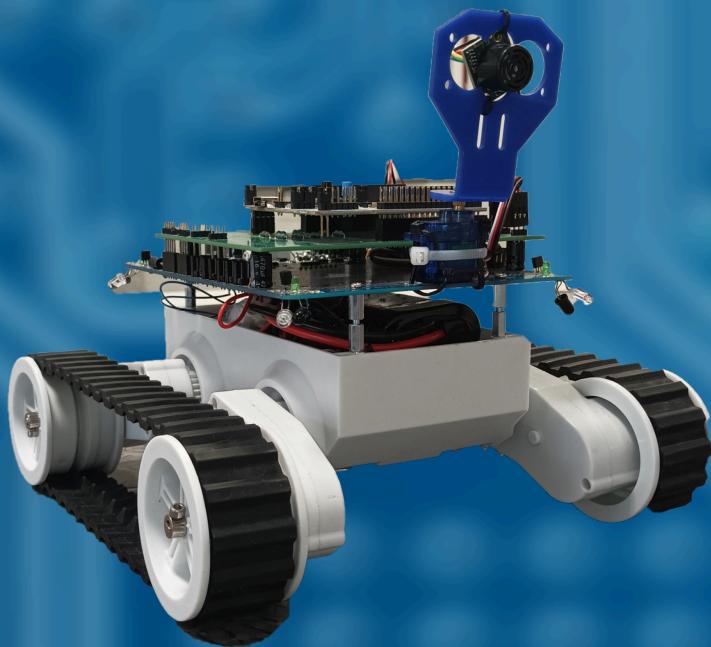


# COMPTE RENDU DE PROJET ROBOT

## Contrat n°6



Maisonnette Mathieu  
Larroze Chloé

## Table des matières

<b>1. Introduction</b>	<b>2</b>
<b>2. Rappel du contrat n°6 et cahier des charges</b>	<b>3</b>
2.1. Cahier des charges :	3
2.2. Configuration des pins :	3
<b>3. Algorigramme et initialisation des pins</b>	<b>4</b>
3.1. Fonctionnement global de notre programme:	8
3.2. Initialisation du signal PWM (TIM2) :	8
3.3. Calcul vitesse robot :	10
3.4. Calcul des seuils batterie :	10
3.5. Initialisation du module Bluetooth (uart) :	10
<b>3.6. Initialisation du Timer 6 :</b>	<b>11</b>
3.7. Leds infrarouges :	11
<b>3.8. Mode scan :</b>	<b>14</b>
3.9. Sens de dégagement du robot :	14
3.10. Améliorations	15
3.10.1. Accélération progressive - Initialisation du TIM7 et PWM	15
3.10.2. Anti-rebond	16
3.10.3. Gestion des coins	16
<b>4. Réalisation et tests</b>	<b>16</b>
4.1. Module bluetooth : l'uart en fonctionnement	16
4.2. Test des PWM	16
4.3. Sous-programme d'évitement d'obstacles	17
4.3.1. Tests des leds infrarouges	18
4.3.2. Test du timer 6	18
4.4. Tests des améliorations	19
4.4.1. Timer 7	19
4.4.2. PWM pour une accélération	19
4.4.3. Anti-rebond	20
4.4.4. Coins	21
<b>5. Conclusion</b>	<b>21</b>
5.1. Évolutions possibles du contrat	21
5.2. Applications sur le marché	21
<b>6. Annexes</b>	<b>21</b>
<b>7. Tables des figures</b>	<b>21</b>
<b>8. Sources</b>	<b>22</b>



## 1. Introduction

L'objectif de ce projet est de concevoir et de réaliser un robot autonome répondant aux contraintes spécifiques définies dans le contrat n°6. Ce projet s'inscrit dans le cadre de notre formation en systèmes embarqués et vise à approfondir nos compétences en programmation et en intégration de composants matériels avec CubeIDE. Le robot doit être capable d'établir une connexion Bluetooth avec un téléphone pour recevoir des commandes, éviter les obstacles de manière autonome, gérer son niveau de batterie et reprendre son parcours après une éventuelle interruption. À travers ce rapport, nous détaillerons le processus de conception, les choix techniques, les algorithmes utilisés, ainsi que les tests effectués pour valider le bon fonctionnement du robot.

## 2. Rappel du contrat n°6 et cahier des charges

### 2.1. Cahier des charges :

Le contrat nous ayant été attribué est le contrat numéro 6. Ce dernier inclut les contraintes suivantes :

- **Pilote** : doit pouvoir établir une connexion Bluetooth avec téléphone pour avancer ou tourner à droite ou à gauche ;
- **Évitement** : le robot doit éviter automatiquement la collision avec le mur lorsque ce dernier est à moins de 20 cm en choisissant le côté où le mur n'est pas présent ;
- **Reprise** : lorsqu'il n'y a plus de collision, le pilote reprend la main ;
- **Batterie** : contrôle de la batterie avec l'AnalogWatchdog et allumage de la led si la batterie est faible ;
- **Spécifications chiffrées** : Les spécifications du cahier des charges se traduisent entre autre par les contraintes chiffrées suivantes :
  - o PWM : 2 kHz ;
  - o Vitesse : 10 cm/s ;
  - o Précision pour la détection : plus ou moins 10 cm ;
  - o Fonctionnement en boucle ouverte

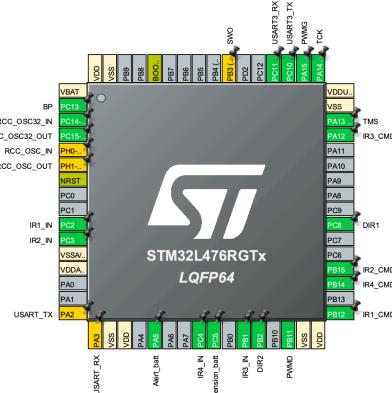
Ce cahier des charges laisse place à l'interprétation sur la manière dont l'évitement est géré. Il est en effet imposé d'aller du côté où il n'y a pas de mur mais aucune indication n'est donnée sur la rotation nécessaire.

Nous avons fait le choix de demander au robot de tourner jusqu'à ce que le mur ne soit plus détecté. Ainsi lors d'une avancée en direction d'un mur, le robot va le détecter et tourner sur lui-même jusqu'à ne plus détecter de mur. Cela revient à se mettre parallèle au mur dans le cas où celui-ci est plat. Le robot continue ensuite son chemin avec la trajectoire précédemment demandée par l'utilisateur.

### 2.2. Configuration des pins :

Sous cubeIDE, nous reprenons la configuration proposée dans le carnet de projet "Projet Robot" et optons donc pour le pinout suivant :



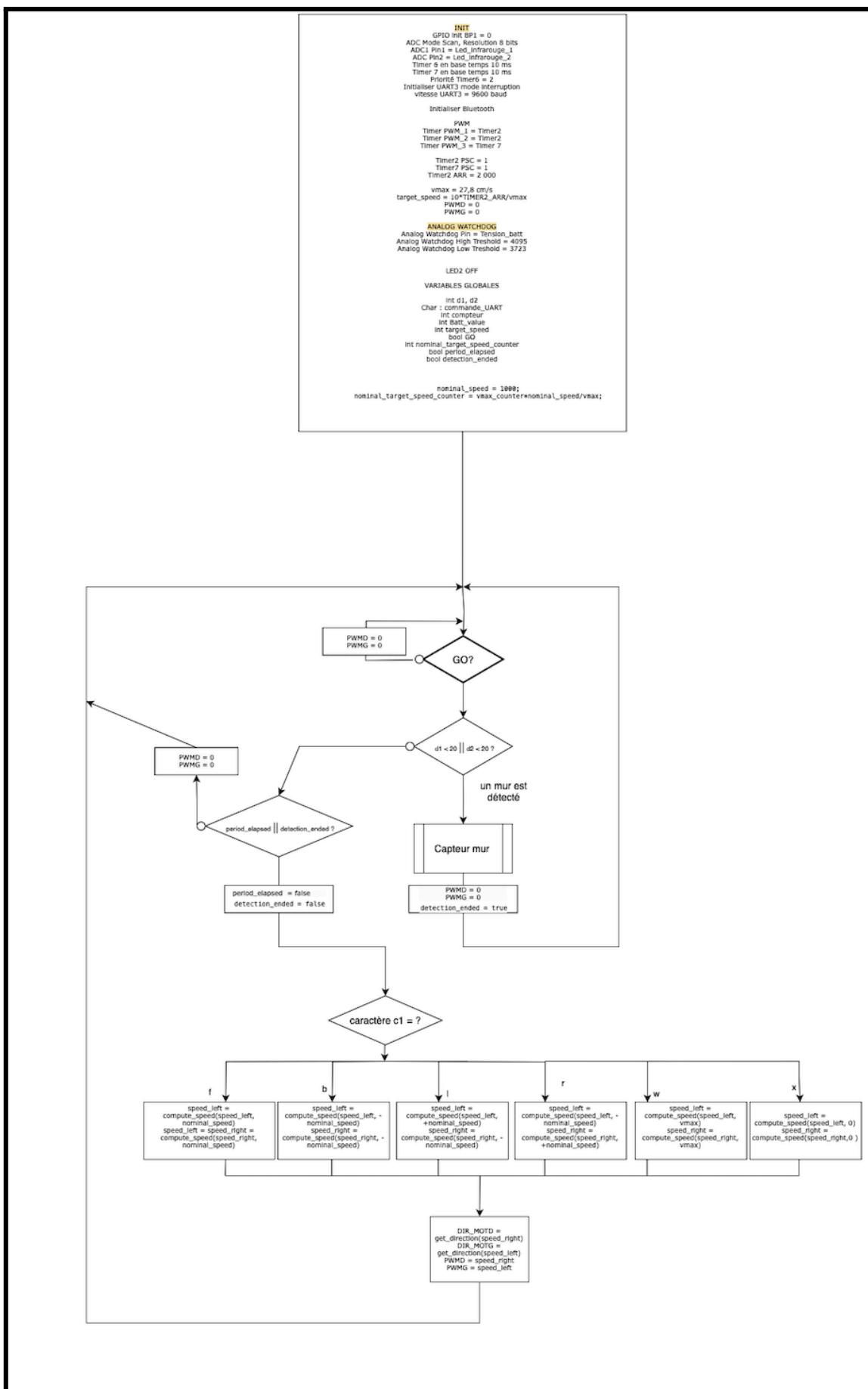


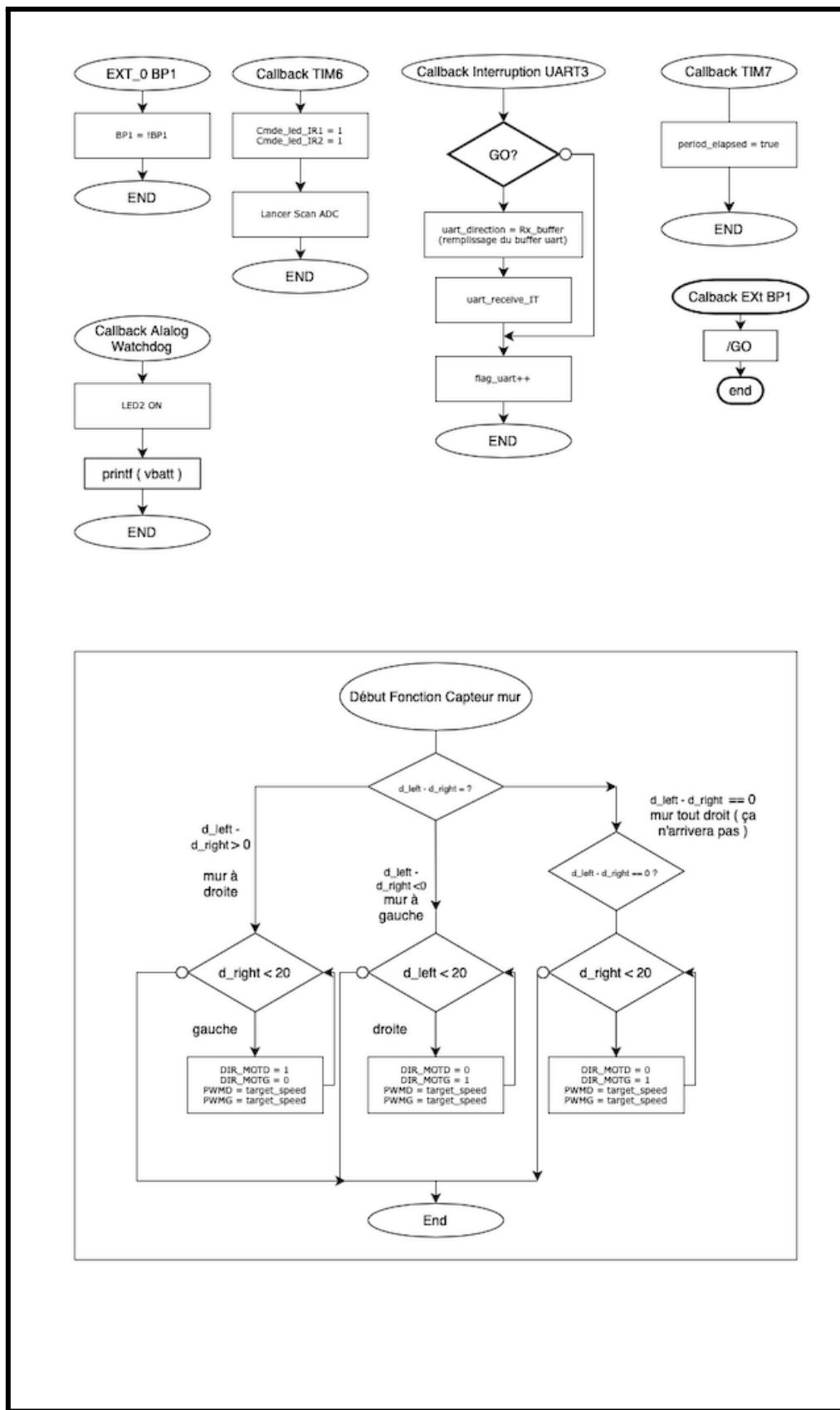
*Figure 1 : Configuration pins STM32*

### 3. Algorigramme et initialisation des pins

Avant même de commencer à coder, il est important de réaliser un algorigramme complet afin de ne pas s'égarer lors de la phase de codage ou encore de ne pas confondre des pins, consignes ou périphériques. Une fois notre algorigramme validé, nous pourrons nous lancer dans la phase de réalisation.

L'organigramme proposé est disponible dans les pages suivantes avec la fonction principale ci-dessous suivi des différentes fonctions et des différentes interruptions.





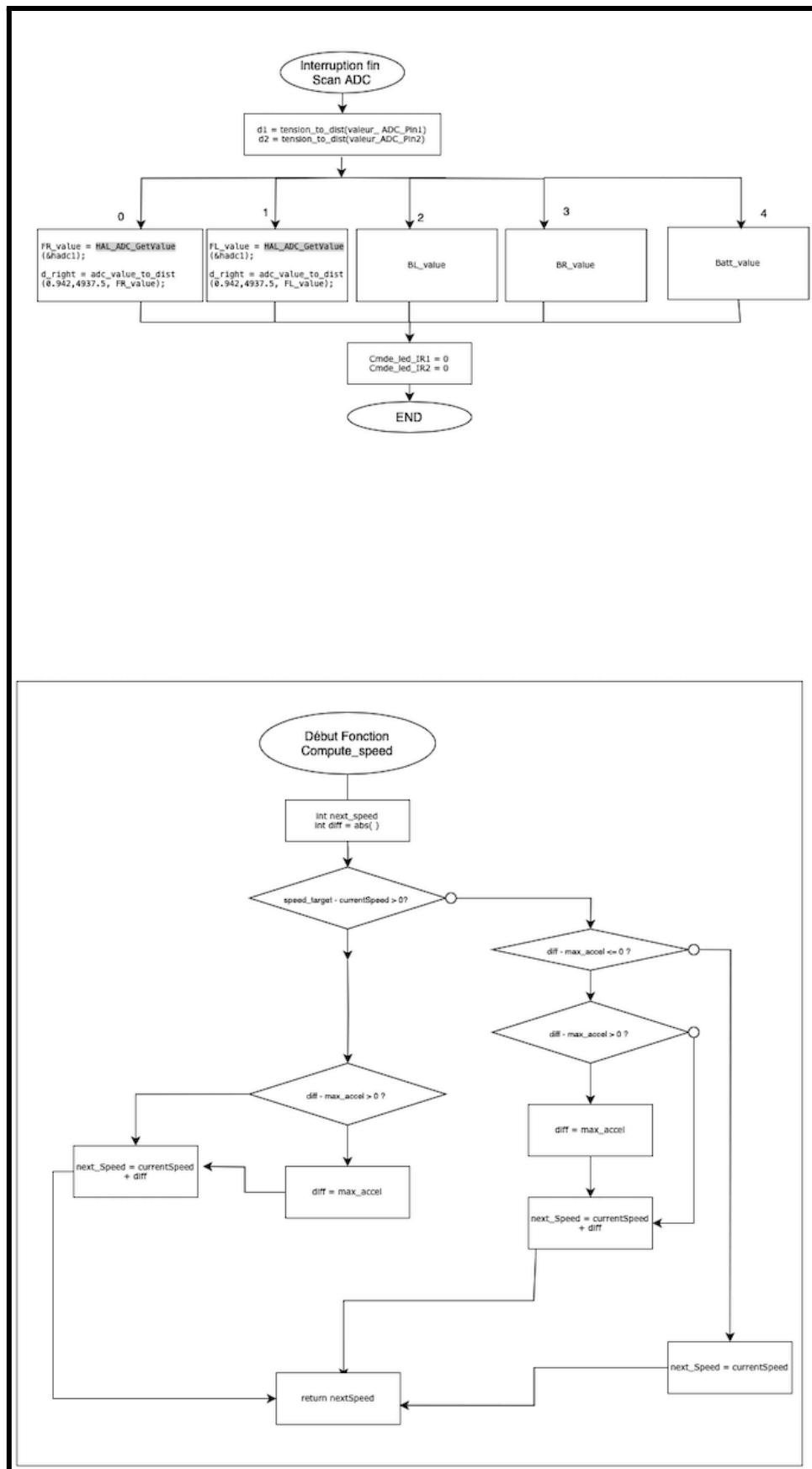


Figure 2 à 4: Algorigramme complet



### 3.1. Fonctionnement global de notre programme:

Détaillons à présent l'algorigramme ci-dessus pour en expliciter le fonctionnement. Chaque périphérique sera, quant à lui, plus amplement détaillé dans la partie suivante.

Une fois le bouton marche/arrêt enclenché, le robot démarre et entre dans la boucle principale du programme. Cette détection de l'appui du bouton est faite en interruption.

Tant que le bouton bleu "go" situé sur le dessus du robot n'est pas appuyé, le robot n'avance pas et n'interprète aucune information provenant du téléphone de l'utilisateur. Si l'utilisateur effectue envoie quelque chose au robot, ceci sera reçu dans l'interruption de l'UART3 mais ne sera pas utilisé. En même temps, les interruptions de scan ADC des leds infrarouges basés sur le timer TIM6 démarrent et relèvent les valeurs d\_left et d\_right toutes les 10 ms.

Dès que le bouton "go" est enfoncé, la carte STM prévoit un buffer contenant deux caractères qu'il remplit au fur et à mesure que l'utilisateur choisit une direction sur son application, à l'aide d'une interruption sur l'uart 3 que l'on aura précédemment initialisé en tant que interruption. On ne récupère cependant uniquement le premier caractère, fixant la direction du robot qui permettra le pilotage. En effet l'application envoie, à chaque appui, 2 caractères mais le second caractérise la vitesse choisie dans l'application que nous n'utilisons pas.

Si le robot s'approche trop d'un mur, c'est-à-dire si l'une de ces distances relevées par les leds infrarouges est inférieure à 20 cm, soit la distance qu'il faut toujours conserver entre le robot et le mur, on rentre dans un sous-programme intitulé « Capteur mur ».

Celui-ci implémente une disjonction de cas afin de choisir une direction pour le dégagement en fonction de l'angle d'approche. Puis, une fois le dégagement effectué et le robot sorti de tout risque de collision, ce dernier reprend sa route en direction de la dernière commande entrée par l'utilisateur. En l'absence d'obstacles, les vitesses des moteurs sont ajustées selon les commandes reçues via UART.

Une fois cet algorigramme établi et validé, nous pouvons démarrer l'initialisation des pins. Cette dernière requiert quelques calculs préliminaires :

Nous nous baserons dans tous les dimensionnement de timers faites par la suite sur les formules introduites durant le cours de SAM1 :

$$\begin{aligned} \text{Periode} &= \text{ARR} * \text{PSC} * \text{Timer}_{clk} \\ \text{PSC}_{opti} &\geq \left( \frac{\text{Periode}}{2^{16} * \text{Timer}_{clk}} \right) \geq 1220,7 \\ \text{Soit PSC} &= 1221 \\ \text{ARR} &= \left( \frac{\text{Periode}}{\text{PSC} * \text{Timer}_{clk}} \right) = 65520 \end{aligned}$$

Figure 5 : Formules initialisation, cours SAM1, A.Marques

### 3.2. Initialisation du signal PWM (TIM2) :



Le robot avance à l'aide de deux chenilles de part et d'autre du robot, actionnées par deux petits moteurs reliés à la carte STM32. Ces moteurs fonctionnent grâce à un driver utilisant un pont de hacheur, également appelé pont en H, est un circuit électrique utilisé pour contrôler la direction et la vitesse des MCC. Il se compose de quatre interrupteurs (généralement des transistors MOSFET ou IGBT) disposés en forme de H.

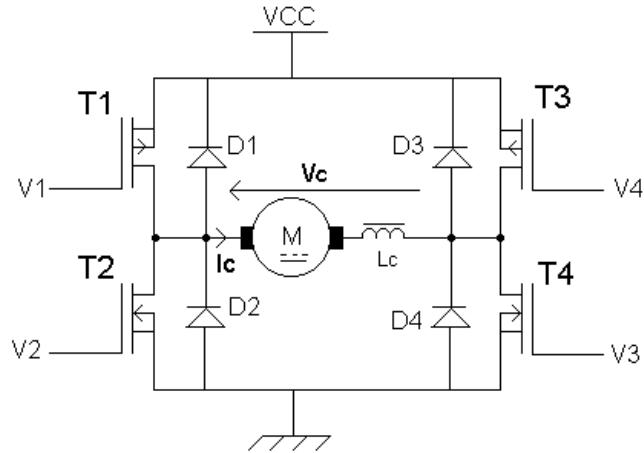


Figure 6 : Pont de hacheur

Pour faire tourner le moteur dans une direction, par exemple, vers l'avant, on active les interrupteurs diagonaux opposés et inversement pour l'autre sens de rotation.

La vitesse du moteur est quant à elle contrôlée par PWM. Il est exprimé en pourcentage et représente la proportion de temps où le signal est à l'état haut par rapport à la période totale du signal. Ce type de signaux permet de donner une tension moyenne variable en sortie en utilisant uniquement un signal logique. Par exemple, un cycle de travail de 50 % signifie que le signal est haut pendant la moitié du temps et bas pendant l'autre moitié. La tension moyenne sera alors la moitié de la tension maximale donc le moteur tournera à 50% de sa vitesse maximale.

En se référant au pinout indiqué dans le polycopié “Projet Robot”, nous initialisons le signal PWM, permettant un contrôle précis de la vitesse du moteur, sur le timer TIM2: Comme indiqué dans le cahier des charges, on veut  $f_{PWM} = 2 \text{ kHz}$  donc la période est :

$$T_{PWM} = \frac{1}{f_{PWM}} = \frac{1}{2.10^3} = 5e-4;$$

Calculons donc le réglage du Prescaler PSC, qui est un composant HardWare utilisé pour diviser la fréquence du timer à une valeur gérable par l'auto-reload register (ARR). En effet l'ARR codé sur :

On sait que  $f_{Timer} = 4 \text{ MHz}$ ,

Donc

$$PSC_{opti} > \frac{5.10^{-4} * 4.10^6}{2^{32}} = 1.63e-9$$

L'utilisation d'un prescaler n'est donc pas nécessaire pour une fréquence de 2kHz et l'aurait été seulement une période plus grande. Nous prendrons donc  $PSC = 1$  ;



Enfin,

$$\text{ARR} = 5 \cdot 10^{-4} \cdot 4 \cdot 10^6 = 2\,000$$

Ceci signifie que le timer comptera jusqu'à 2 000 pour générer une période de PWM de 0.5 ms, correspondant à une fréquence de 2 kHz.

Une fois notre signal PWM initialisé, poursuivons les calculs et calculons les paramètres nécessaires à la détermination de la vitesse de déplacement du robot.

### 3.3. Calcul vitesse robot :

Dans le contrat, il est fourni une vitesse maximale à laquelle doit avancer le robot :

$$v_{max\ robot} = 1\text{km/h} = 0,278\text{ m/s} = 27,8\text{ cm/s}$$

donc, cela correspond à CCR1 = ARR. Ainsi,

$$vitesse_{voulue} = 10\text{ cm/s.}$$

$$\text{On met donc } CCR1 = \frac{\text{ARR} * 10}{v_{max}} \text{ où } v_{max} = 27,8 \text{ donc } CCR1 = \frac{40.000 * 10}{27,8} = 14\,388$$

Dans la partie 3.10 intitulée "améliorations", nous avons également implémenté une accélération en début de déplacement et donc une vitesse progressive nous ayant contraint de changer notre algorithme initial.

De surcroît, on utilise des micromètres afin de ne pouvoir travailler qu'avec des entiers et pas des flottants.

### 3.4. Calcul des seuils batterie :

Pour le contrôle du niveau de batterie, nous avons choisi de remplacer l'interruption software Callback\_ADC par un callback\_Watchdog, c'est-à-dire une surveillance hardware dont on pourra fixer les seuils. Cette fonctionnalité se nomme Analog Watchdog. Nous l'activons dans les paramètres de l'ADC et nous définissons les seuils de déclenchement.

L'ADC est sur 12 bits, les tensions acquises (qui sont entre 0 et 3,3V) sont donc converties en des valeurs numériques entre 0 et 4095. On peut déterminer le seuil haut :

$$4095 \rightarrow 3,3\text{ V}$$

donc pour une tension de 3V correspondant au seuil bas, on aura

$$3723 \rightarrow 3\text{ V}$$

que l'on obtient par un simple produit en croix.

Pour tester l'analog watchdog, nous avons défini dans un premier temps des seuils plus faibles (low threshold à 100 et high threshold à 500), étant donné que



nous ne pouvions pas modifier la tension de la batterie. Ainsi, lorsque la tension sort de cette plage (out of window), cela déclenche l'interruption *HAL\_ADC\_LevelOutOfWindow Callback*. Dans cette interruption on allume la LED branchée sur la sortie Alert\_Batt. Nous utilisons ensuite des breakpoints pour confirmer les tests et vérifier que la led s'allume correctement.

### 3.5. Initialisation du module Bluetooth (uart) :

Comme attendu dans le cahier des charges, le robot doit être capable de suivre une commande utilisateur envoyée depuis un téléphone portable muni de l'application Bluetooth RC Controller.

Le module Bluetooth émet et reçoit des signaux radio dans la bande de fréquence de 2,4 GHz et est souvent utilisé en mode esclave, où il attend qu'un autre dispositif (le maître) initie la connexion. D'autre part, l'UART est un protocole de communication série qui permet l'échange de données entre deux dispositifs en utilisant deux lignes : une ligne de transmission (TX) et une ligne de réception (RX). De plus, la communication est asynchrone, ce qui signifie qu'il n'y a pas de signal d'horloge partagé entre les dispositifs. Au lieu de cela, chaque appareil doit s'accorder sur une vitesse de transmission (baud rate).

Pour ce faire, il faut respectivement assigner les broches pour TX et RX du module Bluetooth aux broches RX et TX du microcontrôleur.

On définit ensuite en variable globale un buffer de 2 caractères qui se remplit grâce à une interruption suite à l'appui d'un bouton sur le téléphone de l'utilisateur. On reçoit donc une direction et une vitesse, nous ne prendrons cependant en compte que le premier caractère correspondant à la direction choisie par le pilote.

L'UART va recevoir 2 interruptions successivement (une pour chacun des caractères). Nous stockerons donc le contenu du buffer (de taille 1 caractère) uniquement 1 fois sur 2.

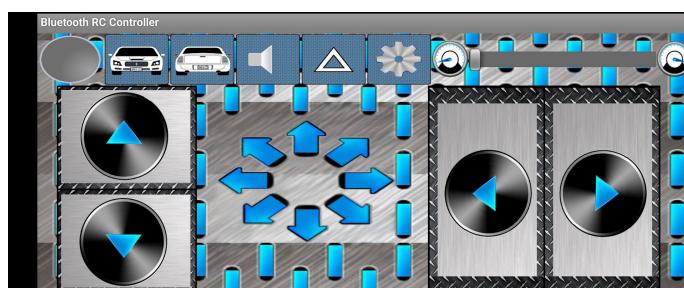


Figure 7 : Application de commande du robot

La commande du robot se fait en un appui sur les flèches, il n'est pas nécessaire de rester appuyé. Les 2 boutons de droite font rentrer le robot en rotation. Les 2 boutons de gauche permettent quant à eux de faire avancer et reculer le robot.

Le caractère est F pour la flèche avant, B pour la flèche arrière, R pour la flèche droite et L pour la flèche gauche. Nous avons également implémenté l'utilisation du bouton phares avant (voiture de face qui génère le caractère W) pour aller à vitesse maximale et le bouton arrêt (triangle qui génère le caractère X) pour arrêter le robot.



### **3.6. Initialisation du Timer 6 :**

L'objectif est de régler le timer 6 toutes les 200 ms afin de le synchroniser avec l'ADC qui relève à la fois la tension de la batterie et les quatre leds infrarouges:

- Periode :  $T = 200 \cdot 10^{-3} \text{ s}$  ;
- $T_{\text{horloge}} = 4 \cdot 10^6 \text{ MHz}$  ;

donc

$$PSC = \frac{200 \cdot 10^{-3} \cdot 4 \cdot 10^6}{2^{16}} = 12.2 \cong 13 \text{ et } ARR = 61538.$$

Enfin, nous gérons l'ensemble de ces variables grâce à un scan qui balaiera toutes les possibilités.

### **3.7. Leds infrarouges :**

Pour détecter des obstacles devant le robot, nous servirons des leds infrarouges dont le principe de fonctionnement est le suivant: une LED infrarouge émet de la lumière dans le spectre infrarouge, typiquement invisible à l'œil humain (La longueur d'onde de cette lumière est généralement comprise entre 850 nm et 950 nm). La lumière infrarouge émise par la LED peut ensuite être réfléchie par un objet ou être interrompue lorsqu'un objet passe devant le faisceau lumineux. De plus, un photodétecteur ou un phototransistor est positionné pour recevoir la lumière infrarouge réfléchie par l'objet. Ce photodétecteur convertit la lumière reçue en un signal électrique proportionnel à l'intensité de la lumière infrarouge détectée et sera interprétable par la STM32.

En programmant les LED infrarouges, nous avons constaté que leurs caractéristiques n'étaient ni symétriques, ni linéaires et dépendaient grandement de la position de celles-ci, ce qui a entraîné de nombreuses erreurs en raison du choix d'un modèle inapproprié. Par conséquent, nous avons décidé de déterminer une caractéristique approchée à l'aide d'une régression exponentielle ou polynomiale. Pour ce faire, nous avons relevé différentes valeurs données par l'ADC en fonction de la distance d'un obstacle (cahier pour simuler un mur).

Pour la led infrarouge avant gauche du robot, on a :

I <sub>max</sub>	205	250	295	291	330	350	395	458	575	695	878	2040	3795
I <sub>min</sub>	167	235	275	281	315	335	385	445	565	678	860	2010	3785
I <sub>min</sub>	186	242,5	285	286	322,5	342,5	390	451,5	570	686,5	869	2025	3790
distan- ce	1000	30	25	22	20	18	15	12	10	8	6	4	0



On répète l'opération pour la led avant droite:

I <sub>max</sub>	295	345	360	395	418	465	530	690	805	119 0	160 0	243 0	381 0
I <sub>min</sub>	280	335	348	380	405	450	521	670	790	116 0	159 0	242 0	379 5
I <sub>moy</sub>	3802 .5	2425 .0	1595 .0	1175 .0	797. 5	680 .0	525. 5	457. 5	411 .5	387. 5	354. 0	340 .0	287. 5

Puis, on en trace la caractéristique et on recherche les équations modélisant le mieux la situation.

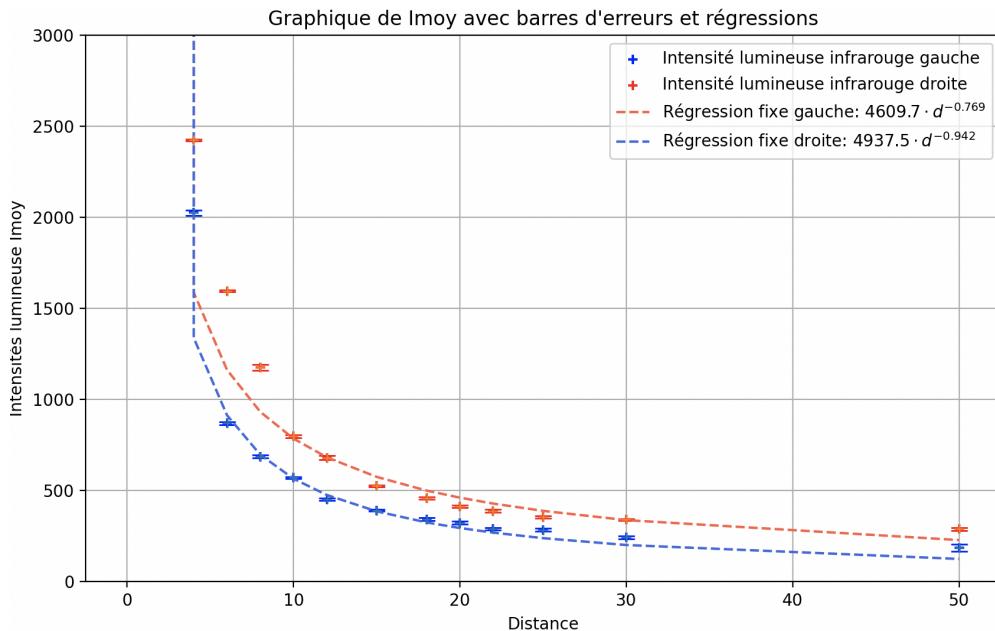


Figure 8: Régression puissance pour les caractéristiques des infrarouges

On obtient donc deux types de régressions : une équation pour l'infrarouge avant droit et une pour l'infrarouge avant gauche que l'on renseignera directement dans le code.

Nous avons jugé bon de ne pas prendre en compte les premières valeurs car celles-ci ne nous intéressent pas dans l'étude de notre modèle et ne participent, au contraire, qu'à fausser celui-ci. Avec un  $r^2$  de 0.993 ( contre 0.987 pour la régression exponentielle ), nous choisissons la fonction régression "puissance" pour l'infrarouge droit:

$$f(d)_d = 4\,937,50 \cdot d^{-0.942}$$

De même, nous avons choisi, pour des questions de simplicité du code, la régression "puissance" pour la fonction de la led gauche même si elle présente un  $r^2$  moins précis que la régression exponentielle ( 0.925 pour la régression puissance contre 0.975 pour l'exponentielle ) . Ainsi :

$$f(d)_g = 4\,609,70 \cdot d^{-0.769}$$



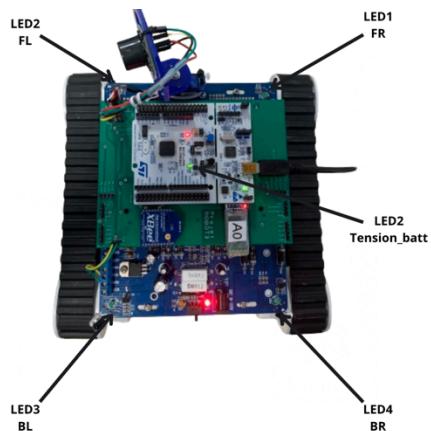
On en obtient donc les réciproques, où  $f(d)$  est la valeur mesurée par les leds, que l'on peut désormais intégrer au code et que l'on retrouve dans la fonction `adc_value_to_dist` :

$$\text{distance infrarouge droit: } d_d = \left( \frac{f(d)}{4936,5} \right)^{\frac{1}{0,942}} \text{ en centimètres}$$

$$\text{distance infrarouge gauche: } d_g = \left( \frac{f(d)}{4609,7} \right)^{\frac{1}{0,769}} \text{ en centimètres}$$

Il faut cependant noter que ce modèle est uniquement valide sur un robot bien précis. En effet, si la position des leds est modifiée, ne serait-ce que de quelques millimètres, il faut refaire les mesures de la valeur en fonction de la distance puis une nouvelle régression pour obtenir les bonnes équations.

### 3.8. Mode scan :



- IR1\_IN sur CH3 ADC1
- IR2\_IN sur CH4
- IR3\_IN sur CH16
- IR4\_IN sur CH13
- V\_batt : Tension\_Batt sur CH14

Figure 9 : Position des leds

On choisit d'utiliser le mode Scan sur l'ADC afin de récupérer les 4 tensions provenant des Leds et 1 tension provenant de la batterie. Ce mode permet de convertir plusieurs canaux analogiques de manière séquentielle. Nous pouvons donc mesurer plusieurs tensions analogiques sur un seul et même ADC. Un seul lancement de l'ADC (via la fonction `HAL_ADC_Start_IT` est nécessaire).

Implémenter ce mode SCAN fût complexe et nous avons été confrontés à un problème de décalage des valeurs où l'assignation n'est plus en lien avec la valeur du compteur.

Nous avons ainsi dû abaisser la fréquence globale du système jusqu'à 4MHz (pour une valeur maximale de 80 MHz), nous avons également ajouté un Prescaler de 64 pour la conversion de la valeur de l'ADC.

Nous aurions également pu utiliser l'ADC en mode DMA avec lequel les 5 valeurs auraient été écrites directement dans un buffer mémoire pour être récupérées facilement.

On détecte la fin de l'acquisition et conversion dans l'interruption `HAL_ADC_ConvCpltCallback`.



On met en place un compteur qui comptera jusqu'à 5 pour récupérer une par une les valeurs `HAL_ADC_GetValue`). Pour la batterie, il faut seulement lancer l'acquisition mais aucun `GetValue` n'est nécessaire grâce à l'AnalogWatchdog.

### 3.9. Sens de dégagement du robot :

À présent, nous sommes en mesure de récupérer des informations de batterie et des quatre leds sur l'ADC. Nous pouvons donc commencer à programmer à proprement parler les déplacements que devra pouvoir effectuer notre robot.

Afin de diriger aux mieux le robot lorsque ce dernier fait face à un mur, nous effectuerons la disjonction de cas suivante selon la position relative du robot au mur :

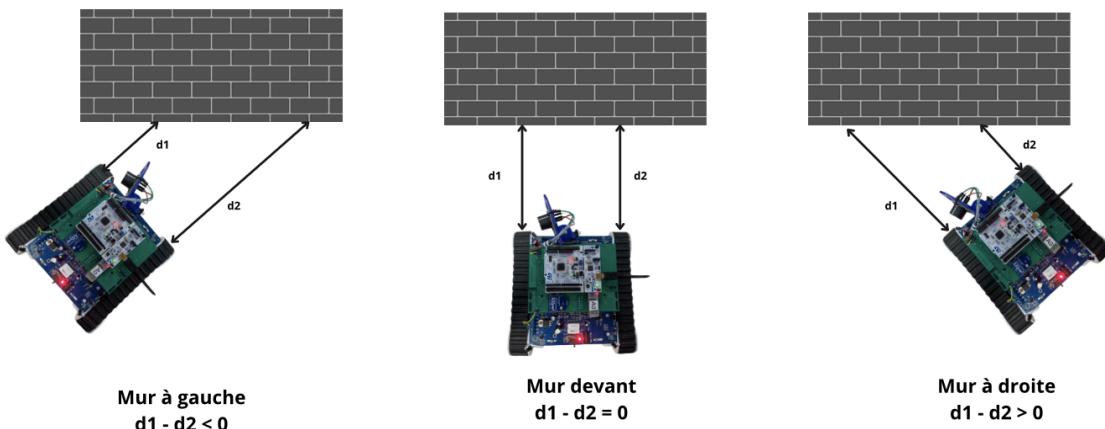


Figure 10 : Disjonction de cas - collision avec le mur

Où  $d1 = d_{\text{left}}$  et  $d2 = d_{\text{right}}$ .

De cette manière, le dégagement se fera par la droite dans la première configuration et par la gauche dans le deuxième et dans le dernier cas ( nous avons choisi arbitrairement de dégager par la gauche dans le cas où le robot arrive droit sur le mur ).

Pour cela, nous comparons la différence de  $d_{\text{right}}$  et  $d_{\text{left}}$  à 0, afin de savoir de quel côté se trouve l'obstacle, puis mettons les DIR des moteurs dans "le bon sens". Par exemple, dans le premier cas, le dégagement du robot se fera par la droite. Donc on met : `DIR_Moteur_droit` prendra la valeur 1 et `DIR_Moteur_gauche` prendra la valeur 0. Ces deux moteurs tournent à la vitesse `target_speed_counter` mais chacuns dans un sens différent.

### 3.10. Améliorations

#### 3.10.1. Accélération progressive - Initialisation du TIM7 et PWM

En piste d'amélioration, nous nous proposons d'implémenter une accélération progressive pour notre robot.



On limite l'accélération maximale à une valeur définie par `max_accel` afin d'éviter des changements brusques de vitesse qui pourraient endommager le matériel ou entraîner un comportement instable. La fonction `compute_speed` est utilisée pour calculer la vitesse suivante en prenant en compte cette limite, en ajustant progressivement la vitesse actuelle vers la vitesse cible (`speedTarget`). Pour cela, à chaque déclenchement du timer nous autorisons le calcul et l'attribution de la nouvelle vitesse qui est :

- la nouvelle vitesse si la valeur absolue de l'écart entre l'ancienne et la nouvelle vitesse est inférieure à l'accélération maximale
- L'ancienne vitesse plus ou moins l'accélération maximale si la valeur absolue de l'écart est trop grande

Ce fonctionnement est expliqué dans l'organigramme page 7.

Dans le code toutes les vitesses sont écrites en  $\mu\text{m}/\text{s}$  afin d'avoir des grandes valeurs permettant une variation plus simple. Ainsi la vitesse maximale est de 2780  $\mu\text{m}/\text{s}$ .

Pour définir l'accélération il est nécessaire de choisir la période entre les modifications de vitesse. Pour cela nous utilisons un timer en mode interruption qui permettra de modifier la vitesse après une période précise. Nous avons choisi de modifier la vitesse toutes les 10ms. Ainsi, toutes les 10ms la vitesse des moteurs sera mise à jour en fonction de la vitesse actuelle et de l'accélération maximale.

Calculons à présent le PSC et ARR pour ce nouveau timer (timer 7) :

$$\text{PSC}_{\text{opti}} > \frac{1.10^{-3} \cdot 4.10^6}{2^{32}} = 8.3 \cdot 10^{-7}$$

Nous prendrons donc  $\text{PSC} = 1$  ;

Enfin,

$$\text{ARR} = 1.10^{-3} \cdot 4.10^6 = 4\,000$$

Pour accélérer nos moteurs, on peut augmenter progressivement le cycle de travail du signal PWM. Au début, le cycle de travail sera bas (par exemple 10 %), ce qui fournit une faible puissance au moteur. En augmentant le cycle de travail (par exemple à 30 %, puis 50 %, etc.), la puissance moyenne appliquée au moteur augmente, ce qui entraîne une accélération du moteur.

La variable `target_speed` représente la vitesse cible du robot. Cette vitesse est ajustée progressivement selon l'accélération maximale permise. L'incrément de cette vitesse cible est géré par les commandes reçues via UART, qui déterminent la direction et la vitesse du robot.

Les vitesses des moteurs gauche (`speed_left`) et droit (`speed_right`) sont ajustées en fonction de cette vitesse cible et des commandes directionnelles reçues ( voir code en Annexe ).



Nous avons défini une accélération maximale correspondant à la vitesse que l'on peut gagner dans les 10 ms d'intervalle. Cela est fait via la variable `max_accel` que nous choisissons à 10cm/s.

### 3.10.2. Anti-rebond

Lorsque l'on presse ou relâche un bouton, celui-ci oscille généralement pendant quelques millisecondes entre les états ouvert et fermé. Par conséquent, durant ces brèves fractions de seconde, le bouton peut basculer de manière aléatoire entre ces états avant de se stabiliser dans l'état attendu, comme illustré par la figure ci-dessous :

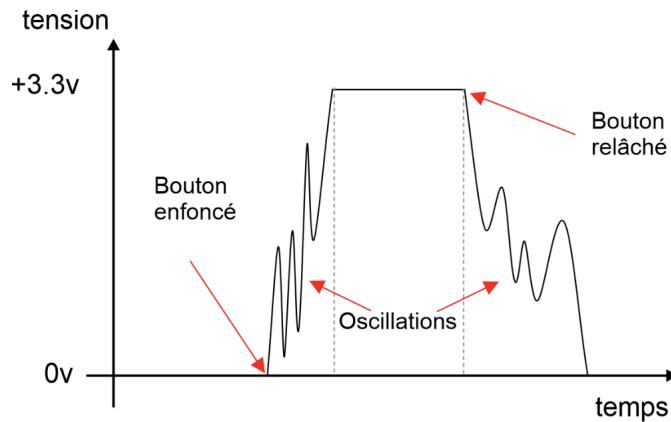


Figure 11 : Rebond sur l'appui d'un bouton

Cependant, nous gérons l'appui sur le bouton (et donc son inversion d'état “allumé” à l’état “éteint”) grâce à une interruption. Or, l’usage de la fonction `Hal_Delay` dans cette même interruption causerait un conflit dans les priorités. Nous avons donc décidé de mettre en place une attente de 200 ms à l'aide de la fonction `HalGetTick()`, qui renvoie le temps passé en millisecondes depuis le dernier reset ou démarrage.

Nous stockons donc dans une variable globale le timing du dernier appui sur le bouton et nous le comparons au nouvel appui.

### 3.10.3. Gestion des coins

Lorsque deux murs sont détectés successivement, le robot est programmé pour reculer afin de se libérer de l’impasse. À la suite de ce recul, le robot reprend automatiquement la dernière commande ou direction donnée par l’utilisateur.

## 4. Réalisation et tests

### 4.1. Module bluetooth : l'uart en fonctionnement

Nous avons observé que le robot répond correctement aux commandes de direction, de vitesse et d’arrêt. Chaque commande a été testée plusieurs fois pour garantir la fiabilité. De plus, les résultats montrent que le module Bluetooth maintient une connexion stable sans perte de données, même en présence



d'interférences mineures, notamment en augmentant l'éloignement entre le téléphone et le robot (nous sommes parvenus à aller au bout d'un couloir, à une distance d'environ 20m).

## 4.2. Test des PWM

Vérifions désormais que nos PWM comportent les bonnes valeurs :

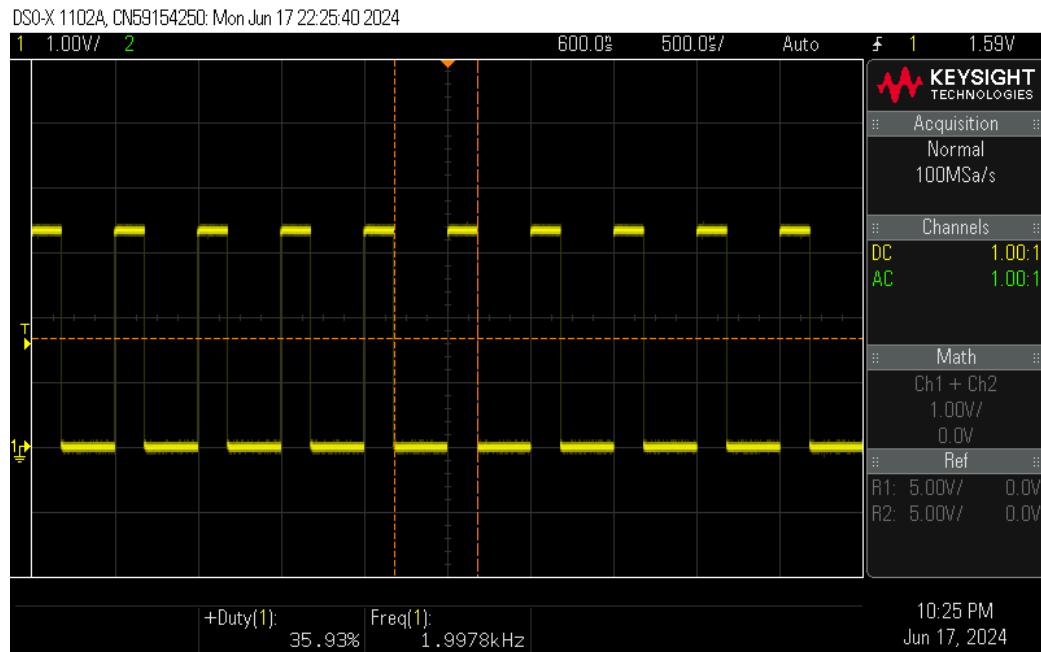


Figure 12 : signal PWM à 36%

Cette première capture montre bien un rapport cyclique d'environ 36% pour le contrôle des moteurs ainsi qu'une fréquence très proche des 2 kHz attendus (le calcul d'erreur relative nous donne  $\epsilon_r = 0.11\%$ ).

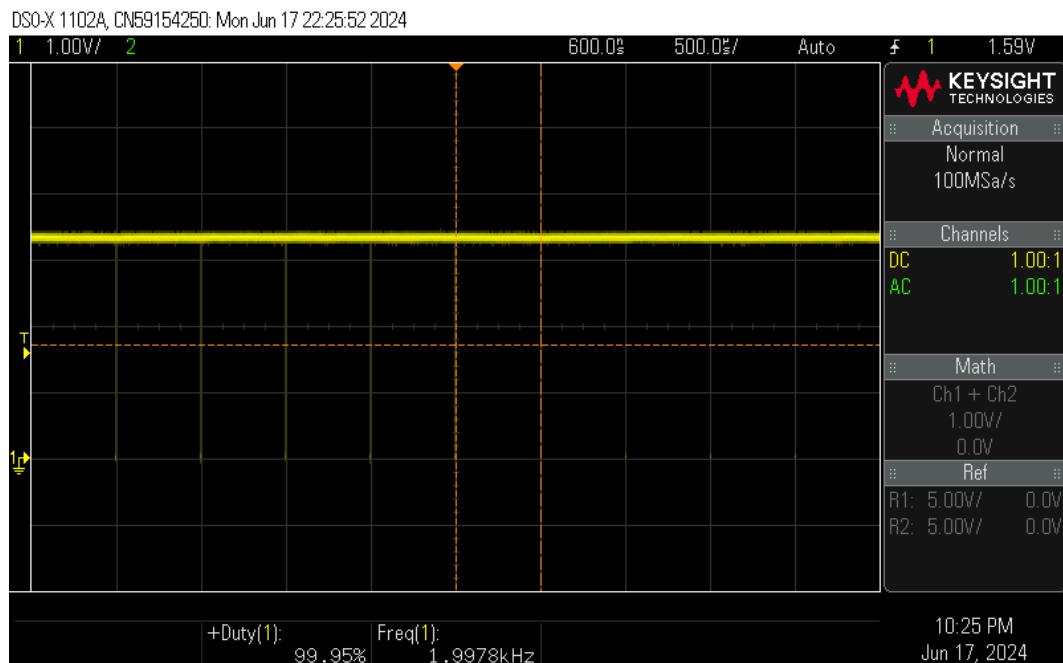


Figure 13 : PWM à 100%



La figure ci-dessus montre quant à elle le PWM avec un rapport cyclique de 1, c'est-à-dire toujours à l'état haut et donc une vitesse maximale des moteurs. On comprend donc ici facilement le concept d'une PWM qui permet d'obtenir une tension moyenne en sortie pour contrôler les moteurs à courant continu.

### 4.3. Sous-programme d'évitement d'obstacles

#### 4.3.1. Tests des leds infrarouges

Afin de tester les leds infrarouges et les codes inhérents à ces dernières, nous pouvons observer les variables d\_right et d\_left, correspondant aux distances d'un potentiel obstacle aux leds en direct.

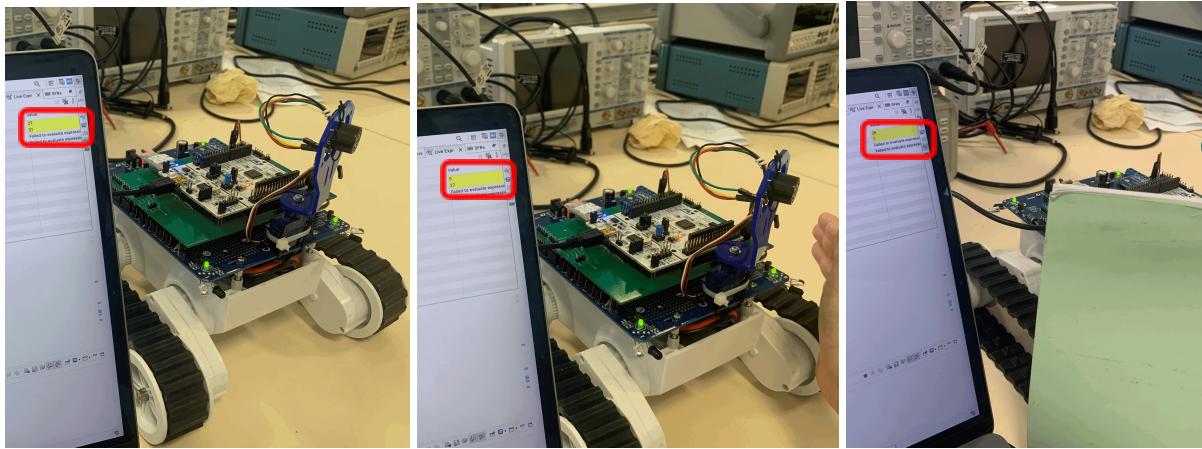


Figure 14 : Distance relevées infrarouges

La première image montre que le robot relève bien des valeurs autour de 27-28 cm, il ne détecte donc bien aucun obstacle devant lui. Dans les deux images qui suivent, on place un objet, en l'occurrence respectivement une main et un cahier en face des leds gauches (en haut dans le tableau) puis droites (en bas dans le tableau). On observe bien une diminution des distances sur ces dernières.

#### 4.3.2. Test du timer 6

Le timer 6 permettant entre autre de mettre en place le scan ADC :



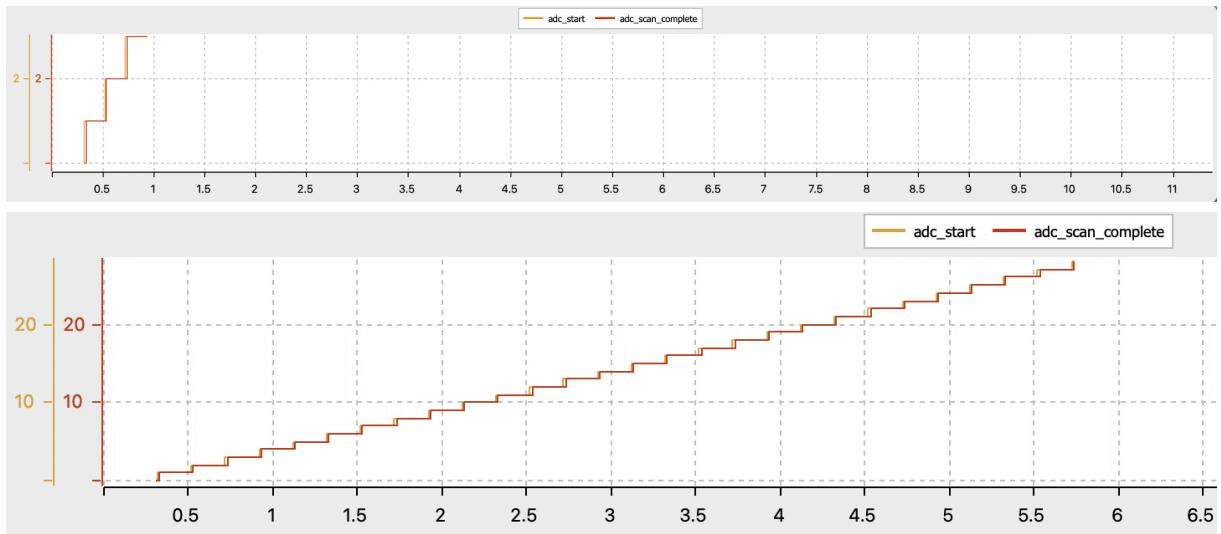


Figure 15 : Contrôle timer 6 par le SWV

On va de 0 à 20 soit 20 incrémentations. On comptabilise donc 5 incrémentations et demi pour une durée de 1s, soit une période de  $\frac{1}{5.5} = 0.181\text{ s}$ . Nous retrouvons bien une valeur proche des 200 ms attendus, avec une erreur absolue de  $|200 - 181| = 19\text{ ms}$  soit une erreur relative de  $\frac{19}{200} \times 100 = 9.5\%$ , sans doute dûe à une erreur humaine dans le relevé.

Nous voyons 2 courbes ; adc\_start et adc\_complete, la première correspond à l'incrémantation au lancement du scan et la seconde à l'incrémantation à la fin de la récupération des valeurs à la fin du scan. Nous avons donc un temps de décalage correspondant à l'acquisition et au temps de calcul pour la conversion.

On constate par ailleurs un bon fonctionnement du scan de l'ADC calé sur ce timer grâce au léger écart présent entre les deux courbes, correspondant au début et à la fin de conversion.

#### 4.4. Tests des améliorations

##### 4.4.1. Timer 7

En plaçant un simple compteur en variable globale puis en l'observant grâce au mode SWV du debugger, on observe la courbe d'évolution suivante. On peut en vérifier la fréquence à l'oeil :

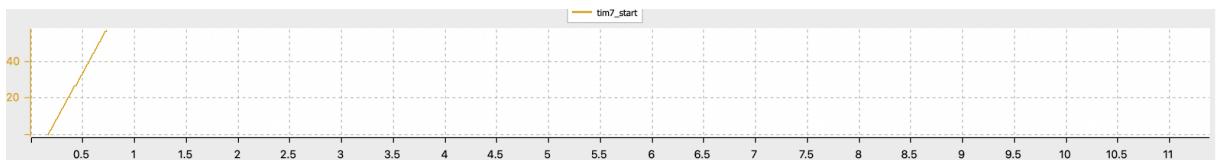


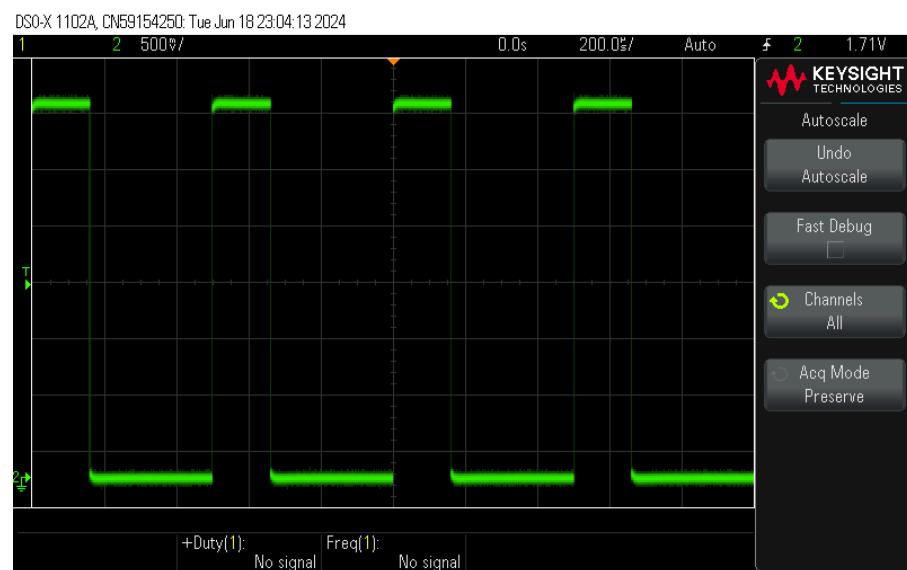
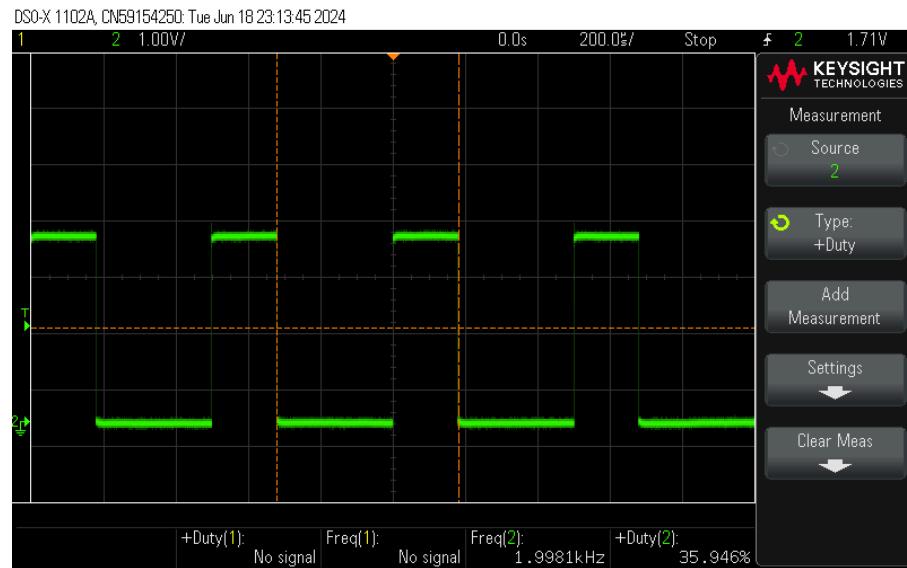
Figure 16 : Contrôle timer 7 par SWV

On relève en effet 52 incrémentations sur une période d'une demie seconde, soit 104 incrémentations par seconde. On a donc bien une période de  $\frac{1}{104} = 0.0096\text{ s} = 9.6\text{ ms}$ . On retrouve bien une période proche des 10 ms attendus.



#### 4.4.2. PWM pour une accélération

Les figures ci-dessous permettent de rendre compte d'une décélération du robot. Le rapport cyclique diminue au fur et à mesure, comme visible sur les figures ci-dessous : .



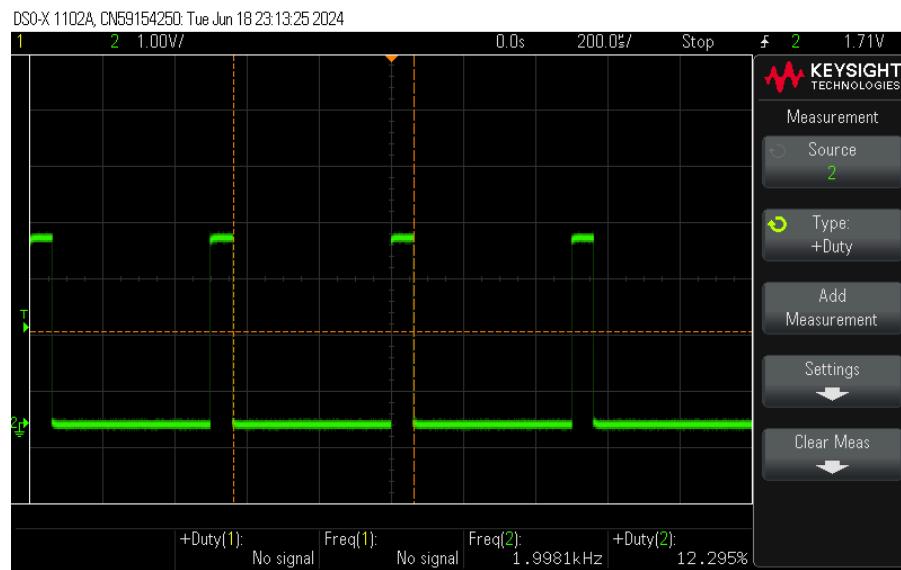


Figure 17 à 20 : Décélération vue sur le signal PWM

#### 4.4.3. Anti-rebond

Une fois le module d'anti-rebond codé dans l'interruption, nous avons implémenté un compteur qui s'incrémente à chaque appui du bouton et n'avons pas observé de défaillance, un seul appui est compté.

#### 4.4.4. Coins

Les tests ont été conformes à notre code mais il nous a été fait remarquer que le robot aurait pu continuer de tourner au lieu de simplement reculer puis de se bloquer à cause de la commande avant encore en mémoire du buffer.

### 5. Conclusion

#### 5.1. Évolutions possibles du contrat



Au cours de notre projet, nous avons identifié plusieurs axes d'amélioration pour optimiser le fonctionnement et les capacités de notre robot.

L'une des premières améliorations serait l'utilisation des LEDs arrière. Actuellement, notre robot utilise des LEDs infrarouges à l'avant pour détecter les obstacles et éviter les collisions. En ajoutant des LEDs à l'arrière, nous pourrions améliorer la capacité du robot à se dégager efficacement lorsqu'il se trouve dans un coin ou un espace confiné.

Une autre amélioration significative serait l'intégration du sonar, présent sur les contrats plus évolués. Ce dernier utilise des ondes sonores pour détecter les objets et mesurer les distances. Il émet des impulsions sonores (ultrasons) qui se réfléchissent sur les objets et reviennent vers le capteur. Ce dernier aurait permis une détection précise des obstacles. De plus, le sonar est moins affecté par les conditions d'éclairage que les capteurs infrarouges, ce qui permettrait au robot de fonctionner de manière fiable dans des environnements sombres ou très lumineux.

## 5.2. Applications sur le marché

Les technologies mises en place dans notre projet, comme la connexion Bluetooth, l'évitement d'obstacles et la gestion de la batterie, pourraient trouver des applications concrètes dans les robots domestiques tels que les aspirateurs et les tondeuses autonomes. Par exemple, les robots aspirateurs comme le *Roomba* utilisent des capteurs infrarouges pour naviguer et éviter les obstacles, tandis que les tondeuses autonomes adaptent leur trajectoire pour contourner des objets et retourner à leur base de chargement lorsque la batterie est faible.

Ce fonctionnement avec les Leds infrarouges est moins consommateur d'énergie qu'une solution avec Lidar utilisant un servo-moteur ce qui peut amener à une baisse rapide de la charge de la batterie. Les Leds permettent également une acquisition en temps réel sans jamais arrêter le robot.

En somme, le projet de robot autonome nous a permis de mettre en pratique de nombreux concepts, notamment la gestion des périphériques avec CubeIDE, la programmation des signaux PWM, et l'utilisation des capteurs infrarouges pour l'évitement d'obstacles. Les résultats obtenus sont satisfaisants et montrent que le robot répond efficacement aux exigences du contrat n°6. Ce projet ouvre également des perspectives pour des améliorations, telles que l'implémentation d'une accélération progressive et la gestion des rebonds des boutons, afin d'optimiser encore les performances du système.

Si ce contrat était amené à évoluer, il pourrait être intéressant de demander un évitement spécifique pour les coins où le robot effectue un demi-tour. Obliger à utiliser le DMA pourrait également nous permettre d'utiliser une autre fonctionnalité de la carte STM32 souvent plus facile à utiliser que la carte STM32.

## 6. Tables des figures

Figure 1 : Configuration pins STM32	4
Figure 2 à 4 : Algorigramme complet	7
Figure 5 : Formules initialisation, A.Marques	8



Figure 6 : Pont de hacheur	9
Figure 7 : Application de commande du robot	11
Figure 8: Régression puissance pour les caractéristiques des infrarouges	13
Figure 9 : Position des leds	14
Figure 10 : Disjonction de cas - collision avec le mur	15
Figure 11 : Rebond sur l'appui d'un bouton	17
Figure 12 : signal PWM à 36%	18
Figure 13 : PWM à 100%	18
Figure 14 : Distance relevées infrarouges	19
Figure 15 : Contrôle timer 6 par le SWV	20
Figure 17 à 20 : Décélération vue sur le signal PWM	21

## 7. Annexes

```
/* USER CODE END Header */
/* Includes
-----
#include "main.h"
/* Private includes
-----
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include<stdio.h>
#include<stdlib.h>
#include <math.h>
/* USER CODE END Includes */
/* Private typedef
-----
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define
-----
/* USER CODE BEGIN PD */
#define vmax_counter 2000
#define vmax 2780//2304//en um/s car 4,34 s/m // 27.8 m/s en théorie
#define max_accel 10//en um/10ms donc cm/s 10um/10ms soit 1000 um/s
//on actualise ttes les 5ms
//l'accel max dans ce temps est : max_accel*
/* 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;
TIM_HandleTypeDef htim7;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
```



```

/* USER CODE BEGIN PV */
int target_speed = 0;
uint16_t Batt_value;
volatile char cnt_adc_scan = 0;
volatile uint16_t FL_value = 0, BL_value = 0, FR_value = 0, BR_value = 0;
volatile uint16_t ADC_current_value = 0;
bool GO = false;
volatile uint8_t adc_on_nb = 0;
// UART
int flag_uart;
unsigned char Rx_buffer;
char uart_direction;
int d_right, d_left;//distance à droite, distance à gauche
int flag_uart = 0;
int dist_detection = 20;
int nominal_speed;
int speed_left;
int speed_right;
int nominal_target_speed_counter;
bool period_elapsed;
uint32_t last_trigger_bp = 0;
char msg[80];
/* USER CODE END PV */
/* Private function prototypes
-----*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM6_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_TIM7_Init(void);
static void MX_USART2_UART_Init(void);
/* USER CODE BEGIN PFP */
void capteur_mur(void);
bool get_direction(int speed);
int compute_speed(int currentSpeed, int speedTarget);
int speed_to_ARR(int speed);
/* USER CODE END PFP */
/* 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_ADC1_Init();
MX_TIM2_Init();
MX_TIM6_Init();
MX_USART3_UART_Init();
MX_TIM7_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */
    nominal_speed = 1000;
    nominal_target_speed_counter = vmax_counter*nominal_speed/vmax;
    //target_speed = 0;
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4, 0);
    //HAL_GPIO_WritePin(Alert_batt_GPIO_Port, Alert_batt_Pin,
GPIO_PIN_SET);
    HAL_TIM_Base_Start_IT(&htim6);
    HAL_TIM_Base_Start_IT(&htim7);
    //HAL_UART_Transmit_IT(&huart3, (uint8_t)tab, sizeof(tab));
    HAL_UART_Receive_IT(&huart3, &Rx_buffer, 1);
    d_right = 100;
    d_left = 100;//On initialise dd et dq à une valeur supérieure au seuil.
Ils serons modifiés dès le premier lancement de l'ADC.
    bool detection_ended = false;
    //HAL_ADC_Start_IT(&hadc1);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
    //HAL_ADC_Start_IT(&hadc1);
while (1)
{
    HAL_GPIO_WritePin(IR1_CMD_GPIO_Port, IR1_CMD_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IR4_CMD_GPIO_Port, IR4_CMD_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IR2_CMD_GPIO_Port, IR2_CMD_Pin, GPIO_PIN_SET);
    HAL_GPIO_WritePin(IR3_CMD_GPIO_Port, IR3_CMD_Pin, GPIO_PIN_SET);
    if(GO)
    {
        // Les leds seront tout le temps allumées pour éviter les
perturbations
        if(d_right < dist_detection || d_left < dist_detection) // mur
déTECTé
    }
}

```



```

        capteur_mur();
        //On mets brièvement la PWM à 0
        // __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, 0); //moteur
gauche
        //__HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4, 0); //moteur
droit
        detection_ended = true;
    }
else if(period_elapsed || detection_ended)
{
    period_elapsed = false;
    detection_ended = false;
    switch (uart_direction)
    {
        case 'F': //Front
        {
            speed_left = compute_speed(speed_left,
nominal_speed);
            speed_right = compute_speed(speed_right,
nominal_speed);
            break;
        }
        case 'B': //back
        {
            speed_left = compute_speed(speed_left,
-nominal_speed);
            speed_right = compute_speed(speed_right,
-nominal_speed);
            break;
        }
        case 'L' : //left
        {
            speed_left = compute_speed(speed_left,
+nominal_speed);
            speed_right = compute_speed(speed_right,
-nominal_speed);
            break;
        }
        case 'R' : //right
        {
            speed_left = compute_speed(speed_left,
-nominal_speed);
            speed_right = compute_speed(speed_right,
+nominal_speed);
            break;
        }
        case 'W' : //Appel de phare, mode super_vitesse
        {
            speed_left = compute_speed(speed_left, vmax);
            speed_right = compute_speed(speed_right, vmax);
            break;
        }
        case 'X' : //Warning : stop
    }
}

```



```

        {
            speed_left = compute_speed(speed_left, 0);
            speed_right = compute_speed(speed_right, 0);
            break;
        }
        default :
        {
            speed_left = compute_speed(speed_left, 0);
            speed_right = compute_speed(speed_right, 0);
        }
    }
    // on applique sur la PWM et sur le choix de la direction
la vitesse calculée précédemment
    HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin,
get_direction(speed_left));//moteur droit
    HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin,
get_direction(speed_right));//moteur droit
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,
speed_to_ARR(speed_left));//moteur gauche
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4,
speed_to_ARR(speed_right));//moteur droit
}
}
else
{
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_4, 0);
    //flag_uart = 0; //plus utile
}
/* 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();
    }
    /**
     * Configure LSE Drive Capability
     */
    HAL_PWR_EnableBkUpAccess();
    __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
}

```



```

/** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
RCC_OscInitStruct.LSEState = RCC_LSE_ON;
RCC_OscInitStruct.MSISState = RCC_MSI_ON;
RCC_OscInitStruct.MSICalibrationValue = 0;
RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
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_MSI;
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_0) != HAL_OK)
{
    Error_Handler();
}
/** Enable MSI Auto calibration
 */
HAL_RCCEx_EnableMSIPLLMode();
}
/** @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_DIV64;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
    hadc1.Init.LowPowerAutoWait = DISABLE;
}

```



```

hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.NbrOfConversion = 5;
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_OVERWRITTEN;
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_3;
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();
}
/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_4;
sConfig.Rank = ADC_REGULAR_RANK_2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/** Configure Regular Channel
*/

```



```

sConfig.Channel = ADC_CHANNEL_16;
sConfig.Rank = ADC_REGULAR_RANK_3;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_13;
sConfig.Rank = ADC_REGULAR_RANK_4;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
    Error_Handler();
}
/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_14;
sConfig.Rank = ADC_REGULAR_RANK_5;
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_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 = 2000;
    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
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 = 61538-1;
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 TIM7 Initialization Function
 * @param None
 * @retval None
 */

```



```

/*
static void MX_TIM7_Init(void)
{
    /* USER CODE BEGIN TIM7_Init_0 */
    /* USER CODE END TIM7_Init_0 */
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    /* USER CODE BEGIN TIM7_Init_1 */
    /* USER CODE END TIM7_Init_1 */
    htim7.Instance = TIM7;
    htim7.Init.Prescaler = 1-1;
    htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim7.Init.Period = 40000;
    htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
    {
        Error_Handler();
    }
    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN TIM7_Init_2 */
    /* USER CODE END TIM7_Init_2 */
}
*/
/* @brief 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_8;
    huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart3) != HAL_OK)
    {
        Error_Handler();
    }
    /* USER CODE BEGIN USART3_Init_2 */
    /* USER CODE END USART3_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(GPIOA, Alert_batt_Pin|IR3_CMD_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOB, DIR2_Pin|IR1_CMD_Pin|IR4_CMD_Pin|IR2_CMD_Pin,
GPIO_PIN_RESET);
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, GPIO_PIN_RESET);
    /*Configure GPIO pin : BP_Pin */

```



```

GPIO_InitStruct.Pin = BP_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(BP_GPIO_Port, &GPIO_InitStruct);
/*Configure GPIO pins : Alert_batt_Pin IR3_CMD_Pin */
GPIO_InitStruct.Pin = Alert_batt_Pin|IR3_CMD_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
/*Configure GPIO pins : DIR2_Pin IR1_CMD_Pin IR4_CMD_Pin IR2_CMD_Pin */
GPIO_InitStruct.Pin = DIR2_Pin|IR1_CMD_Pin|IR4_CMD_Pin|IR2_CMD_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 : 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) //l'origine de
l'interruption est donnée
{
    if(htim->Instance == TIM6)
    {
        /*Si nous avions voulu allumer les leds infra, ce serait ici */
        HAL_ADC_Start_IT(&hadc1);
    }
    if(htim->Instance == TIM7)
    {
        period_elapsed = true;
    }
}
int adc_value_to_dist(float a, float b, int p)
{
    //int dist = (int)((1.0f/a)*log(b/((float)p))); // regression
exponentielle
    int dist = powf(b/(float)p, 1/a); // regression "puissance"
    return dist;
}
int compute_speed(int currentSpeed, int speedTarget)
{
    int nextSpeed;
    int diff = abs(speedTarget - currentSpeed);
}

```



```

    if(speedTarget > currentSpeed)
    {
        if(speedTarget <= 0)
        {
            if(diff > max_accel) diff = max_accel;
            nextSpeed = currentSpeed + diff;
        }
        else
        {
            if(diff > max_accel) diff = max_accel;
            nextSpeed = currentSpeed + diff;
        }
    }
    else if(speedTarget < currentSpeed)
    {
        if(speedTarget < 0)
        {
            if(diff > max_accel)
                diff = max_accel;
            nextSpeed = currentSpeed - diff;
        }
        else
        {
            if(diff > max_accel)
                diff = max_accel;
            nextSpeed = currentSpeed - diff;
        }
    }
    else
    {
        nextSpeed = currentSpeed;
    }
    /*if(abs(nextSpeed) > maxSpeed) //necessaire si on veut une
vitesse
    {
        nextSpeed = nextSpeed/abs(nextSpeed) * maxSpeed;
    }*/
    return nextSpeed;
}
int speed_to_ARR(int speed)
{
    int nominal_target_speed_counter = vmax_counter*abs(speed)/vmax;//en
valeur absolue
    return nominal_target_speed_counter;
}
bool get_direction(int speed)
{
    return speed > 0 ? 1 : 0;
}
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    //sprintf(msg, "Tension batterie = %d, vitesse droite = %d,
vitesse gauche = %d\r\n", Batt_value, speed_right, speed_left);
}

```



```

        //HAL_UART_Transmit(&huart2, (uint8_t*) msg, sizeof(msg),
HAL_MAX_DELAY);
    if(cnt_adc_scan == 0)//gauche
    {
        FR_value = HAL_ADC_GetValue(&hadc1);
        d_right = adc_value_to_dist(0.942,4937.5,
FR_value);//TODO ici modifier coeffs
    }
    if(cnt_adc_scan == 1)//droite
    {
        FL_value = HAL_ADC_GetValue(&hadc1);
        d_left = adc_value_to_dist(0.942,4937.5, FL_value); //TODO ici modifier coeffs
    }
    if(cnt_adc_scan == 2)
    {
        BL_value = HAL_ADC_GetValue(&hadc1);
    }
    if(cnt_adc_scan == 3)
    {
        BR_value = HAL_ADC_GetValue(&hadc1);
    }
    if(cnt_adc_scan == 4)
    {
        Batt_value = HAL_ADC_GetValue(&hadc1);//Ce get_value n'est pas nécessaire,
//nous le récupérons quand même à des fins de débogage
puisque nous
        //faisons déjà un scan
        cnt_adc_scan=-1;
    }
    cnt_adc_scan = cnt_adc_scan+1;
}
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc)
{
    HAL_GPIO_WritePin(Alert_batt_GPIO_Port, Alert_batt_Pin, GPIO_PIN_SET);
}
//inversion du booléen GO lors d'un appui sur le bouton bleu
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    uint32_t actual_time = HAL_GetTick();
    if (GPIO_Pin == BP_Pin && actual_time - last_trigger_bp > 7)//plus logique vers les centaines de ms
    {
        GO = !GO;
        last_trigger_bp = actual_time;
    }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART3)
    {
        if(flag_uart % 2 ==0)

```



```

    {
        if(Rx_buffer != 'S')
            sscanf(&Rx_buffer, "%c", &uart_direction);
    }
    HAL_UART_Receive_IT(&huart3, &Rx_buffer, 1); //on relance après
réception
    UNUSED(huart);
    if(Rx_buffer != 'D') {
        //si on se déconnecte et reconnecte il envoie un D pour
indiquer qu'il
        //y a eu une connexion. Ce D ne doit donc pas être pris en
compte comme une commande
        flag_uart++;
    }
}
//uint8_t *coucou;
//sscanf(Rx_buffer, "%hu", coucou );
}

void capteur_mur()//d1 = d_left et d2 = d_right
{
    bool avoidanceEnded = false;
    int diff;
    while(!avoidanceEnded && GO)
    {
        diff = d_left - d_right;//on recalcule à chaque fois
        if(d_left <= dist_detection && d_right <= dist_detection) //bloqué
dans un coin
        {
            HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, 0); //droit
            HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, 0); //gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,
nominal_target_speed_counter); //moteur gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,
nominal_target_speed_counter); //moteur droit
        }
        else if(diff >= 0)//mur à droite, on tourne à gauche
        //Si le mur est en face on choisit d'aller à gauche
        {
            HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, 1); //droit
            HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, 0); //gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,
nominal_target_speed_counter); //moteur gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,
nominal_target_speed_counter); //moteur droit
        }
        else if(diff < 0) //mur à gauche on tourne à droite
        {
            HAL_GPIO_WritePin(DIR1_GPIO_Port, DIR1_Pin, 0); //droit
            HAL_GPIO_WritePin(DIR2_GPIO_Port, DIR2_Pin, 1); //gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1,
nominal_target_speed_counter); //moteur gauche
            __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4,
nominal_target_speed_counter); //moteur droit
        }
    }
}

```



```

        }
        if(d_right > dist_detection && d_left > dist_detection)
            avoidance_ended = true;
    }
}

/* 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 */

```

