buffer_RX
GO?
PWM OFF
avance = 0
dir = 0
count_batt >= 1000
ADC_Start_IT
Alert < seuil
LED OFF
LED ON
count_batt = 0
Vérification batterie
dir == AVANCE
avance == 0
PWM OFF
avance = 0
PWM ON
avance = 1
dir = 0
count_avance = 0
Reception de "avance"
dir == STOP
PWM OFF
avance = 0
dir = 0
DIR1 SET
DIR2 SET
Reception de "stop"
Calback Tim6
count_batt ++
count_avance ++
count_go ++
va_tourner
count_tourne ++
tourne_court == 1
and
count_tourne >= 224
droitegauche == 1
DIR2 SET
DIR1 SET
va_tourner = 0
tourne_court = 0
end
Calback EXt BP1
count_go > 70
Igo
count_go =0
end
Calback ADC1
ADC on
end

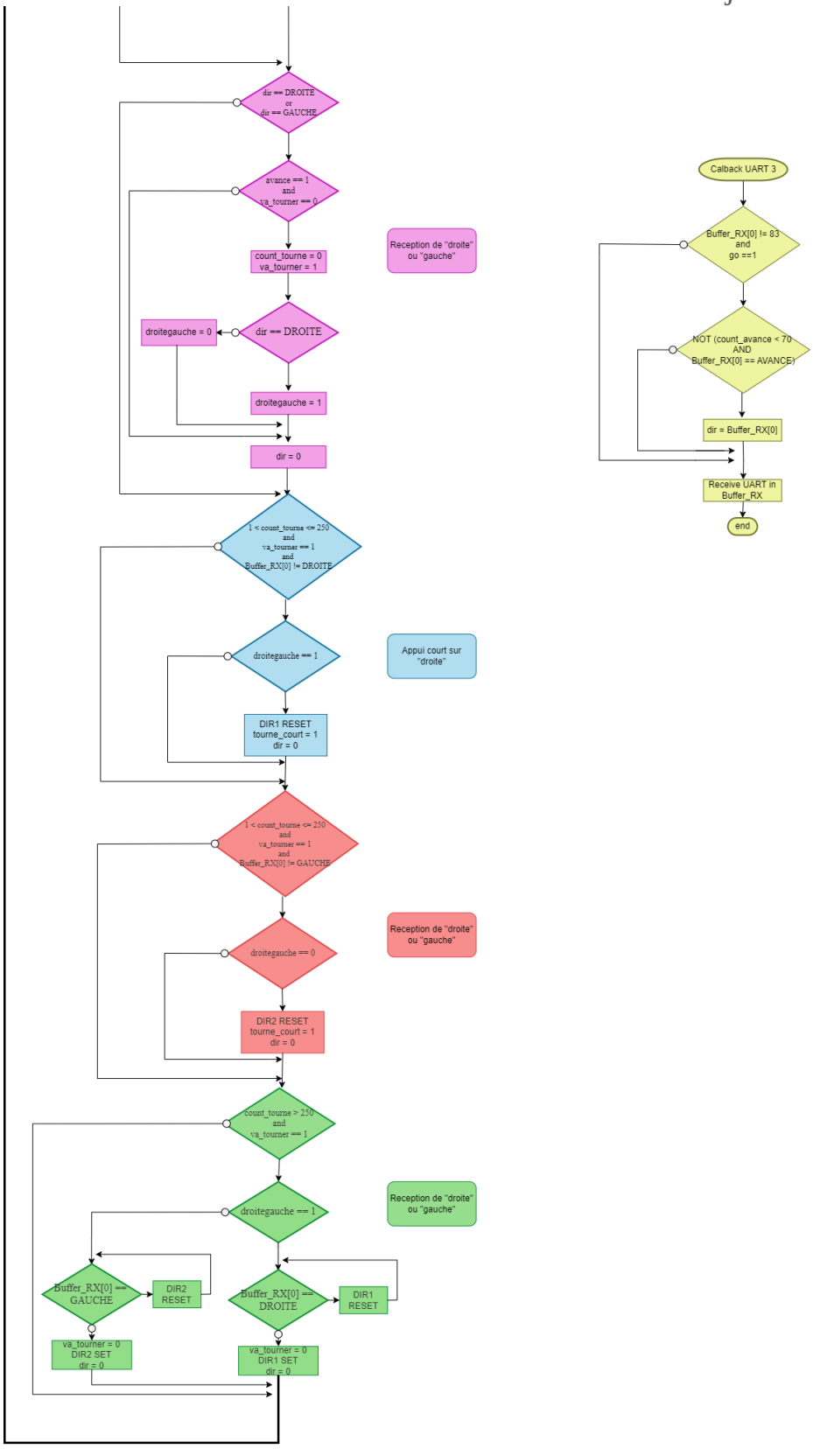


Figure 1 : Organigramme complet.

Notations

```
#define seuil 232

//Pour les réceptions de l'UART
#define AVANCE 70
#define STOP 66
#define DROITE 82
#define GAUCHE 76
```

Figure 2 : Extrait du code sur STM32. Définitions des constantes.

```
//Bouton de lancement et d'arret
volatile uint16_t go = 0;
volatile int count_go = 0;

//Surveillance batterie
volatile uint16_t Alert; //Valeur de tension de l'alimentation
volatile int count_batt = 0; //Compteur entre deux mesures de valeur de tension d'alimentation

//Pour l'avance
int avance = 0; //Etat d'avance du robot : vaut 0 si le robot n'avance pas, vaut 1 si il est en cours d'avance
volatile int count_avance = 0; //Compteur entre deux réceptions de "avancer".

//Reception Bluetooth
unsigned char Buffer_RX[2]; //Uart 3
volatile char dir; // Commande reçue par le robot, envoyée par la tablette, en dehors de "Stop".

//Pour tourner
volatile int va_tourner = 0; // Si le robot va tourner : 0 si non, 1 si oui
volatile int count_tourne = 0;//Compteur de temps après la première réception de droite ou gauche
volatile int tourne_court = 0; //Le robot doit tourner de 45 degrés. Appui rapide sur la tablette.

//Pour le sens de rotation
volatile int droitegauche = 0; // 1 si droite, 0 si gauche
```

Figure 3 : Extrait du code sur STM32. Définitions et Initialisations des variables.

Périphériques choisis

Timer 2

Le Timer 2 est utilisé pour générer des signaux PWM pour contrôler les moteurs du robot. Les canaux PWM sont configurés sur *TIM_CHANNEL_1* et *TIM_CHANNEL_4*. Les valeurs de comparaison sont réglées à l'aide de *__HAL_TIM_SET_COMPARE*, ce qui modifie le rapport cyclique de la PWM pour contrôler la vitesse des moteurs.

Notre contrat demande une fréquence de 2kHz. Pour y répondre, on calcule la valeur du Prescaler et de l'AutoReload Register.

$$PSC_{opti} \geq \left(\frac{Periode}{2^{16} * TimerCLK}\right) \geq \left(\frac{\frac{1}{2*10^3} * 80 * 10^6}{2^{30}}\right) \geq 3.7 * 10^{-5}. \text{ Soit PSC} = 1.$$

De même pour l'AutoReload Register,

$$\text{ARR} = \left(\frac{Periode}{PSC * TimerCLK}\right) = \left(\frac{\frac{1}{2*10^3} * 80 * 10^6}{1}\right) = 40000.$$

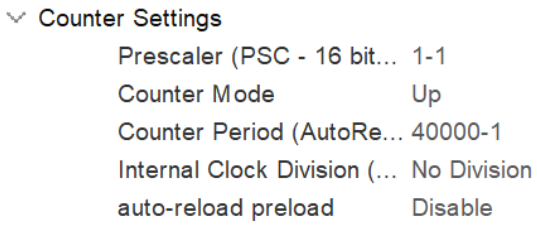


Figure 4 : Extrait de STM32. Configuration du Timer 2.

Timer 6

Le Timer 6 génère des interruptions périodiques utilisées pour gérer diverses tâches du système. Chaque interruption incrémente des compteurs utilisés pour surveiller l'état de la batterie, la durée de rotation, et le temps écoulé depuis la dernière commande reçue. La fonction de rappel *HAL_TIM_PeriodElapsedCallback* est appelée à chaque période. Nous avons choisi une période de 5 ms, base de temps énoncé dans notre contrat par ailleurs. Pour respecter la période choisie, nous avons réalisé les calculs nécessaires pour obtenir la valeur du Prescaler et de l'AutoReload Register.

$$\text{PSC}_{\text{opti}} \geq \left(\frac{Periode}{2^{16} * TimerCLK}\right) \geq \left(\frac{5 * 10^{-3} * 80 * 10^6}{2^{16}}\right) \geq 6.1 \text{ Soit PSC} = 7.$$

De même pour l'AutoReload Register,

$$\text{ARR} = \left(\frac{Periode}{PSC * TimerCLK}\right) = \left(\frac{5 * 10^{-3} * 80 * 10^6}{7}\right) = 57143.$$

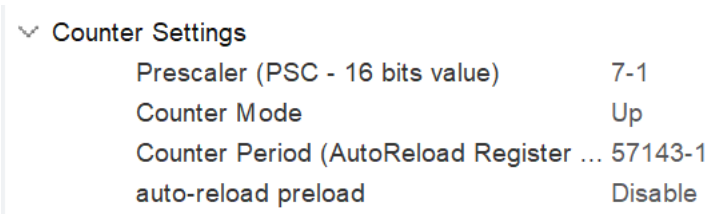


Figure 5 : Extrait de STM32. Configuration du Timer 6.

ADC 1

L'ADC 1 mesure la tension de la batterie du robot. La résolution de l'ADC est configurée à 8 bits. En choisissant cette résolution, on a un quantum de 12mV, ce que nous avons considéré suffisant pour notre utilisation. Lorsqu'une conversion ADC est terminée, la fonction de rappel *HAL_ADC_ConvCpltCallback* est appelée, et la valeur mesurée est stockée

dans la variable *Alert*. Si cette valeur est inférieure au seuil défini (*seuil*), une LED est allumée pour indiquer une batterie faible.

Figure 6 : Extrait de STM32. Configuration de l'ADC1.

GPIO

Les GPIOs sont utilisés pour contrôler les composants physiques du robot. Ils sont configurés pour différentes fonctions :

LED2_Pin : Utilisé pour indiquer l'état de la batterie (allumée si la batterie est faible). DIR1_Pin et DIR2_Pin : Utilisés pour contrôler la direction des moteurs. B1_Pin : Utilisé comme bouton d'interruption pour démarrer ou arrêter le robot, bouton d'urgence. La broche GPIO est configurée pour générer une interruption sur un front descendant

Pin N...	Signal on...	GPIO ou...	GPIO m...	GPIO Pu...	Maximum...	Fast Mode	User Label	Modified
PA5	n/a	Low	Output P...	No pull-u...	Low	n/a	LED2	☑
PB2	n/a	Low	Output P...	No pull-u...	Low	n/a	DIR2	☑
PC8	n/a	Low	Output P...	No pull-u...	Low	n/a	DIR1	☑
PC13	n/a	n/a	External ...	No pull-u...	n/a	n/a	B1	☑

Figure 7 : Extrait de STM32. Configurations des GPIOs.

UART 3

L'UART 3 est utilisé pour recevoir des commandes de la tablette toutes les 50 ms. Les données reçues sont stockées dans le buffer (*Buffer_RX*). Lorsqu'un message est reçu, la fonction de rappel *HAL_UART_RxCpltCallback* est appelée pour traiter la commande et configurer la direction et le mouvement du robot. Le débit en bauds (*BaudRate*) est configuré à 9600.

Fonctionnement

Les objectifs principaux de notre contrat étaient de vérifier l'état de la batterie à l'aide de l'ADC, de piloter le robot en Bluetooth, de le faire tourner à droite soit de 45°, soit sur lui-même, dépendamment de la durée d'appui sur la tablette. Nous avons donc implémenté cela, en commençant par la mise en place du bouton d'arrêt d'urgence (bouton poussoir B1), puis par la vérification de la batterie en vérifiant son état toutes les 5 secondes.

Le bouton poussoir est utilisé pour démarrer et arrêter le robot, qu'il soit en mouvement, en rotation ou à l'arrêt. Lorsque le bouton est pressé, on vérifie si le délai minimum entre deux appuis est respecté (filtre anti-rebond). Si c'est le cas, l'état du bouton bascule.

On réalise également une surveillance de la batterie toutes les 5 secondes en utilisant l'ADC pour mesurer la tension de la batterie. On compare la valeur mesurée à un seuil prédéterminé qui vaut 3V. Si la valeur mesurée est plus faible que ce seuil, alors une LED (LED2) est allumée pour indiquer que la batterie est faible.

Ensuite, on se concentre sur les commandes de déplacement, c'est-à-dire avance, stop, tourner de 45° à droite ou à gauche, ou tourner longuement. Le robot reçoit des commandes en code ASCII toutes les 50 ms (période d'envoi d'information du logiciel installé sur la tablette). Pour commencer, le contrat stipulait que le robot devait avancer à une vitesse de 20 cm/s. Pour respecter la vitesse d'avance énoncée dans le contrat, c'est-à-dire V_{voulue} = 20 cm/s, on a commencé par observer la vitesse du robot avec un rapport cyclique de 50%, donc R_{test} = *__COMPARE__* == 20000. On a mesuré une vitesse de V_{test} = 16.6 cm/s. Pour obtenir la valeur de *__COMPARE__* nécessaire pour avoir la vitesse demandée V_{voulue}, on a $V_{voulue} = (R_{\text{voulue}} / R_{test}) * V_{test}$.

$$R_{\text{voulue}} = \frac{20}{16.6} * 20000 = 24096.$$

Dû aux imprécisions de mesures, probablement dues à l'erreur humaine, on se rend compte que le robot va un peu trop vite, et on diminue la valeur de *__COMPARE__* à 22470.

Le contrat précisait également qu'une réception de « avance » alors que le robot était déjà en cours d'avance, devait l'arrêter. Nous avons ajouté un filtre anti-rebond pour éviter les doubles (voir plus) réceptions de la même commande. Ensuite, si le robot avance, nous avons ajouté, par soucis de simplicité de la prise en main, l'équivalent du bouton poussoir sur la tablette. A la réception de cette commande (*STOP*), le robot arrête tout mouvement.

Le plus complexe a été de faire la différence entre un appuie court, qui entraine une rotation de 45° et un appui long, qui entraine une rotation en continue du robot sur lui-même jusqu'à relâchement de l'appuie sur l'écran de la tablette. Nous avons donc déterminé une durée d'appui à partir de laquelle on considère la réception d'un appui long. Cette durée vaut 1s25.

Dans le cas d'un appuie court, nous avons déterminé le temps de rotation équivalent à une rotation de cet angle. Pour cela nous avions réalisées plusieurs mesures de temps de rotation du robot pour un tour complet (360°). A partir de la moyenne, nous avons calculé avec un produit en croix le temps nécessaire pour 45°, qui vaut 1s12.

Dans le cas d'un appui long, c'est à dire que, pendant 1s25, on réceptionne uniquement droite (ou gauche pour une rotation vers la gauche). Lorsque le délai est passé, tant que l'on réceptionne droite (réciproquement pour gauche), le robot tourne sur lui-même.

Réalisation/test

Au cours des séances, nous avons petit à petit réussi à remplir les exigences du contrat. Cependant, nous observions des imprécisions sur la vitesse de notre robot et son angle de rotation.

Comme expliqué précédemment, pour obtenir une vitesse de 20 cm/s, nous avons d'abord effectué une moyenne des mesures de temps nécessaires pour que le robot parcourt 2 mètres lorsque __COMPARE__ == 20000.

```
#define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__)
```

Figure 8 : Fonction __HAL_TIM_SET_COMPARE modulant la vitesse d'un moteur.

Pour réaliser les mesures, nous nous sommes munis d'un mètre et d'un chronomètre. Après la réalisation des calculs, nous avons appliqué la valeur trouvée pour *__COMPARE__*. Lors du premier test, nous avons remarqué une légère imprécision moyenne de 200 ms sur 1 m de distance parcouru. Bien que cet écart avec la vitesse voulu est moindre, nous pouvons l'expliquer par le fait que nous avons effectué une moyenne des mesures de temps à l'aide d'un chronomètre ce qui entraîne une imprécision. Cependant, au fil des séances, il devenait visible que le robot n'allait plus à la même vitesse. En effet, l'état de charge de la batterie est un facteur non négligeable des performances du robot. Nous nous sommes donc assurés que le robot soit bien chargé à chaque début de séance pour remplir les exigences de vitesse.

De plus, on a remarqué que le robot n'allait pas droit, il avait tendance à tourner légèrement vers la droite. On a donc diminué le *__COMPARE__* de gauche pour régler l'écart de vitesse des PWM. Ainsi les fonctions *__HAL_TIM_SET_COMPARE* n'avait pas les mêmes paramètres.

En conclusion, lorsque notre robot était bien chargé, les fonctions *__HAL_TIM_SET_COMPARE* modulant la vitesse des moteurs possédait les paramètres suivant :

```
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 20300);
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 22470);
```

Figure 9 : Fonctions __HAL_TIM_SET_COMPARE modulant la vitesse du robot.

Après ces modifications, il était dorénavant essentiel que l'on utilise le même robot à chaque séance, et qu'il soit bien chargé afin qu'il aille à la bonne vitesse et roule de façon rectiligne. En effet, tous les robots n'ont pas le même niveau d'usure des PWM, les ajustements pour que les robots roulent droits sont donc propres à chacun d'entre eux.

Concernant les erreurs sur le degré de l'angle de rotation du robot, elles sont liées à sa vitesse et donc également à l'état de charge de la batterie. Ainsi, une fois les calculs théoriques appliqués pour faire tourner le robot de 45°, il faut s'assurer que le robot soit bien chargé pour que la marge d'erreur de +/- 5° soit respectée. Pour réaliser les mesures, nous avons utilisé un chronomètre pour évaluer le temps que le robot mettait pour faire un tour (360°). Nous faisions tourner le robot sur lui-même et on comptabilisait les tours grâce à une marque sur le sol.

Chronogrammes

Nous avons vérifié expérimentalement la fréquence et le rapport cyclique des PWM grâce à des chronogrammes réalisés à l'oscilloscope.

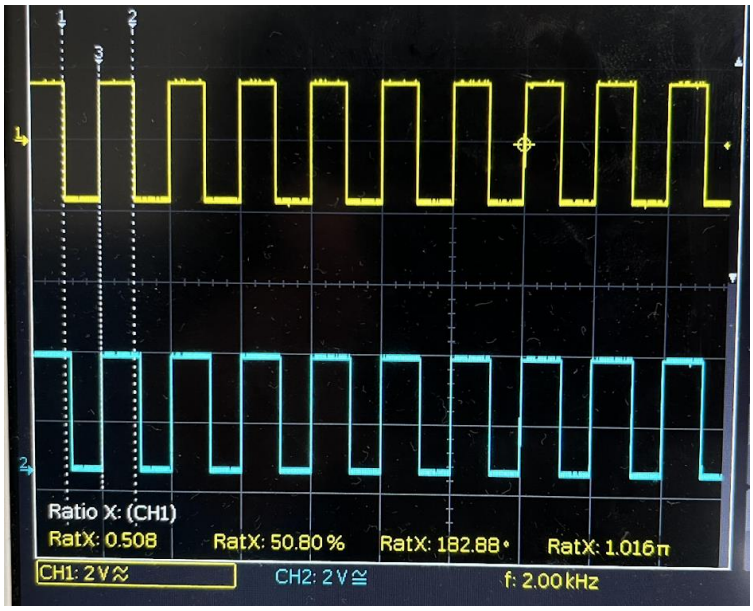


Figure 10 : Chronogramme 1 des PWM.

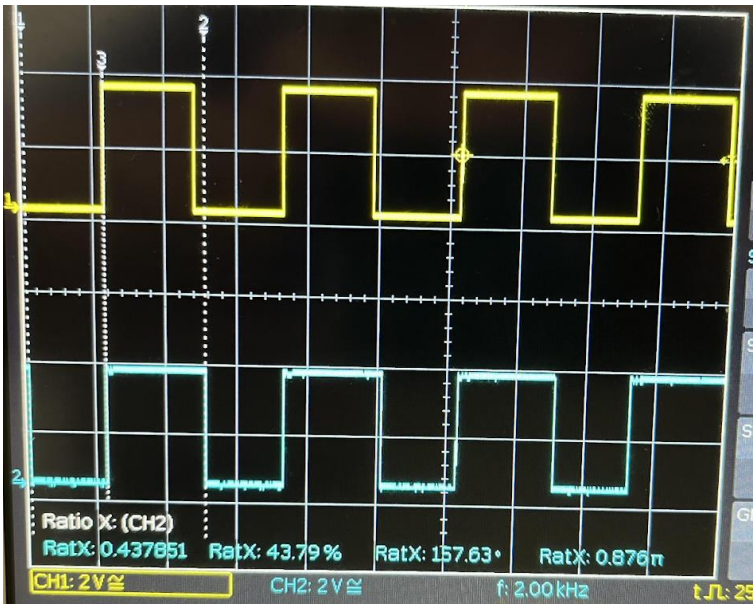


Figure 11 : Chronogramme 2 des PWM.

Sur les chronogrammes, CH1 correspond au PWM de gauche, CH2 correspond au PWM de droite. On retrouve bien que la fréquence du signal est de 2kHz. De plus, sachant que notre AutoReload Register vaut 40000, on retrouve bien nos valeurs de rapport cyclique pour chaque PWM. D'après la figure 10, pour le PWM de gauche, on obtient un ratio de

50,8%, or $\frac{50.8}{100} * 40000 = 20320$. Cela correspond bien à la valeur de __COMPARE__ établie pour le PWM de gauche. On observe également que le PWM de droite à un plus grand rapport cyclique, comme espéré. D'après la figure 11, pour le PWM de droite, on obtient un ratio de 56,21%, or $\frac{56.21}{100} * 40000 = 22470$. Cela correspond bien à la valeur de __COMPARE__ établie pour le PWM de droite.

SWV

Nous avons également effectué les mesures via SWV de tous nos timers pour observer leur fonctionnement.

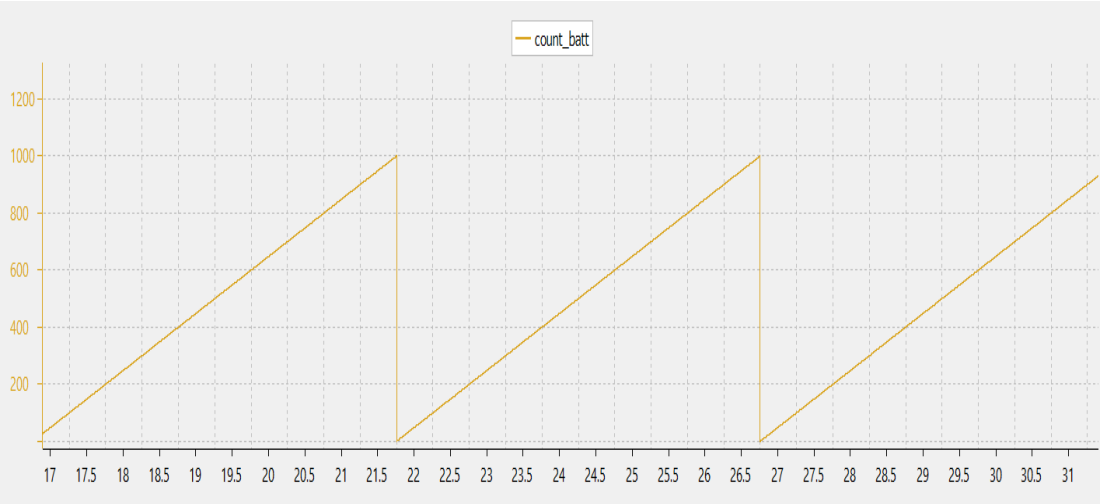


Figure 12 : Courbe SWV du compteur count_batt.

Comme expliqué précédemment, ce compteur qui s'incrémente toutes les 5 ms (grâce au Timer 6) permet d'effectuer une mesure de l'ADC toutes les 5 secondes. Donc quand il atteint la valeur 1000, une mesure est prise et le compteur redémarre. On observe que le fonctionnement du compteur est correct.

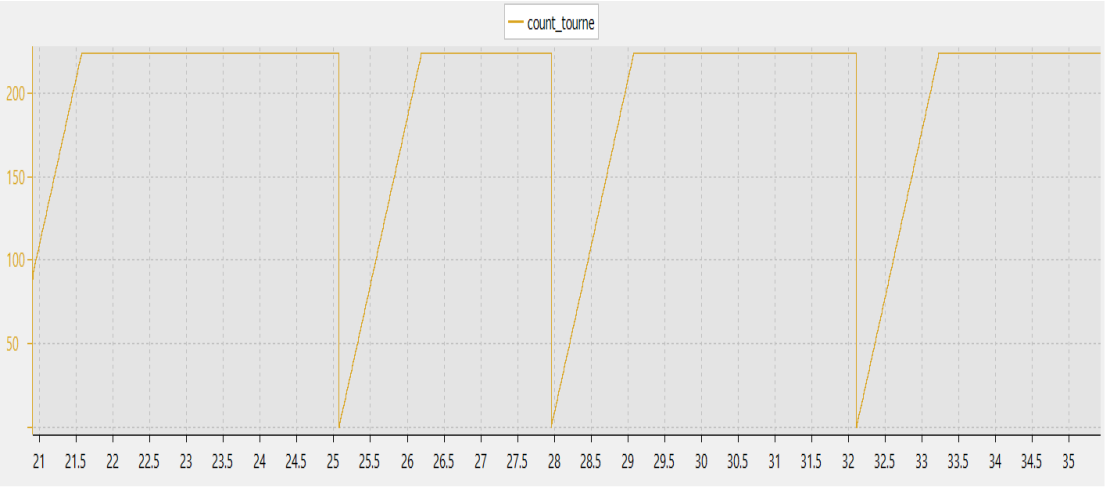


Figure 13 : Courbe 1 SWV du compteur count_tourne.

Comme expliqué, ce compteur permet de définir si on a fait un appui long sur la touche tourner ou pas, mais également à gérer le temps que le robot doit passer à tourner afin de réaliser 45°. Nous avions retenu qu'il mettait 1s12, donc comme le compteur s'incrémente toutes les 5 ms, le compteur doit atteindre 224 pour laisser le temps au robot de faire 45°. Ensuite il reste à cette valeur jusqu'à un nouvel appui (long ou court) où il se remet à zéro. Sur la figure 13, on voit que nous avons fait plusieurs appuis court sur le bouton tourner (gauche ou droite) et que le compteur s'incrémente de 0 jusqu'à 224 pour laisser au robot le temps de faire 45°. Le compteur fonctionne donc bien comme prévu.

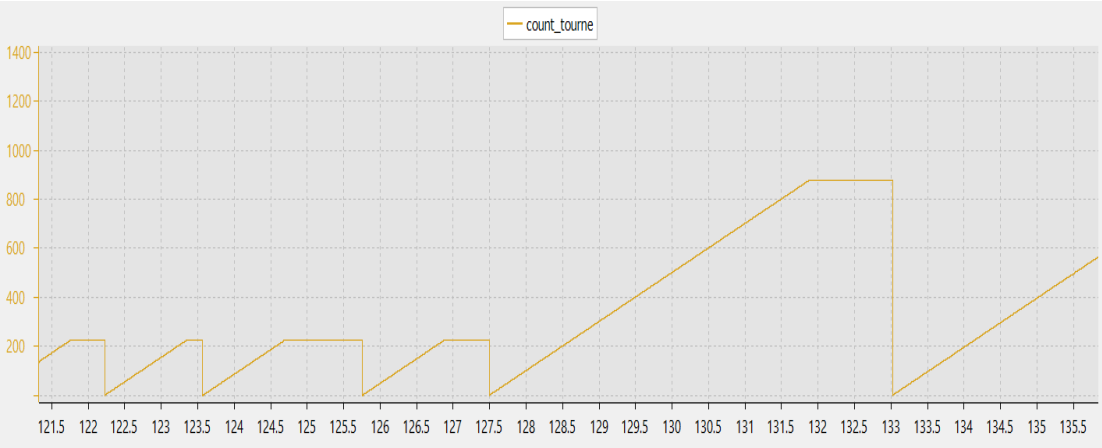


Figure 14 : Courbe 2 SWV du compteur count_tourne.

Sur cette figure, on voit que *count_tourne* sert à faire tourner le robot pendant 1s12 (45°) lors d'un appui court, mais également à le faire tourner jusqu'à relâchement du bouton lorsque l'appui est considéré comme long (compteur > 250 soit 1.25 s). Dans le cas où l'appui sur tourner (gauche ou droite) est long, le compteur s'incrémente jusqu'à relâchement du bouton.

Les deux prochains compteurs sont les compteurs servant à mettre en place des filtres anti-rebond sur le bouton poussoir B1 et sur le bouton avancer de la tablette.

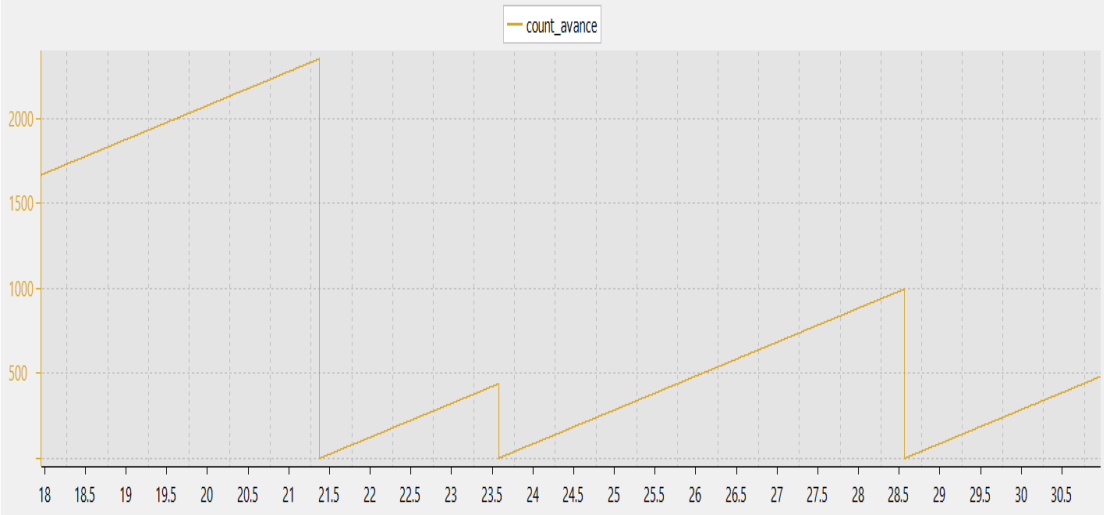


Figure 15 : Courbe SWV du compteur count_avance.

Ce compteur s'incrémente jusqu'à l'infini toutes les 5 ms. À chaque appui, il est réinitialisé à 0. Si on reçoit la commande avancer alors que le compteur est inférieur à 70 (environ ⅓ de seconde), on ne prend pas en compte cette commande. Ainsi, on évite de recevoir plusieurs fois en même temps l'information d'avancer.

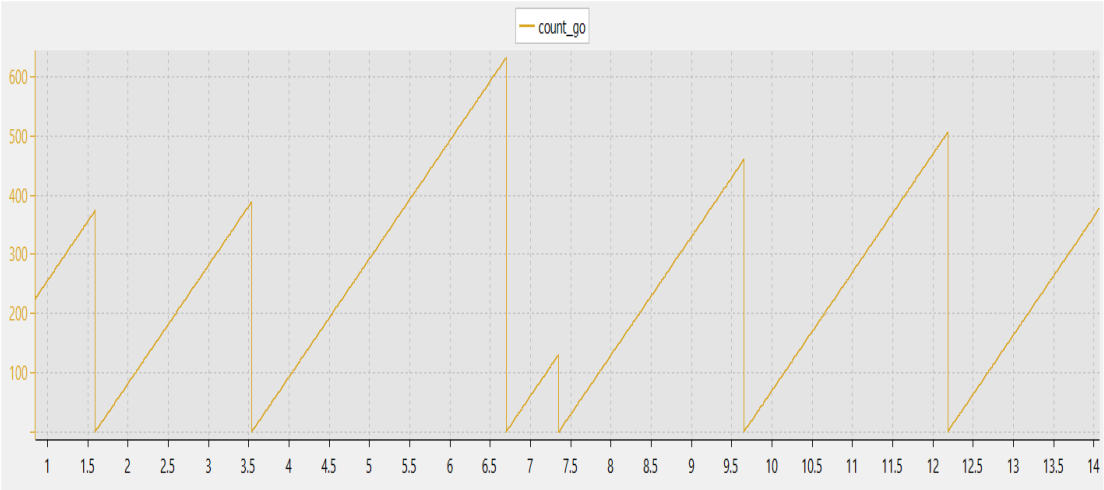


Figure 16 : Courbe SWV du compteur count_go.

Même fonctionnement pour ce compteur, il s'incrémente jusqu'à l'infini tous les 5 ms. Il est réinitialisé à 0 à chaque appui. Si on appui sur le bouton poussoir alors que le compteur est inférieur à 70 (environ ⅓ de seconde), on ne prend pas en compte cette commande.

Vérification de la surveillance batterie

Le robot est alimenté par une tension de 8V, cependant cette tension est régulée à plusieurs reprises pour alimenter la carte NUCLEO a 3.3V. On nous demande de surveiller la tension de batterie pour détecter les chutes de tension à des valeurs inférieures à 3V.

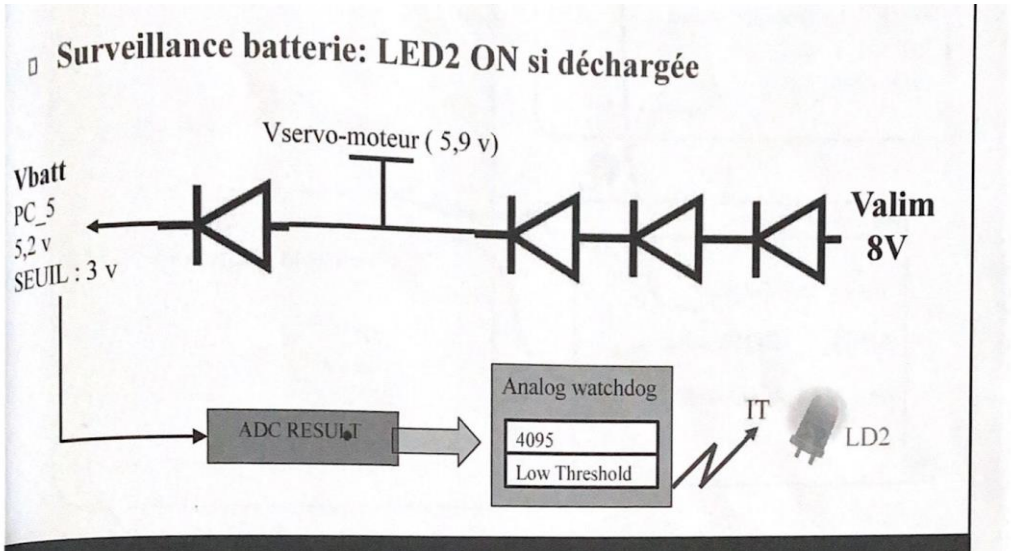


Figure 17 : Schéma du polycopié projet robot sur la surveillance batterie.

Pour surveiller la batterie, on utilise l'ADC connecté au PIN PC_5. Comme expliqué précédemment, nous avons choisi une résolution 8 bits. Ainsi, toutes les valeurs de tension perçues entre 5.2V et 3.3 V seront donc converties à la valeur 255 par l'ADC. Pour vérifier que la surveillance batterie fonctionne bien, nous avons donc pris un robot déchargé pour vérifier que la valeur ne soit pas constamment à 255.

Finalement, malgré qu'il n'était pas précisé dans notre contrat que nous devions effectuer un asservissement, nous avons rempli les spécifications du contrat concernant la précision en vitesse et en degré. Il était demandé que le robot aille à une vitesse de 20 cm/s. Nous avons mesuré que notre robot prenait en moyenne 10 secondes pour avancer de 2m avec une précision de +/- 100 ms. Cela équivaut à une imprécision de +/- 0.2 cm/s soit +/- 1%. Notre précision angulaire correspond aux attentes d'une erreur inférieure à +/- 5°. Pour le vérifier, nous avons appuyé deux fois sur le bouton tourner pour réaliser un angle de 90°. Nous avons pu constater que l'erreur était bien inférieure à 5°.

Conclusion

Les améliorations les plus simples à imaginer et à implémenter sont celles que nous avons déjà mises en place en addition à notre contrat. Elles comprennent les filtres anti-rebond, la rotation courte et longue du robot vers la gauche, ainsi qu'une commande d'arrêt du robot sur la tablette.

Du fait des difficultés que nous avons rencontrées pour respecter la vitesse et l'angle de rotation demandés pour le robot, nous pensons que l'ajout d'un asservissement serait l'amélioration la plus utile. En effet, la vitesse (et par conséquent l'angle de rotation) de notre robot dépendaient de manière non négligeable de l'état de sa batterie. Nous avons dû plusieurs fois modifier manuellement la valeur du rapport cyclique en fonction de la batterie, mais également du robot que nous utilisions. Nous pourrions penser à établir une régulation proportionnelle, proportionnelle intégrale ou proportionnelle intégrale dérivée.

Nous pouvons imaginer les applications industrielles suivantes pour notre robot :

Gestion des stocks : Le robot pourrait naviguer dans les entrepôts, transporter des charges, facilitant ainsi la tâche de l'homme.

Plantation et récolte : Le robot pourrait naviguer dans les champs pour planter ou récolter des cultures, tout en tournant et en se déplaçant pour suivre les rangées de plantations.

Cependant, pour réaliser ces actions, notre robot nécessiterait sûrement l'ajout de nouvelles améliorations telles que des capteurs, comme un sonar, qui lui permettrait de détecter des obstacles. En exagérant, on pourrait imaginer que notre robot serait utilisable pour désamorcer des bombes!

Annexes

```
/* 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 "string.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

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

    #define seuil 232

    //Pour les réceptions de l'UART
    #define AVANCE 70
    #define STOP 66
    #define DROITE 82
    #define GAUCHE 76


/* USER CODE END PD */

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

/* USER CODE END PM */
```

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

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
/* USER CODE BEGIN PV */

    //Bouton de lancement et d'arret
    volatile uint16_t go = 0;
    volatile int count_go = 0;

    //Surveillance batterie
    volatile uint16_t Alert;//Valeur de tension de l'alimentation
    volatile int count_batt = 0;//Compteur entre deux mesures de valeur de tension d'alimentation

    //Pour l'avance
    int avance = 0; //Etat d'avance du robot : vaut 0 si
    //le robot n'avance pas, vaut 1 si il est en cours d'avance
    volatile int count_avance = 0; //Compteur entre deux récéptions de "avancer".

    //Reception Bluetooth
    unsigned char Buffer_RX[2]; //Uart 3
    volatile char dir; //Commande reçue par le robot, envoyée par la tablette.

    //Pour tourner
    volatile int va_tourner = 0; //Si le robot va tourner : 0 si non, 1 si oui
    volatile int count_tourne = 0;//Compteur de temps après la première réception de droite ou gauche
    volatile int tourne_court = 0;//Le robot doit tourner de 45 degrés. Appui rapide sur la tablette.

    //Pour le sens de rotation
    volatile int droitegauche = 0; // 1 si droite, 0 si gauche


/* 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_TIM6_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART3_UART_Init(void);
/* USER CODE BEGIN PFP */
```

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

/* 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_TIM6_Init();
  MX_TIM2_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

  HAL_TIM_Base_Start_IT(&htim6);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //gauche
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);// droite
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
  HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_SET);
  HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_SET);
  HAL_UART_Receive_IT(&huart3, (unsigned char *)Buffer_RX, sizeof(Buffer_RX));
```

```
/* USER CODE END 2 */

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

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    while(go){
        if(count_batt >= 1000) //Vérification batterie du robot
        {
            HAL_ADC_Start_IT(&hadc1);
            if(Alert < seuil) //Robot déchargé
            {
                HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, SET); //Allume LED2
            }
            else //Robot chargé
            {
                HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, RESET); //Eteind LED2
            }
            count_batt = 0; //Remise a zero du compteur
        }
        if(dir == AVANCE){ //Reception du bouton avancement
            if(avance == 0)//N'avance pas
            {
                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 20300);
                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 22470);
                avance = 1;
            }
            else if(avance == 1)//En cours d'avance
            {
                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
                __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
                avance = 0;
                }
            dir = 0;
            count_avance = 0; //Remise a zero du compteur
        }
        if(dir == STOP) //Bouton d'arret des moteurs, équivalent à Stop
        {
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
            avance = 0;
            dir = 0;
            HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_SET);
            HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_SET);
```

```
if((dir == DROITE) || (dir == GAUCHE)) //Réception de droite ou gauche
{
    if(avance == 1 && !va_tourner) //En cours d'avance et c'est  prévu qu'il tourne
    {
        count_tourne = 0;
        va_tourner = 1;
        if(dir == DROITE){ //droite
            droitegauche = 1;
        }
        else if(dir == GAUCHE){ //gauche
            droitegauche = 0;
        }
    }
    dir = 0;
}
if ((count_tourne > 1) && (count_tourne <= 250) && (Buffer_RX[0] != DROITE) && (va_tourner == 1)){
     //Appui court sur droite
    if(droitegauche == 1){
        HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_RESET);
        tourne_court = 1; //doit tourner de 45 degrés
        dir = 0;
        }
}
if ((count_tourne > 1) && (count_tourne <= 250) && (Buffer_RX[0] != GAUCHE) && (va_tourner == 1)){
//appui court sur gauche
    if(droitegauche == 0){
        HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_RESET);
        tourne_court = 1; //doit tourner de 45 degrés
        dir = 0;
            }
    }
if(count_tourne > 250 && va_tourner == 1){ //En train de réceptionner un appui long
    if(droitegauche == 1){ //appui long sur droite
        while(Buffer_RX[0] == DROITE) // on peut pas faire while buffer et droitegauche = 1
        {
            HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_RESET);
        }
        va_tourner = 0;
        HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_SET);
        dir = 0;
        }
    else if(droitegauche == 0){
          while(Buffer_RX[0] == GAUCHE) //appui long sur gauche
          {
              HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_RESET);
          }
          va_tourner = 0;
          HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_SET);
          dir = 0;
```

```
            }
          }
        }
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, 0);
        avance = 0;
        dir = 0;
    }
    /* 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_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_SYNC_PCLK_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_8B;
  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 Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_14;
  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 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 = 40000;
  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
  */
```

```c
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 = 7-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 57143-1;
  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 USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 9600;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  * @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(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_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 : LED2_Pin */
  GPIO_InitStruct.Pin = LED2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : DIR2_Pin */
  GPIO_InitStruct.Pin = DIR2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(DIR2_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);
```

```
  /*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 HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    UNUSED(htim);
    count_batt++;
    count_avance++;
    count_go++;
    if(va_tourner){
        count_tourne++;
        }
    if((tourne_court == 1) && (count_tourne >= 224)){
        if(droitegauche == 1){
            HAL_GPIO_WritePin(GPIOC, DIR1_Pin, GPIO_PIN_SET);
        }
        else if(droitegauche == 0){
            HAL_GPIO_WritePin(GPIOB, DIR2_Pin, GPIO_PIN_SET);
        }
        va_tourner = 0;
        tourne_court = 0;

        }
    }

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    Alert = HAL_ADC_GetValue(hadc);
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
        if((Buffer_RX[0] != 83) && (go == 1)){
            if (!( (count_avance < 70) && (Buffer_RX[0] == AVANCE)))
                dir = Buffer_RX[0];
        }
        HAL_UART_Receive_IT(&huart3, (unsigned char *)Buffer_RX, sizeof(Buffer_RX));

}
```

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if (GPIO_Pin == B1_Pin){
        if (count_go > 70){
            go =!go;
            count_go = 0;
        }
    }
}
/* 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 */
```