PROJET ROBOT : CONTRAT 10

PROJET ROBOT

SEMESTRE 6

Auteur :

Youssef ENNOURI

Fares GHARBI

Professeur :

Acacio MARQUES

Date :

Juin 2024

SOMMAIRE

INTRODUCTION

L'objectif du projet robot est de se familiariser avec les cartes et l'environnement STM32, tout en acquérant de l'expérience pratique avec le débogage. Dans le cadre du contrat 10, notre mission consiste à piloter un robot à l'aide d'une application téléphonique via Bluetooth. Les fonctionnalités principales du robot incluent l'accélération, la décélération et le changement de direction en réponse aux commandes de l'utilisateur.

Le robot sera asservi par un correcteur PI et devra se déplacer selon les instructions données par la croix directionnelle de l'application téléphonique. Chaque direction sur la croix correspond à trois vitesses distinctes. Chaque pression supplémentaire dans une direction augmente la vitesse, tandis qu'une pression dans la direction opposée diminue la vitesse, jusqu'à l'arrêt complet.

Le contrat doit remplir plusieurs objectifs :

- Fréquence de PWM de 500 Hz :

  La fréquence de la modulation de largeur d'impulsion (PWM) utilisée pour contrôler les moteurs du robot doit être fixée à 500 Hz.

- Différentes vitesses du robot :

  Le robot doit pouvoir se déplacer à quatre vitesses distinctes : 0 cm/s (arrêt), 10 cm/s, 20 cm/s, et 25 cm/s[1].

- Précision de +/- 2 cm/s :

  La vitesse de déplacement du robot doit être maintenue avec une précision de +/- 2 cm/s. Cette précision est importante pour garantir que le robot suit les commandes de l'utilisateur de manière fiable et constante.

- Asservissement PI :

  Un asservissement proportionnel-intégral (PI) doit être implémenté pour le contrôle du robot. L'asservissement PI permet de maintenir la stabilité et la précision de la vitesse du robot en corrigeant les erreurs entre la consigne (vitesse souhaitées) et la valeur réelle mesurée.

- Décalage en ligne droite limité à 10 cm sur 3 m :

  Lors des déplacements en ligne droite, le robot ne doit pas s'écarter de plus de 10 cm par rapport à la trajectoire idéale sur une distance de 3 m.

[1] La vitesse max a été plafonné à 25 cm/s et non 30 cm/s, nous l'expliquons par la suite dans la partie liée à l'asservissement.

Le robot que nous allons utiliser tout au long de notre projet dispose d'une plateforme Rover 5 à chenilles qui est équipée de 2 moteurs avec 2 encodeurs. Une carte mère est également présente sur le robot pour permettre le contrôle des 2 moteurs et gérer la recharge de celui-ci. Enfin, le robot est également composé d'une carte NUCLEO-L476RG qui sert de contrôleur principal, gérant les différentes fonctions et interactions du robot. Nous utiliserons dans notre projet l'environnement de développement STM32CubeIDE qui va nous permettre de configurer les périphériques nécessaires à notre étude, programmer et débugger notre carte STM32.

Le présent rapport est découpé en 2 grandes parties. La partie conception décrit en détail la conception du système, incluant le fonctionnement des différents algorithmes, les organigrammes correspondants de ceux-ci et le fonctionnement des différents périphériques. La partie réalisation et tests se concentre sur la configuration des périphériques, les calculs nécessaires (comme les gains et timers), les tests des fonctions et la vérification des résultats.

CONCEPTION

ARCHITECTURE GLOBALE (MAIN)

Figure 1 - Architecture Générale du Code (Main)

La figure 1 représente l'algorigramme du programme principal du robot. Le programme commence par vérifier la tension de la batterie. Pour mesurer la tension de la batterie nous utilisons un ADC. Un ADC est un convertisseur analogique-numérique qui permet de transformer un signal analogique (comme une tension variable) en une valeur numérique que le microcontrôleur peut traiter. L'ADC est démarré pour commencer la mesure de la tension de la batterie. Le programme attend que la conversion de l'ADC soit terminée et lit ensuite la valeur de Vbatt. Cette valeur est comparée à un seuil prédéfini (3V). Si la tension de la batterie est inférieure à ce seuil, une LED s'allume pour indiquer que la batterie est faible. Dans le cas inverse la LED reste éteinte.

Le programme appelle ensuite la fonction 'Detecteur_de_requete()', qui vérifie et traite les requêtes reçues par le robot via l'UART. Il vérifie ensuite si le temps écoulé pour l'asservissement (Temps_Asser) est supérieur ou égal à un temps d'attente prédéfini (Attente) qui est de 200 ms. Si c'est le cas, le temps d'asservissement est réinitialisé à zéro (Temps_Asser = 0), et la fonction 'asservissement()' est appelée pour ajuster le mouvement du robot. L'asservissement est effectué toutes les 200 millisecondes, celui-ci est activé à l'aide de la variable Temps_Asser. Celle-ci est incrémentée toutes les 2 millisecondes à l'aide d'une interruption sur le Timer 2. Il suffit donc que cette variable atteigne la valeur 100 pour que 200 millisecondes se soient passées.

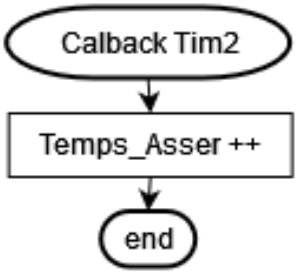

Figure 2 - Callback Timer 2

En parallèle, le programme surveille l'état du bouton poussoir B1 grâce à une interruption. En effet, le bouton poussoir B1 joue un rôle important car le robot ne pourra se déplacer que si celui-ci est activé. Celui-ci joue le rôle d'un start and stop qui permet de démarrer et d'arrêter le robot. Il est initialisé à 0 au début du programme, et chaque appui commute son état. Le premier appui autorise le démarrage du robot, tandis que le deuxième appui entraîne un arrêt d'urgence. Cette fonctionnalité est gérée par une interruption qui surveille les changements d'état du bouton poussoir.

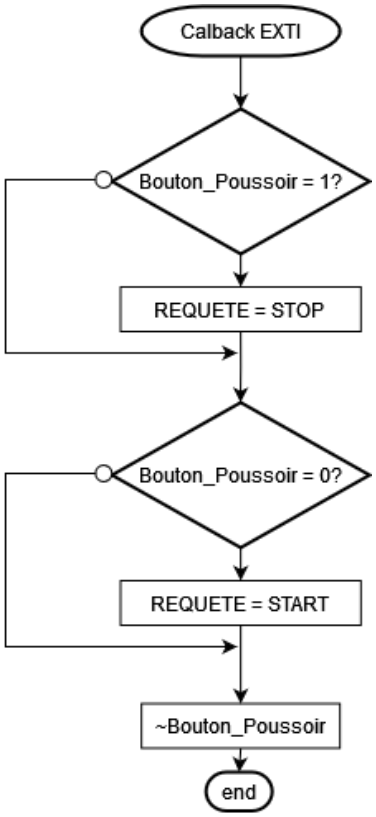

Figure 3 - Callback du Bouton Poussoir

DÉTECTION DES COMMANDES VIA BLUETOOTH

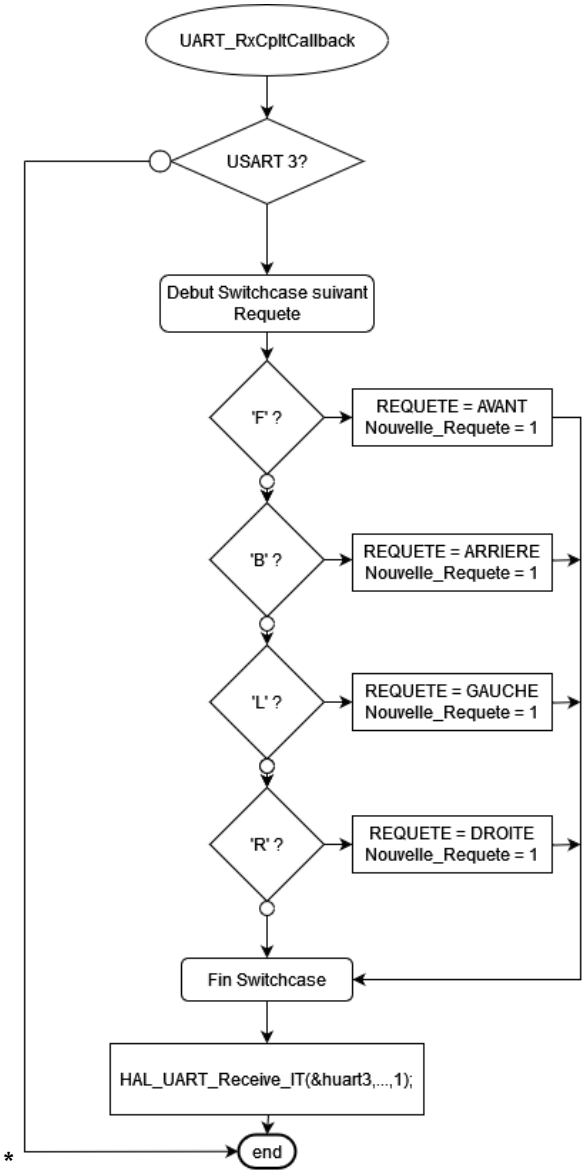

Figure 4 - Détection des Commandes Bluetooth

Les commandes du robot sont gérées par le callback « HAL_UART_RxCpltCallback » qui est responsable de la détection d'une nouvelle demande qui est envoyé par l'opérateur à travers une interface de communication série ‘Arduino Bluetooth Control’. L’interruption arrête le programme chaque fois qu'une donnée nouvelle est capturée sur le port série, permettant ainsi une interprétation directe et rapide des commandes saisies par l'utilisateur.

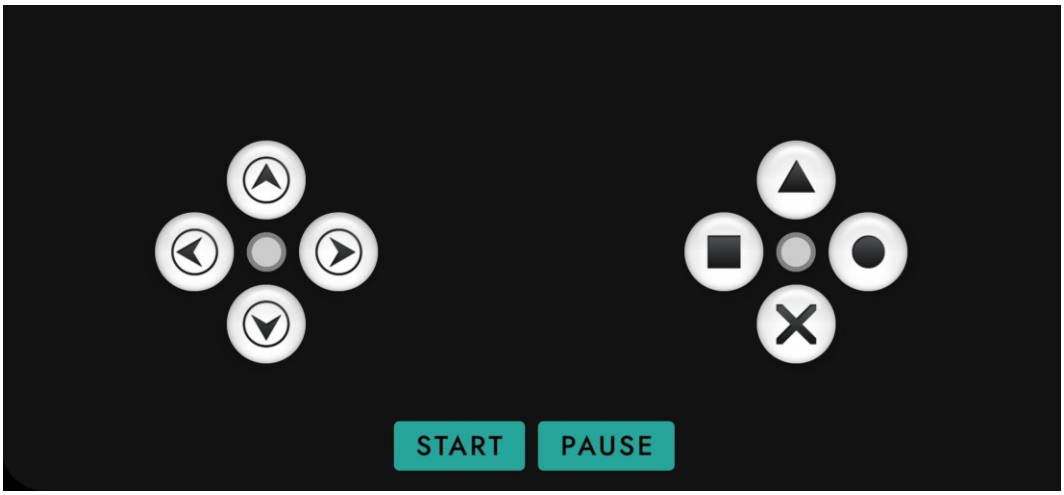

Figure 5 - Interface Arduino Bluetooth Controller

Le périphérique matériel utilisé pour la communication série est l'USART3.
Lorsqu'une nouvelle donnée est reçue par le robot sur l'USART3, l'interruption est activée. Cette fonction effectue alors les tâches suivantes :

- La nouvelle requête est lue et stockée dans la variable 'BLUE_RX'.
- En fonction de la valeur de 'BLUE_RX', la variable 'REQUETE' est mise à jour pour la prochaine commande (par exemple, 'F' pour avancer, 'B' pour reculer, 'L'pour tourner à gauche, 'R' pour tourner à droite).
- La variable 'Nouvelle_Requete' est initialisée à '1', ce qui équivaut pratiquement pour le programme principal à une nouvelle requête reçue et à recevoir.
- La fonction 'HAL_UART_Receive_IT' est appelée pour réactiver la réception de données afin que le système puisse écouter d'autres commandes une fois la première reçue.

La variable 'Nouvelle_Requete' est ensuite utilisée dans 'Detecteur de Requete()'.
À chaque itération du programme principal, 'Detecteur de Requête()' vérifie la variable 'Nouvelle_Requête'. Si sa valeur est '1', alors cela signifie qu'une nouvelle requête a été mise en route.

Lorsqu'une nouvelle requête arrive, la variable 'Nouvelle_Requete' est remise à '0' pour signaler qu'une requête est en cours de traitement.

La fonction 'Detecteur de Requete' et le callback sur l'USART3 fonctionnent en tandem pour traiter les instructions de l'utilisateur rapidement et de manière cohérente. Les fonctions UART détectent les nouvelles requêtes et détermine quels paramètres sont concernés, puis cette information est utilisée par 'Detecteur de Requete' pour déterminer l'état et le comportement souhaités du robot.

TRAITEMENT DES RÊQUETES ET ACTIONS CORRESPONDANTES

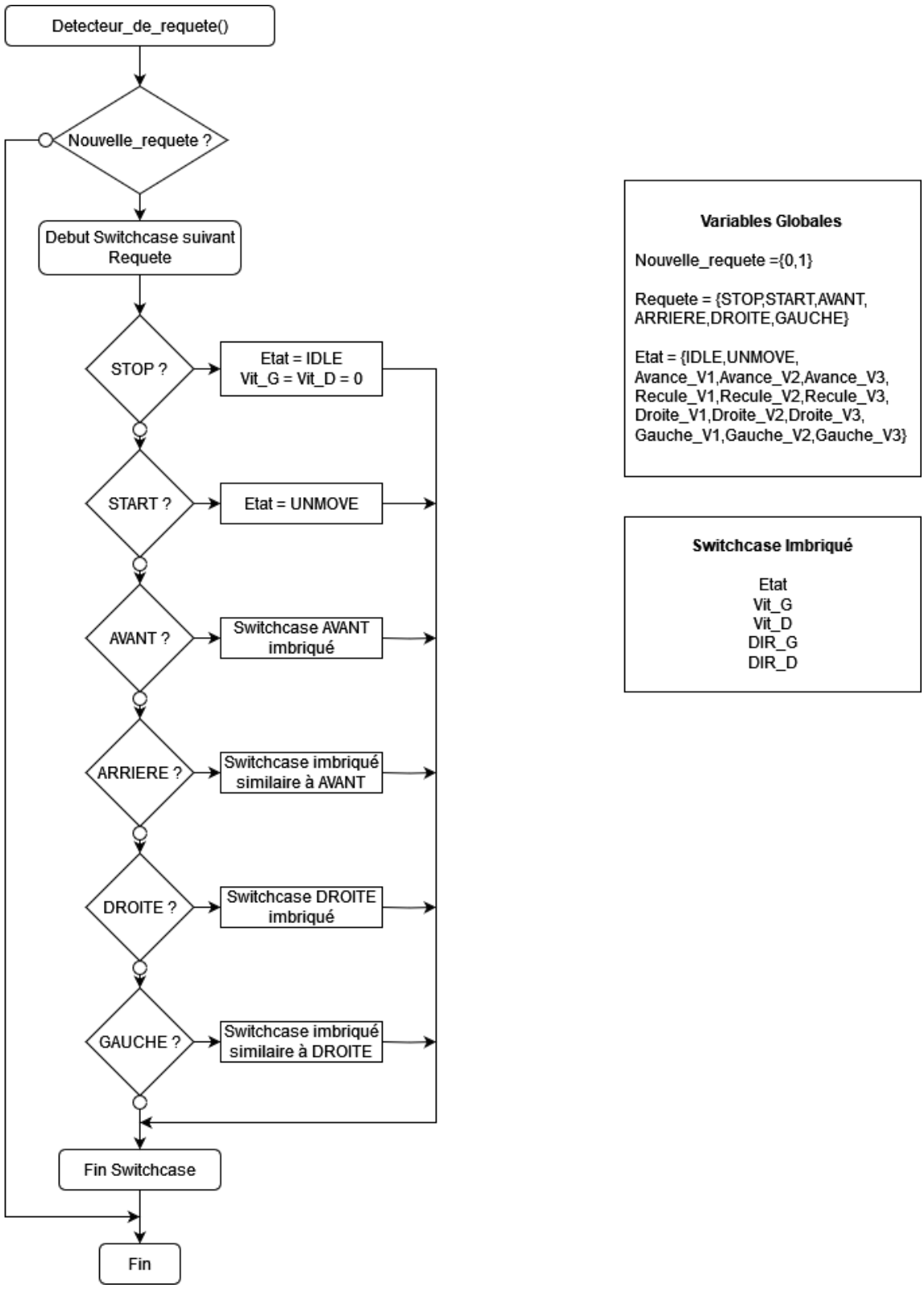

Figure 6 - Algorigramme Détecteur de Requête

La fonction ‘Detecteur_de_requete’ gère le mouvement du robot en conséquence des différentes requêtes de l'utilisateur. Il le fait avec l'aide d'un ensemble de structures ‘switchcase’ imbriquées, où les requêtes sont exécutées de manière séquentielle. La fonction commence par vérifier si une nouvelle requête est émise par l'utilisateur à l’aide de la variable ‘Nouvelle_requete’. Ensuite, le switchcase est atteint et la variable `Nouvelle_Requete` est réinitialisée à ‘0’.

L’état du robot est mémorisé à l’aide de la variable de type enum ‘Etat’ qui définit un ensemble de variables représentant les différents états du robot. On initialise alors la variable ‘Etat’ qui est utilisée pour stocker et manipuler l'état du système en utilisant les valeurs définies dans l'énumération ‘ETAT’. Ainsi, une instruction ‘switchcase’ initiale examine l’instruction de l’utilisateur (par exemple ‘START’, ‘STOP’, ‘AVANT’, ‘ARRIERE’, ‘DROITE’, ‘GAUCHE’). Chaque cas de l'instruction `switch-case` initiale inclut une autre instruction `switch-case` basée sur l'état courant du robot (‘Etat’) pour décider de l'action suivante.

La variable ‘Etat’ contient alors l'état actuel du robot, que l’on peut utiliser pour connaitre l’état futur du robot en fonction de la requête de l’utilisateur. Par exemple, si le robot est déjà en train de se déplacer à une certaine vitesse et qu'une requête est émise pour accélérer, l'état précédent auquel le robot était (état actuel : avancer à la vitesse 1) est renvoyé pour passer au prochain état (avancer à la vitesse 2). Dans les changements d'état, les vitesses et la direction des moteurs droits et gauches sont modifiées. C'est la base sur laquelle le robot pourra se déplacer en adéquation avec chaque commande qu'il reçoit en très peu de temps.

Pour déterminer le comportement du robot à chaque instruction et donc la transition entre les différents états, nous avons modélisé le problème sous la forme d’une machine d’état. On a alors construit le tableau représentant les états futurs en fonction des états présents.

	IDLE	UNMOVE	Avance_V1	Avance_V2	Avance_V3	Recule_V1	Recule_V2
STOP	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE
START	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE
AVANT	IDLE	Avance_V1	Avance_V2	Avance_V3	Avance_V3	UNMOVE	Recule_V1
ARRIERE	IDLE	Recule_V1	UNMOVE	Avance_V1	Avance_V2	Recule_V2	Recule_V3
DROITE	IDLE	Droite_V1	Droite_V1	Droite_V2	Droite_V3	Droite_V1	Droite_V2
GAUCHE	IDLE	Gauche_V1	Gauche_V1	Gauche_V2	Gauche_V3	Gauche_V1	Gauche_V2

Figure 7 - Transition États Robots (1)

	Recule_V3	Droite_V1	Droite_V2	Droite_V3	Gauche_V1	Gauche_V2	Gauche_V3
STOP	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE	IDLE
START	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE	UNMOVE
AVANT	Recule_V2	Avance_V1	Avance_V2	Avance_V3	Avance_V2	Avance_V2	Avance_V3
ARRIERE	Recule_V3	Recule_V1	Recule_V2	Recule_V3	Recule_V1	Recule_V2	Recule_V3
DROITE	Droite_V3	Droite_V2	Droite_V3	Droite_V3	UNMOVE	Droite_V1	Droite_V2
GAUCHE	Gauche_V3	UNMOVE	Gauche_V1	Gauche_V2	Gauche_V2	Gauche_V3	Gauche_V3

Figure 8 - Transition États Robots (2)

Cette solution est très efficace bien que le nombre de lignes de code soit élevé, grâce à la lisibilité et à la flexibilité apportées. Chaque transition d'état est visible, et donc le code est lu facilement. Il est facile d'introduire de nouvelles commandes ou de modifier le comportement des commandes existantes car tout changement peut être effectué par ajout ou suppression d’un bloc du code.

Nous avions d’abord envisagé une autre solution avant la modélisation par une machine d’état. L'autre solution aurait été d'utiliser un tableau de Karnaugh pour les transitions d'état basées sur les requêtes. Bien que cette méthode soit très compacte en termes de mémoire, elle réduit la lisibilité et ajoute une complexité temporelle supplémentaire à la fonction. Tout changement dans les états ou les requêtes signifierait une régénération complète des équations, ce qui est, en fait, la capacité même dont nous voulons priver le système de manière à le rendre un peu plus rigide. La structure en `switch-case` imbriqué existante, bien que plus longue, fournit une solution plus claire et extensible pour contrôler les requêtes du robot.

ASSERVISSEMENT DU ROBOT

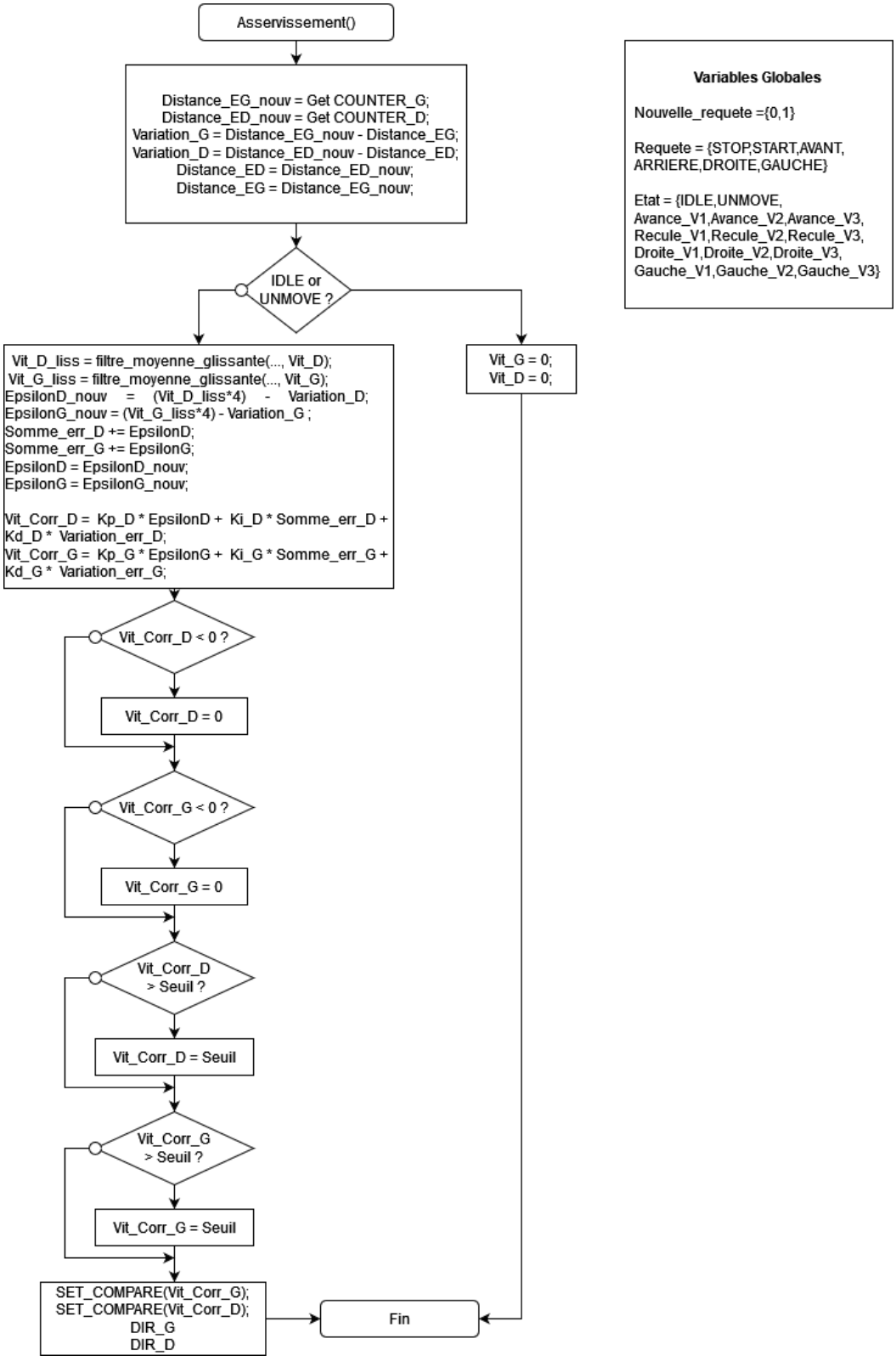

Figure 9 - Algorigramme de l'Asservissement

La fonction ‘asservissement()’ sert à contrôler les vitesses des moteurs selon un contrôle PI (Proportionnel-Intégral). Le module acquiert les tops des encodeurs, calcule l'erreur et règle les vitesses pour obtenir la vitesse corrigée.

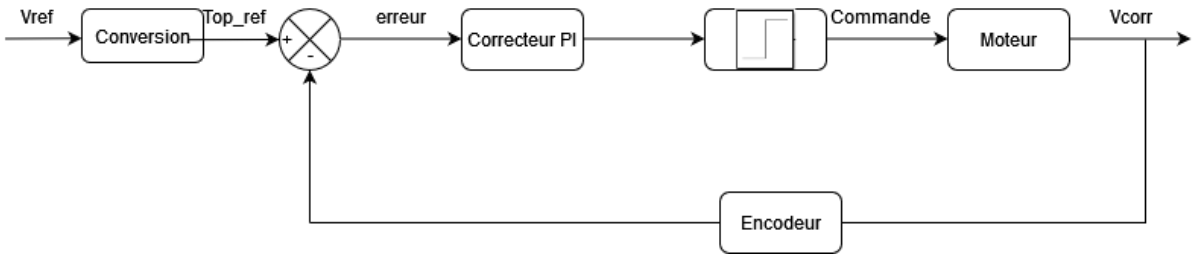

Figure 10 - Schéma Bloc Asservissement en Vitesse

Le schéma bloc d’asservissement fournit une vue d’ensemble du système de contrôle, illustrant comment les différents composants interagissent pour maintenir le robot sur la trajectoire souhaitée.

$$\begin{aligned} \varepsilon(t) &= \text{Consigne} - \text{top}(t) + \text{top}(t-T) \\ \text{sommes\_erreurs}(t) &= \text{sommes\_erreurs}(t-T) + \varepsilon(t) \\ \text{variation\_erreurs}(t) &= \varepsilon(t) - \varepsilon(t-T) \\ \text{Commande}(t) &= K_p \cdot \varepsilon(t) + K_i \cdot \text{sommes\_erreurs}(t) \end{aligned}$$

Figure 11 - Équations Asservissement

À partir du schéma bloc, nous obtenons différentes équations qui permettent de décrire mathématiquement le comportement du contrôleur PI et sa relation avec les autres composants du système. Le contrôleur PI ajuste la commande des moteurs en fonction de l'erreur, suivant les équations de la figure 11.

Pour mettre en place notre asservissement à partir des équations, la première action réalisée dans la fonction est la lecture des encodeurs de roues droite et gauche. Pour la roue droite, la valeur lue est ‘Distance_ED_nouv’, et pour la roue gauche, c'est ‘Distance_EG_nouv’. Ces valeurs sont des mesures en temps réel des roues. Il calcule alors les écarts de position de la roue gauche (‘Variation_G’) et de la roue droite (‘Variation_D’) en soustrayant les anciennes valeurs des codeurs des valeurs actuelles. On peut alors extraire la distance parcourue par chaque roue depuis la dernière actualisation en utilisant ces écarts.

Ensuite, l’asservissement PI ajuste les vitesses des moteurs pour réduire l'erreur entre la vitesse réelle et la consigne de vitesse. L'erreur courante pour chaque roue est calculée en soustrayant le changement dans la position de la consigne de vitesse de la valeur précédente. Cette erreur est utilisée pour calculer la nouvelle somme d'erreurs pour l'intégrale. Les sommes d'erreurs (‘Somme_err_D’ et ‘Somme_err_G’) sont mises à jour en les accumulant avec l'erreur courante. Les variations d'erreurs sont mises à jour en soustrayant la variation d'erreur précédente. Les vitesses

corrigées des moteurs gauche (‘Vit_Corr_G’) et droit (‘Vit_Corr_D’) sont les sorties du PI appliqué aux coefficients proportionnels et intégratifs de gauche et droit. Les vitesses corrigées sont ensuite saturées à des valeurs maximales afin d'éviter le dépassement. Les nouvelles valeurs de consigne de vitesse sont données aux moteurs pour changer les rapports CCR des moteurs.
Si le robot est dans un état ‘IDLE’ ou ‘UNMOVE’, les vitesses des moteurs sont fixées à 0.

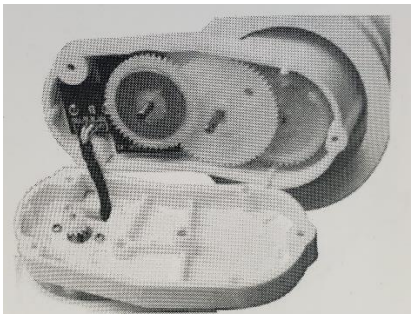

Figure 12 - Motoréducteur du Robot

La première étape pour mettre en place notre asservissement est de configurer nos moteurs pour les rendre fonctionnels. Les moteurs utilisés par notre robot sont 2 motoréducteurs à courant continu. Le principe de fonctionnement des moteurs est le suivant, lorsqu'un courant électrique traverse le bobinage du rotor, un champ magnétique est généré. Ce champ interagit avec le champ magnétique du stator, produisant une force (le couple) qui fait tourner le rotor. Le commutateur, en changeant la direction du courant dans les bobinages du rotor à chaque demi-tour, maintient la rotation du moteur dans une seule direction. Les moteurs à courant continu dont le robot est équipé sont conçus pour fonctionner à une tension nominale de 7.2 V, avec un courant de blocage de 2.5 A. Grâce à un rapport de réduction de 86.8:1, ces moteurs fournissent une vitesse de sortie maximale de 1 km/h soit 27.7 cm/s. Notre contrat stipulait au début une vitesse V3 de 30 cm/s sauf que celle-ci dépasse les capacités de notre moteur et faisait saturer celui-ci. Nous avons donc changé l’objectif et avons imposé une vitesse V3 à 25 cm/s.

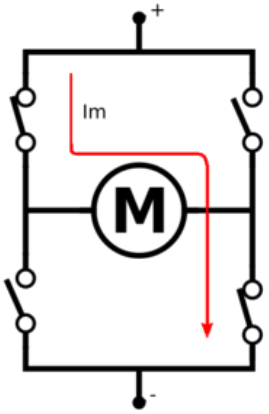

Figure 13 - Pont en H Moteur

Les moteurs que nous utilisions sont également associés à 2 ponts en H. Un pont en H est un circuit électronique utilisé pour contrôler la direction et la vitesse des moteurs à courant continu (DC). Il permet de changer la direction du courant à travers le moteur, ce qui modifie la direction de rotation du moteur.

Les motoréducteurs du robot sont également associés à 2 encodeurs à effet Hall qui nous permette de connaître la distance parcourue par le robot et le sens de rotation de celui-ci. Les encodeurs reposent l'effet Hall qui stipule que lorsque des électrons se déplacent à travers un matériau conducteur ou semi-conducteur soumis à un champ magnétique perpendiculaire, une force (force de Lorentz) dévie ces électrons, créant une différence de potentiel transverse. Cette différence de potentiel est appelée tension Hall.

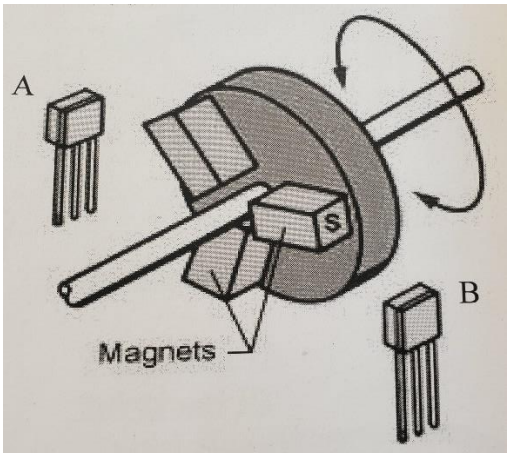

Figure 14 - Encodeur à effet Hall

Un encodeur à effet Hall typique comprend des capteurs à effet Hall et un disque ou une roue magnétique. La roue magnétique est fixée à l'arbre du moteur ou à l'objet en rotation. À mesure que la roue magnétique tourne, les capteurs à effet Hall détectent les variations du champ magnétique. Ces variations correspondent aux pôles nord et sud des aimants sur la roue. Chaque passage d'un pôle magnétique devant un capteur génère un signal électrique (tension Hall). Les signaux électriques produits par les capteurs à effet Hall sont ensuite traités par des circuits électroniques pour produire des impulsions numériques. Ces impulsions peuvent être comptées pour déterminer la position angulaire ou la vitesse de rotation.

Après avoir configuré les moteurs et les encodeurs du robot, l'ensemble des calculs peuvent être effectués mis à part la réponse corrigée du système. Pour calculer les vitesses corrigées par l'asservissement, il faut calculer les différents gains K_i *et* K_p. Les gains dans un asservissement PI sont utilisés pour contrôler la rapidité et la précision de la réponse du système à une consigne de vitesse. Ils permettent de

minimiser l'erreur et de garantir que le robot atteint la vitesse désirée de manière stable et efficace.

Le gain proportionnel K_p augmente la réactivité du système en ajustant la réponse proportionnellement à l'erreur actuelle, ce qui permet une réponse plus rapide. Cependant, un K_p trop élevé peut provoquer des oscillations et rendre le système instable. En parallèle, le gain intégral K_i accumule l'erreur dans le temps, ce qui permet d'éliminer l'erreur stationnaire et d'assurer une précision à long terme. Toutefois, un K_i trop élevé peut également causer des oscillations et de l'instabilité.

Pour déterminer la valeur de nos différents gains, plusieurs méthodes s'offraient à nous. Nous avons choisi de caractériser nos gains à l'aide de la méthode de Ziegler-Nichols. La méthode de Ziegler-Nichols est une approche empirique qui permet de régler précisément un asservissement PI. Il existe deux principales méthodes de Ziegler-Nichols : la méthode de réponse en fréquence (boucle fermée) et la méthode de la réponse transitoire (boucle ouverte).

Dans notre cas, nous avons opté pour la solution en boucle fermée qui nous permet de déterminer les caractéristiques de notre asservissement sans modifier notre fonction. Voici comment procéder pour trouver les différents gains :

- Configurer le système en boucle fermée avec un gain proportionnel uniquement.

- Augmenter progressivement le gain K_p jusqu'à ce que le système atteigne une oscillation stable et soutenue. Le gain critique à ce point est noté K_u (Ultimate Gain) et la période des oscillations est notée T_u (Ultimate Period).

- Utiliser les valeurs K_u et T_u pour calculer les paramètres du contrôleur PI selon les formules suivantes :

    - $K_p = 0.45 \times K_u$
    - $T_i = \frac{T_u}{1.2}$
    - Le terme intégral K_i est $\frac{K_p}{T_u}$

RÉALISATION ET TESTS

CALCULS MATHÉMATIQUES

Nous allons commencer par calculer la valeur seuil de notre ADC. Nous savons que la valeur maximum au niveau de Vbatt lorsque la batterie est chargée est de 5.2 V. Nous fixons le seuil pour allumer la LED à 3 V. Notre ADC est configurée sur 8 bits soit 256 valeurs possibles. Il pourra prendre une valeur entre 0 et 255 selon la tension qu'il va mesurer. Pour trouver le seuil sur cette plage, nous effectuons le calcul suivant :

$$Seuil = \frac{Tension\_Seuil}{Tension\_Max} * 2^8$$

$$Seuil = \frac{3}{5.2} * 256 = 148$$

Ainsi, il faut allumer la LED lorsque la valeur de l'ADC passe sous 148.

Nous pouvons également calculer la valeur du quantum pour retrouver la tension à partir de la valeur de l'ADC.

$$quantum = \frac{Tension\_Max}{2^8} = \frac{5.2}{256} = 0.0203125\,V$$

Ainsi, 1 unité de l'ADC vaut 0.0203125 V. Nous pouvons vérifier que nous obtenons bien 3 V quand l'ADC vaut 148 : $148 * 0.0203125 = 3.00625\,V$.

Nous pouvons passer au Timer 2. Pour obtenir une période de 2 ms soit une fréquence de 500 Hz sur le Timer 2, il faut effectuer le calcul suivant :

$$ARR = \frac{Period}{PSC * T_{Clk}}$$

Pour simplifier les calculs, on va chercher l'ARR du Timer 2 pour un prescaler de 1. On a alors :

$$ARR = \frac{0.002}{1 * \frac{1}{80\,000\,000}} = 160\,000$$

CONFIGURATION DES PÉRIPHÉRIQUES ET DES PINS

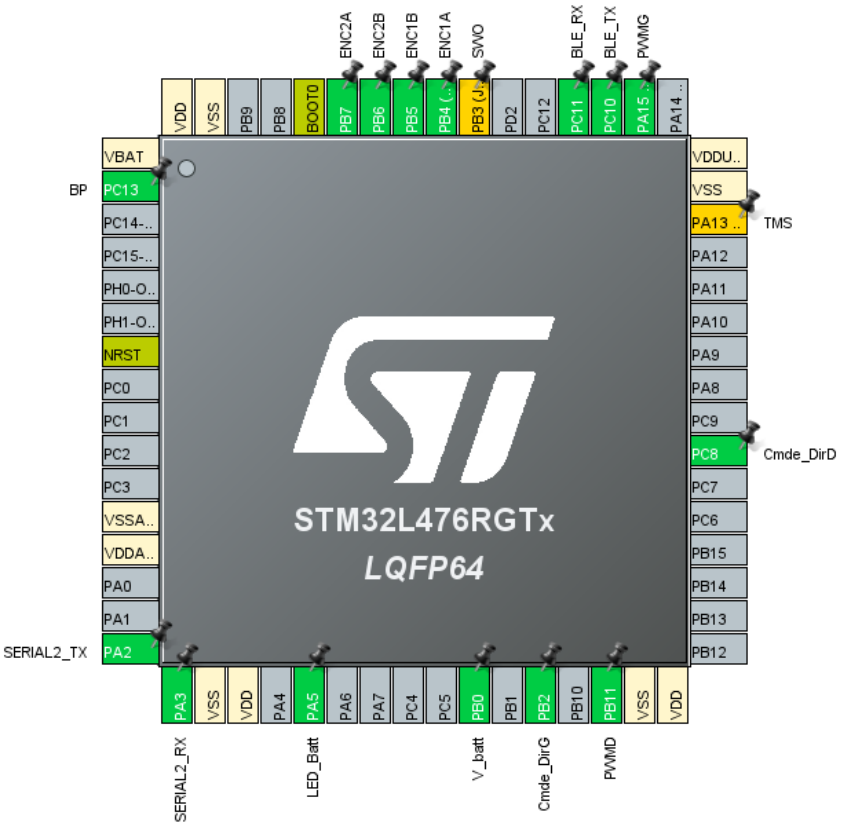

Figure 15 - Configurations Général Pins

- ADC

Nous utilisons le canal 15 pour suivre la tension de notre batterie en mode Single-ended, ce qui signifie que la valeur mesurée est comparée à la masse et est retournée sur 8 bits.

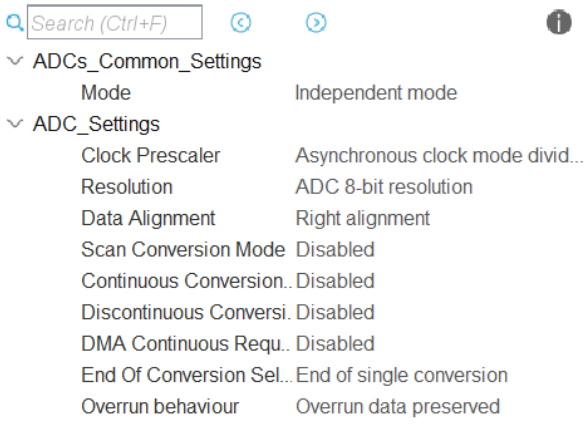

Figure 16 - Configuration de l'ADC (1)

Nous utilisons le canal 15 pour suivre la tension de notre batterie en mode Single-ended, ce qui signifie que la valeur mesurée est comparée à la masse et est retournée sur 8 bits.

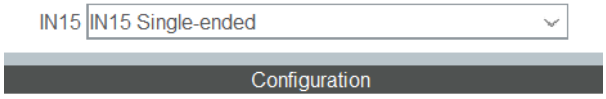

Figure 17 - Configuration de l'ADC1 (2)

L'ADC est configuré sur le pin PB0 en ADC1_IN15 sous le nom de V_Batt et la LED est configurée sur le pin PA5 en GPIO_Output sous le nom LED_Batt.

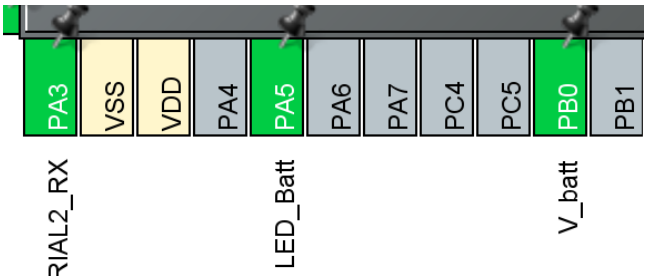

Figure 18 - Configuration des Pins ADC et LED

- TIMER 2

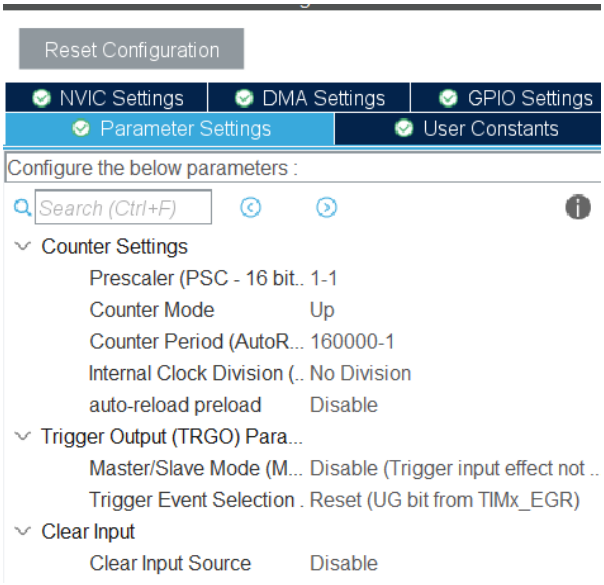

Figure 19 - Configuration du Timer 2 (1)

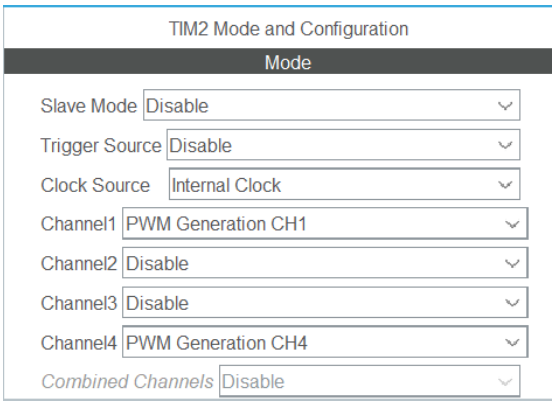

Figure 20 - Configuration du Timer 2 (2)

Nous avons configuré le timer 2 pour générer des signaux PWM sur le channel 1 pour le moteur de gauche et sur le channel 4 pour le moteur de droite. On génère des signaux PWM à une fréquence de 500 Hz. Pour cela, on met notre prescaler à 1 et notre ARR à 160 000 pour obtenir période de 2ms soit une fréquence de 500 Hz.

- BOUTON POUSSOIR

Le bouton poussoir est configuré sur le pin PC13 en GPIO_EXTI13 sous le nom BP.

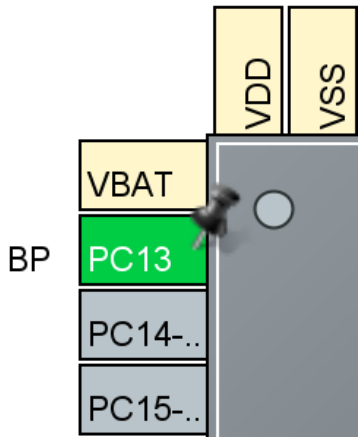

Figure 21 - Configuration Bouton Poussoir

- USART3

L'USART3 est configuré en mode asynchrone à 9600 bauds ce qui permet une communication série où les données sont transmises sans une horloge partagée, à une vitesse de 9600 bits par seconde.

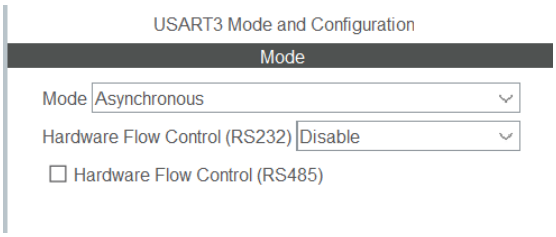

Figure 22 - Configuration de l'USART3 (1)

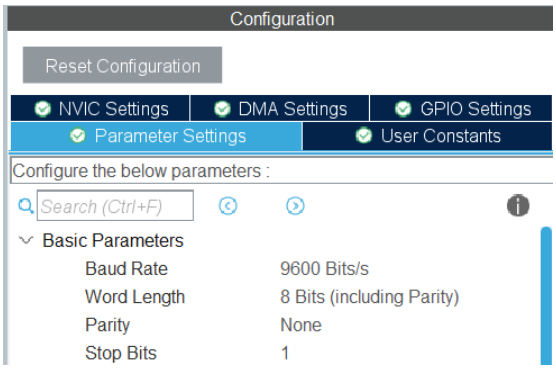

Figure 23 - Configuration de l'USART3 (2)

L'USART3 est configuré en réception sur le pin PC11 et en transmission sur le pin PC10. Dans notre projet seul la réception est utile mais la transmission s'active automatiquement lors de la configuration.

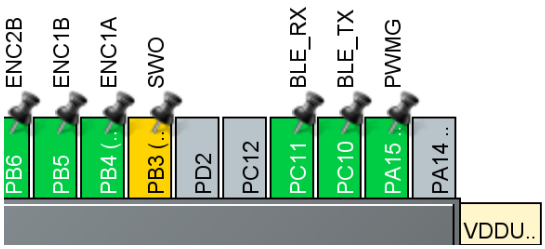

Figure 24 - Configuration Pins USART3

- MOTEURS

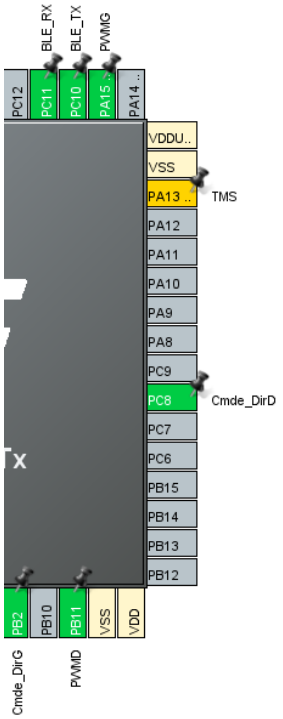

Figure 25 - Configuration Pins Moteurs

Les moteurs sont configurés sur les pins PA15 et PB11 en TIM2_CH1 et TIM2_CH4 pour recevoir les signaux PWM. Les commandes de direction des moteurs sont configurées quant à eux sur les pins PC8 et PB02 en GPIO_Output.

- ENCODEURS

Pour faire fonctionner les encodeurs, nous configurons les Timers 3 et 4 pour recevoir les impulsions de ces derniers. On modifie le mode combined channel des Timers en Encoder Mode et l'encoder mode dans les paramètres en Encoder Mode TI1 et TI2. On laisse l'ARR et le Prescaler par défaut.

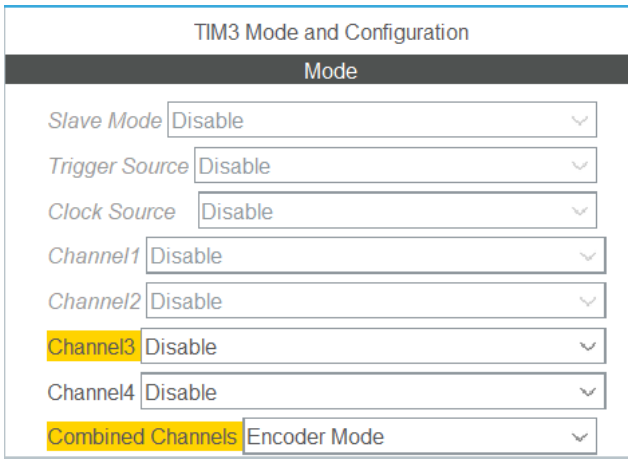

Figure 26 - Configuration du Timer 3 et 4 (1)

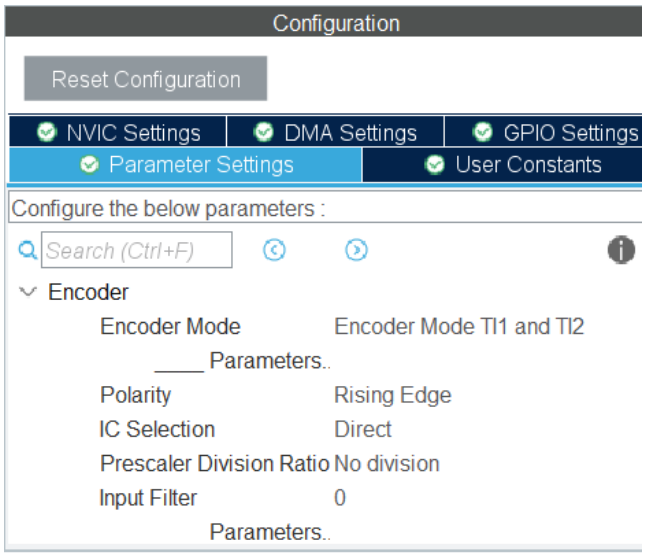

Figure 27 - Configuration Timer 3 et 4 (2)

L'encodeur du moteur gauche est configuré sur le Timer 4 channel 1 et 2 sur les pins PB7 et PB6. L'encodeur du moteur droit est configuré lui sur le Timer 3 channel 1 et 2 sur les pins PB5 et PB4.

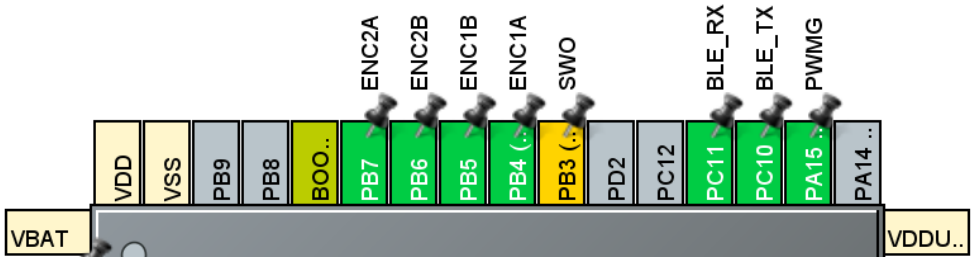

Figure 28 - Configuration Pins Encodeurs

TESTS DES PÉRIPHÉRIQUES ET TIMERS

Avant de lancer et vérifier notre code, il est nécessaire de vérifier la sortie des différents périphériques pour s'assurer qu'ils fonctionnent correctement. La première vérification que nous effectuons est la mesure du timing de Temps_Asser qui est la variable qui permet de lancer l'asservissement toutes les 200 ms. On peut voir sur la figure 25 que la valeur est incrémentée chaque 2 ms. Il faut théoriquement l'incrémenter 100 fois pour avoir 200ms qui sont passés.

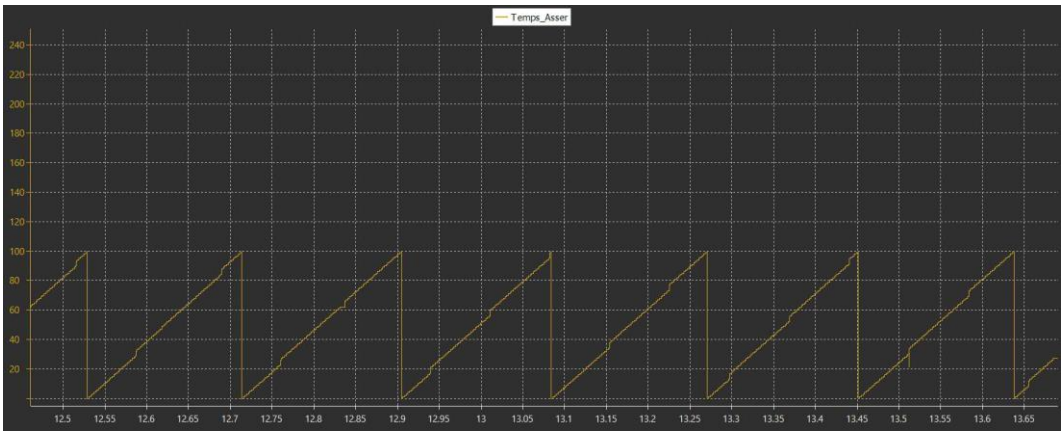

Figure 29 - Évolution de la variable Temps_Asser

On voit bien ici que la variable est bien incrémentée jusqu'à 100, toutefois la période de passage de 0 à 100 prend presque 200 ms avec une très petite erreur de 10 ms. Ce décalage de 10 ms ne représente qu'une erreur de 5 %, on peut la considérer négligeable.

Par la suite, nous allons passer à la vérification des signaux PWM générés par le Timer 2. La vérification de ces signaux, en termes de fréquence et de rapport cyclique, est nécessaire pour s'assurer qu'ils sont conformes aux exigences du cahier des charges et aux consignes. Pour ce faire, nous avons envoyé nos signaux aux moteurs et, à l'aide d'un oscilloscope, nous avons vérifié leurs différentes caractéristiques.

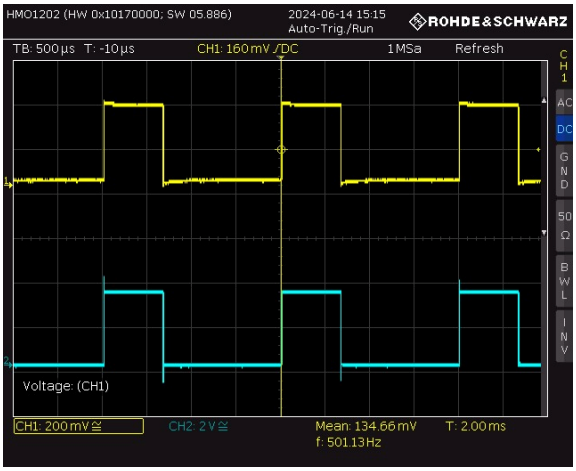

Figure 30 - Chronogramme PWM V1

Nous avons dans un premier temps envoyé au moteur une consigne de vitesse V1. La vitesse maximale des moteurs est de 27.7 cm/s et la consigne V1 est de 10 cm/s. On obtient un rapport de $\frac{10}{27.7} * 100 = 36.1\,\%$. On obtient donc théoriquement un rapport cyclique de 36.1%, on peut alors vérifier expérimentalement cette valeur. Le rapport cyclique est le quotient de la période du signal lorsqu'il est haut sur la période totale. La période totale du signal est de 2 ms et le signal reste haut pendant 0.75 ms soit un rapport cyclique expérimental de $\frac{0.75}{2} * 100 = 37.5\,\%$. Les valeurs expérimentales et théoriques du rapport cyclique sont très proches, on peut

considérer le rapport cyclique comme étant correcte. De plus, la période attendue du signal est de 500 Hz, on voit directement sur l'oscilloscope qu'on obtient une valeur de 501.13 Hz soit également une valeur très proche de la valeur théorique. On peut également valider la valeur de la fréquence du signal.

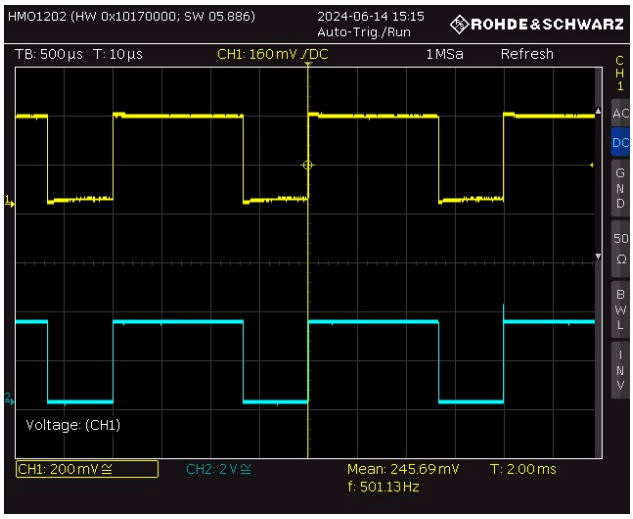

Figure 31 - Chronogramme PWM V2

De même pour le signal PWM de la consigne V2, on obtient un rapport cyclique théorique de $\frac{20}{27.7} * 100 = 72.2\,\%$. Expérimentalement, le signal reste haut pendant 1.375 ms. Le rapport cyclique expérimental est donc de : $\frac{1.375}{2} * 100 = 68.75\,\%$. La valeur expérimentale est ici encore correcte on peut donc valider le rapport cyclique. Enfin, la fréquence expérimentale est également acceptable, elle est de 501.13 Hz contre 500 Hz attendus, on peut valider la contrainte sur la fréquence.

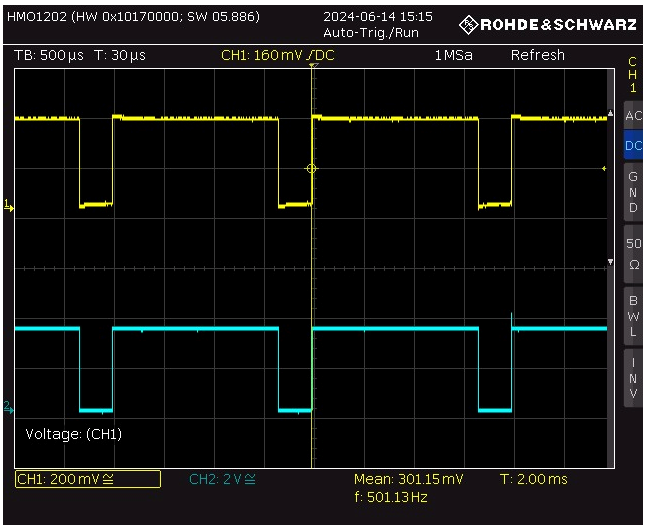

Figure 32 - Chronogramme PWM V3

Pour finir, pour le signal PWM de la consigne V3, on obtient un rapport cyclique théorique de $\frac{25}{27.7} * 100 = 90.2\,\%$. Expérimentalement, le signal reste haut pendant 1.75 ms. Le rapport cyclique expérimental est donc de : $\frac{1.75}{2} * 100 = 87.5\,\%$. La valeur expérimentale est correcte malgré une erreur de 3%. On peut donc valider le rapport cyclique. Enfin, comme pour les autres chronogrammes, la fréquence expérimentale est acceptable, elle est de 501.13 Hz contre 500 Hz attendus, on peut donc valider la contrainte sur la fréquence.

Enfin, nous allons vérifier que les encodeurs fonctionnement théoriquement. D'après les informations sur les encodeurs, nous avons théoriquement 333.33 tops par tour de roue.

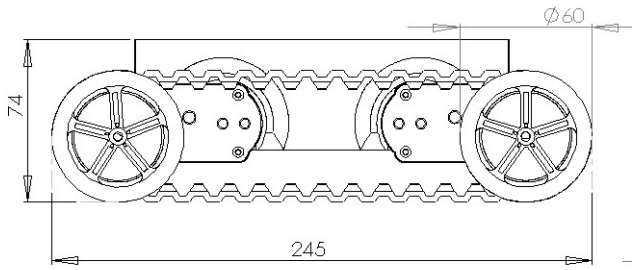

Figure 33 - Dimension du Robot

Notre roue faisant 6 cm de diamètre soit 3 cm de rayon, le périmètre de notre roue est donc :

$$2 * \pi * R = 6 * \pi = 18.85\ cm$$

Nous avons donc 333.33 tops qui sont générés au niveau des encodeurs chaque 18.85 cm. Nous obtenons donc 17.67 tops par cm. Il nous faut donc vérifier cette valeur expérimentalement que nous obtenons un nombre de tops similaire par centimètres.

Pour vérifier ces valeurs expérimentalement, nous allons déplacer le robot sur une distance connue puis relever le nombre de tops et en déduire le nombre de tops par centimètres. Nous commençons tout d'abord par vérifier que le nombre de tops est bien à 0 au début du programme comme nous pouvons le voir sur la figure 32.

Expression	Type	Value
Distance_EG_nouv	volatile float	0
Distance_ED_nouv	volatile float	0

Figure 34 - Initialisation des Encodeurs

Nous avons ensuite fait le choix de déplacer le robot sur 9 cm et nous récupérons la valeur à l'aide de l'outil 'live expressions'.

Distance_EG_nouv	volatile float	162
Distance_ED_nouv	volatile float	165

Figure 35 - Valeur des Encodeurs après 9 cm

On trouve alors 18 tops par cm pour la roue gauche et 18.33 tops par cm pour la roue droite. Ces valeurs sont tout à fait acceptables avec une erreur de 1.92 % pour la roue gauche et 3.79 % pour la roue gauche.

VÉRIFICATION DU START AND STOP

Maintenant que nous avons vérifié nos périphériques matériels, nous pouvons passer à la vérification de nos fonctions. La première fonction de nous allons vérifier est la gestion du start and stop à l'aide du bouton poussoir BP1.

Nous vérifions au début que notre la valeur de notre bouton est bien à 0 et notre requête est définie sur STOP pour que le robot ne puisse se déplacer.

Expression	Type	Value
Bouton_Poussoir	volatile uint8_t	0 '\0'
REQUETE	volatile REQUETE	STOP

Figure 36 - État du Robot au Démarrage

Nous pouvons ensuite appuyer sur le bouton pour vérifier que le robot passe en état START et qu'il est donc capable de se mouvoir.

Expression	Type	Value
Bouton_Poussoir	volatile uint8_t	1 '\001'
REQUETE	volatile REQUETE	START

Figure 37 - État du Robot Après 1 Appui

Enfin, il faut également vérifier que le robot repasse à l'état stop avec un second appui sur le bouton poussoir pour vérifier l'arrêt d'urgence. Nous pouvons voir sur la figure 36 que c'est bien le cas, le robot repasse à l'état STOP.

Expression	Type	Value
Bouton_Poussoir	volatile uint8_t	0 '\0'
REQUETE	volatile REQUETE	STOP

Figure 38 - État du Robot Après 2 Appui

VÉRIFICATION DE LA SURVEILLANCE BATTERIE

Nous avons ensuite testé notre ADC pour vérifier que la mesure de la tension de la batterie se fait correctement. Nous avons effectué une première mesure du robot, nous obtenons une valeur de 41.

Expression	Type	Value
Vbatt	uint8_t	41 ')'

Figure 39 - Mesure Tension Batterie Avant Épreuve

Nous avons ensuite laissé le robot tourné en vitesse pendant 10 minutes pour observer une chute de la tension.

Expression	Type	Value
Vbatt	uint8_t	35 '#'

Figure 40 - Mesure Tension Après Épreuve

On remarque que la tension de la batterie a bien baissé et donc que la mesure se fait correctement. De plus, comme nous pouvons le voir sur la figure 41 la LED s'allume bien, l'ensemble du programme s'exécute correctement.

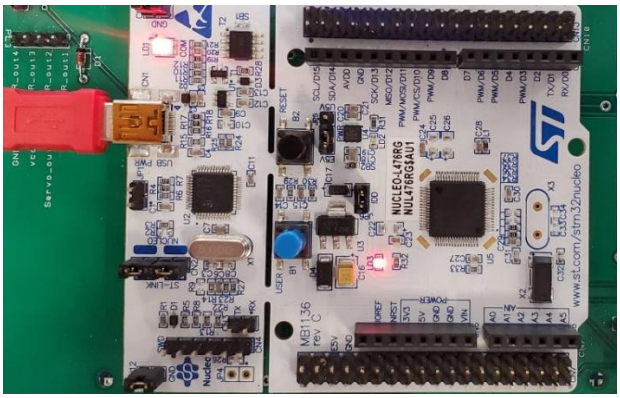

Figure 41 - STM32 Avec LED Allumé

VÉRIFICATION DE LA DETECTION DE REQUÊTE

Par la suite, nous passons à la vérification de la détection des requêtes et du changement des requêtes. Nous avons pour cela par Bluetooth les commandes possibles au robot. On regarde alors s'il reçoit bien les différentes commandes et la mise à jour des requêtes.

Expression	Type	Value
BLUE_RX	volatile uint8_t	70 'F'
REQUETE	volatile REQUETE	AVANT

Figure 42 - Requête AVANT

Expression	Type	Value
BLUE_RX	volatile uint8_t	66 'B'
REQUETE	volatile REQUETE	ARRIERE

Figure 43 - Requête ARRIERE

Expression	Type	Value
BLUE_RX	volatile uint8_t	82 'R'
REQUETE	volatile REQUETE	DROITE

Figure 44 - Requête DROITE

Expression	Type	Value
BLUE_RX	volatile uint8_t	76 'L'
REQUETE	volatile REQUETE	GAUCHE

Figure 45 - Requête GAUCHE

Les différentes commandes sont bien reçues et les requêtes sont bien mises à jour, le programme est fonctionnel.

CONFIGURATION ET RÉSULTAT DE L'ASSERVISSEMENT

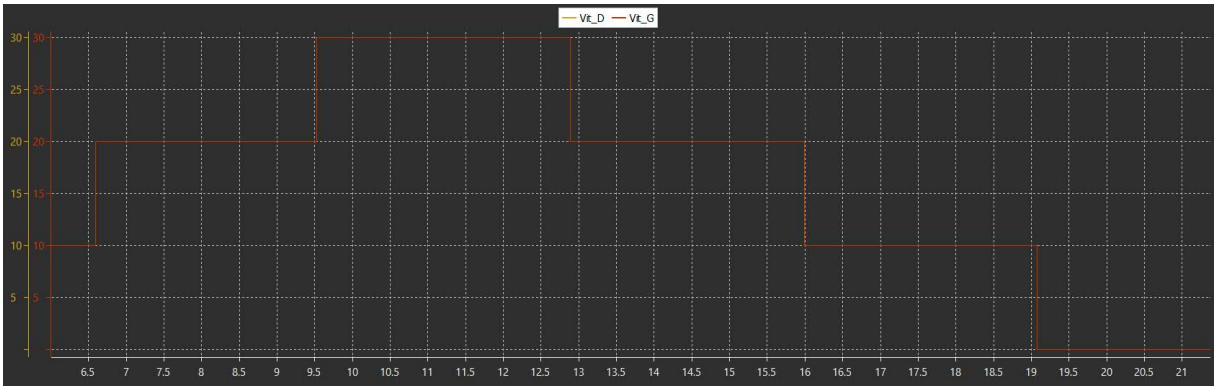

Figure 46 - Échelon d'Entrée du Système V1,V2,V3

Pour calculer les gains de notre système, nous mettons en entrée du système un échelon dont l'amplitude est égale à notre consigne de vitesse. On fixe ensuite Ki à 0 et on augmente Kp jusqu'à obtenir des oscillations constantes et stables. On

procède par dichotomie pour trouver notre gain Ku qui nous permet d'affirmer que nous sommes au début des oscillations constantes et stables.

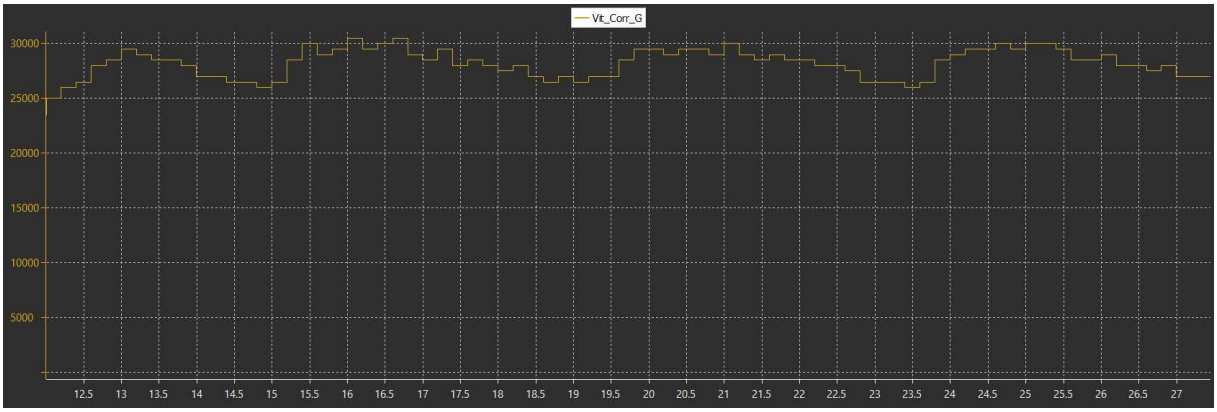

Figure 47 - Réponse du Moteur Gauche Ku = 500

Nous obtenons par dichotomie une valeur Ku = 500 qui permet d'obtenir les premières oscillations stables et constantes. On peut alors en déduire un Kp = 0.45*500 = 225. En regardant la courbe, on obtient une période d'oscillation d'environ 3.75 s ce qui nous donne Ti = 3.75/1.2 = 3.125 s. On a alors Ki = Kp/3.125 d'où Ki = 72.

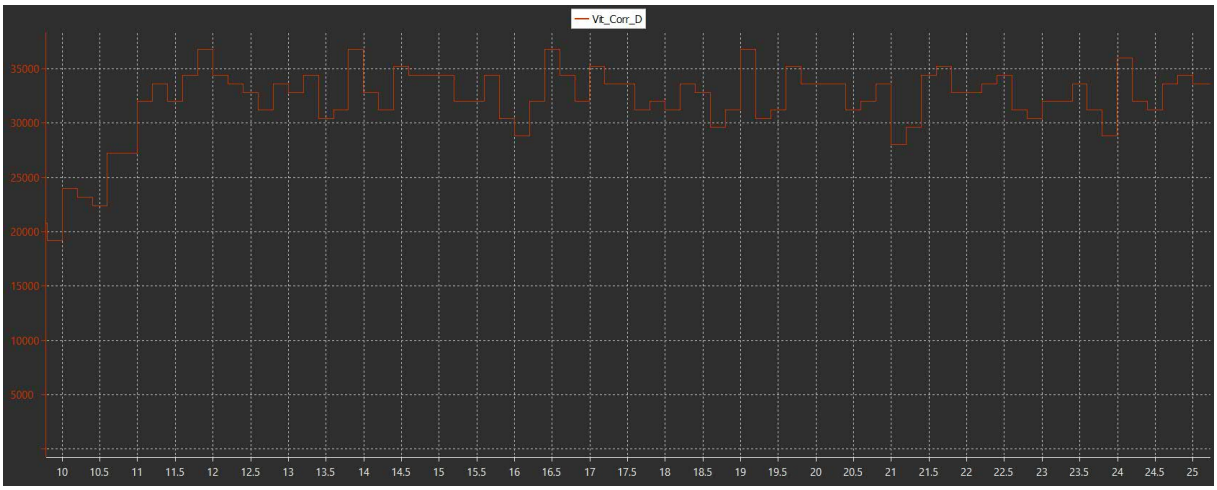

Figure 48 - Réponse Moteur Gauche Ku = 800

De même pour le moteur gauche, nous obtenons par dichotomie une valeur Ku = 800 qui permet d'obtenir les premières oscillations stables et constantes. On peut alors en déduire un Kp = 0.45*800 = 360. Les oscillations du moteur, cependant, ne sont pas très visibles et détaillés mais nous voyons tout qu'il y a des pics qui se démarquent. En regardant la courbe, on obtient une période d'oscillation d'environ 2.16 s ce qui nous donne Ti = 2.16/1.2 = 1.8 s. On a alors Ki = Kp/3.125 d'où Ki = 200.

Il est important de noter que les 2 moteurs du robot étant différents, il faut calculer les gains spécifiques de chaque moteur et non pas un gain commun aux deux. En effet, si nous mettons le même gain aux deux moteurs leur comportement sera différent et des erreurs risque d'exister.

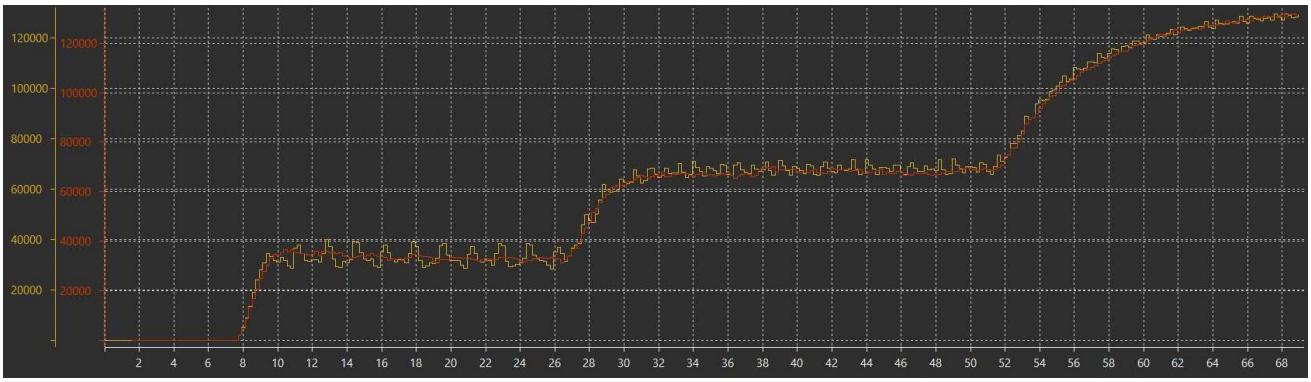

Figure 49 - Réponse des 2 Moteurs avec le Même Gain

Nous pouvons par exemple voir sur la figure 46 que lorsque nous mettons les gains calculer pour le moteur gauche aux deux moteurs les réponses ne sont pas similaires. Ainsi, même si les 2 courbes se suivent, le moteur droit en jaune ici oscille en permanence ce qui montre que les gains ne sont pas adaptés.

Nous pouvons maintenant tracer les réponses du système avec les gains appropriés. Une réponse idéale montrerait une petite période d'oscillations amorties suivie d'une stabilisation à la consigne.

Pour vérifier les performances du système avec les nouveaux paramètres PI, certains points sont à observer :

- La vitesse des roues gauches et droites doit atteindre la consigne rapidement sans oscillations prolongées.
- Les oscillations initiales doivent diminuer progressivement, montrant une convergence vers la vitesse de consigne.
- Le système doit rester stable sans osciller indéfiniment ou devenir instable (déviations croissantes).
- Les erreurs de suivi (différence entre la vitesse de consigne et la vitesse réelle) doivent être minimales.

Figure 50 - Réponses Du Sytème Avec Gains Calculer

Malheureusement, la courbe ci-dessus n'a pas été générée sur le même robot que celui sur lequel nous avons fait le calcul des gains, il est donc normal que la réponse

ne soit pas semblable à celle attendue. Toutefois, nous avons pu relever certaines vitesses de sorties du robot utilisé pour calculer les gains.

VITESSE CONSIGNE	VITESSE RÉEL	ERREUR (EN %)
V1 = 10 CM/S	9.3 CM/S	7 %
V2 = 20 CM/S	19.1 CM/S	4.5 %
V3 = 25 CM/S	23.4 CM/S	6.4 %

Comme nous pouvons le voir sur le tableau ci-dessus, l'asservissement n'est pas parfait mais nous permet tout de même d'atteindre des valeurs assez proches de la consigne. Nous obtenons tout de même des erreurs qui peuvent être conséquentes et qui atteignent jusqu'à 7 %. Notre objectif de précision est de 2 cm/s, celui-ci est vérifié pour toutes les vitesses du robot.

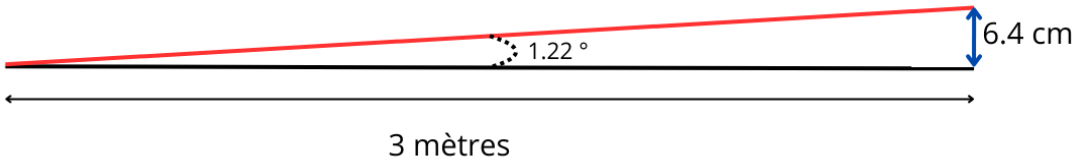

Figure 51 - Décalage Robot en Ligne Droite

De plus, en ligne droite le robot ne connait qu'une déviation moyenne de 6.4 cm sur 3 m et cela sur 5 essais ce qui valide notre cahier des charges car la limite est fixée à 10 cm. Nous pouvons donc que l'asservissement fonctionne et est une réussite.

AMÉLIORATIONS APPORTÉES

• FILTRE MOYENNE GLISSANTE

Un changement brusque de la consigne peut provoquer des à-coups indésirables au début de la réponse du système, ce qui peut affecter la performance de la régulation. Pour atténuer cet effet, nous avons utilisé filtre à moyenne glissante sur la consigne. Ce filtre permet de lisser les variations brusques de la consigne en prenant la moyenne des valeurs récentes de celle-ci sur une fenêtre de temps déterminée. Ainsi, il réduit les transitions soudaines et prévient les à-coups initiaux. En plus de cela, ce filtrage aide à minimiser les décalages en ligne droite, car il assure que les variations rapides et les bruits éventuels dans la consigne sont amortis, permettant au système de suivre une trajectoire plus stable et régulière.

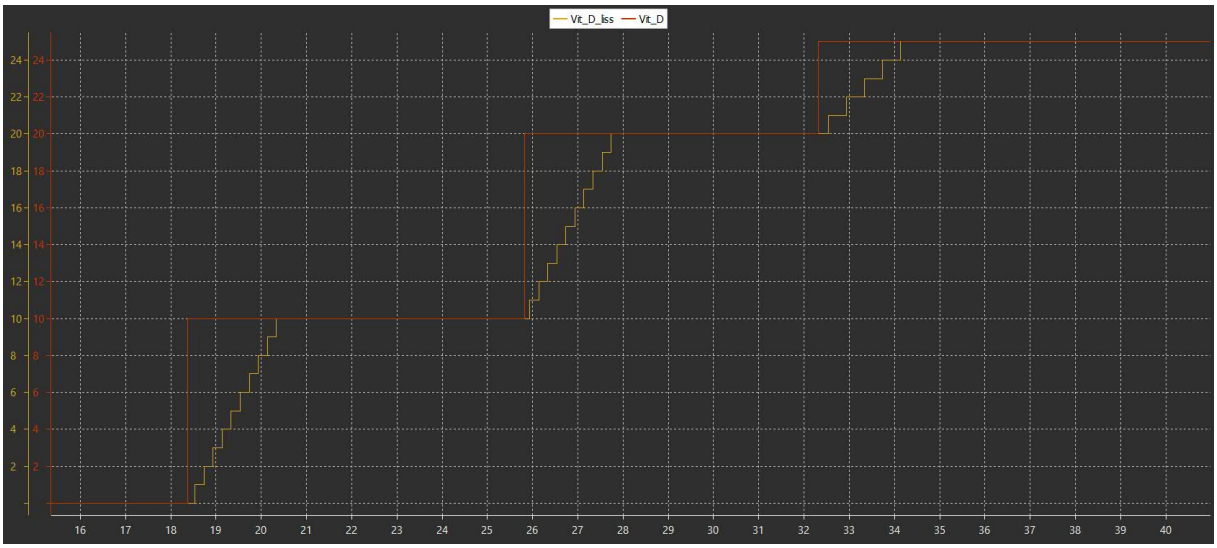

Figure 52 - Vitesse Consigne Filtré Rampe

Voici comment fonctionne ce filtrage :

- **Définir une fenêtre** : Choisir le nombre de valeurs N qui seront utilisées pour calculer la moyenne (la taille de la fenêtre).
- **Calcul de la première moyenne** : Prendre les premières N valeurs de la série de données et calculer leur moyenne.
- **Déplacement de la fenêtre** : Déplacer la fenêtre d'une position vers la droite, en ajoutant la nouvelle valeur et en retirant la plus ancienne.
- **Recalcul de la moyenne** : Calculer la moyenne des valeurs dans la nouvelle fenêtre.
- **Répéter** : Continuer ce processus jusqu'à la fin de la série de données.

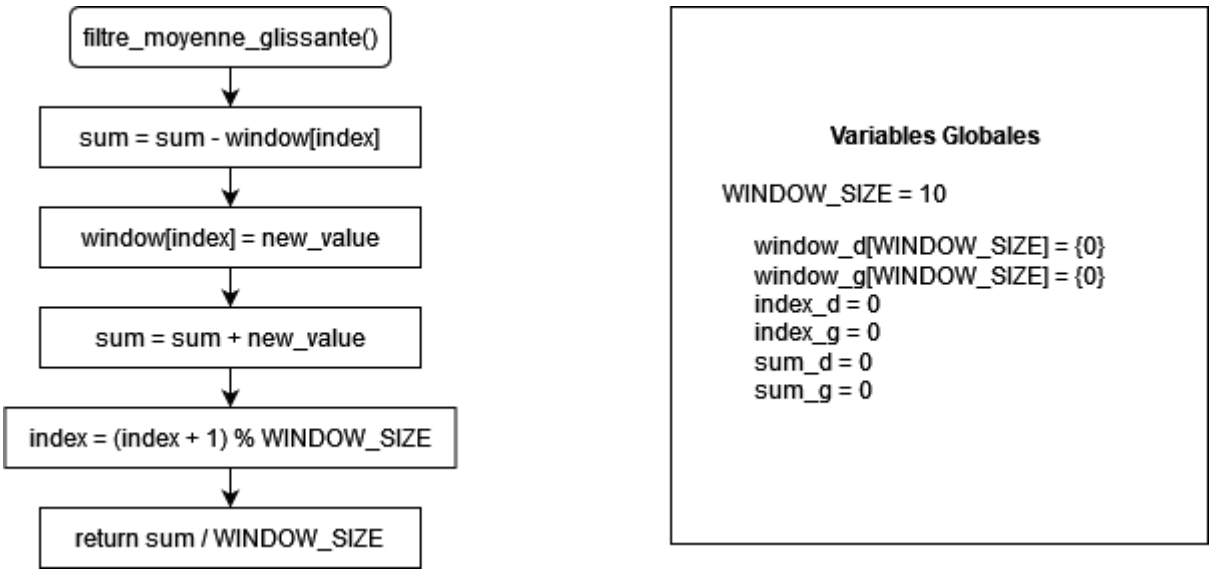

Figure 53 - Algorigramme de la Moyenne Glissante

Cette approche est simple. Chaque itération a une complexité en temps constant, et donc la méthode est très efficace en termes de temps de traitement. De plus, le filtre à moyenne glissante réduit les fluctuations des données, et donc on obtient une version plus lente mais stable du signal d'entrée.

VALIDATIONS DES OBJECTIFS

Nom et unité de la variable	Valeur théorique	Valeur expérimentale	Erreur
Fréquence de PWM (Hz)	500	501.13	1.13 (0.23%)
Vitesse du robot V1 (cm/s)	10	9.3	0.70 (7.00%)
Vitesse du robot V2 (cm/s)	20	19.1	0.90 (4.50%)
Vitesse du robot V3 (cm/s)	25	23.4	1.60 (6.40%)

Figure 54 - Tableau Résultats Objectifs

L'ensemble des objectifs du contrat ont été validés lors des différentes phases de tests de notre projet. En effet :

- La fréquence des signaux PWM est quasiment de 500 Hz.
- Le robot peut se déplacer à plusieurs vitesses : 0 cm/s, 10 cm/s, 20 cm/s et 25 cm/s.
- L'asservissement PI est fonctionnel est régule bien la vitesse des moteurs.
- La précision de la vitesse du robot est également respectée, les variations sont sous les 2 cm/s.
- Enfin, le décalage en ligne droite du robot sur 3 mètres est de seulement 6.4 cm qui est inférieur à la limite des 10 cm.

Nous pouvons donc en conclure que l'ensemble de notre contrat est respecté.

DIFFICULTÉS RENCONTRÉES

De nombreuses difficultés ont été rencontrées tout au long du projet. La mise en place de l'asservissement PI a été une difficulté majeure en raison des oscillations indésirables et de la lecture des courbes qui n'étaient pas précise. Le calcul des gains appropriés pour stabiliser le système sans provoquer de surcompensation ou d'oscillations trop importantes a été délicat. De nombreux ajustements ont été nécessaires pour atteindre une stabilité acceptable.

Une deuxième difficulté a été la mise en place du filtre en Z. Nous avons effectué un filtre par moyenne glissante car la détermination de la fréquence de coupure pour le filtre en Z a été complexe. Une mauvaise fréquence de coupure aurait pu dégrader la performance globale du système. Des essais multiples ont été nécessaires pour ajuster la fréquence mais les résultats n'étaient pas satisfaisants.

Enfin, la qualité des robots utilisés pourraient être amélioré. Les roues mal écartées ont affecté la stabilité et la précision des mouvements. De larges déviations dues au robot lui-même ont été observées mais des modifications mécaniques sont impossibles. On a essayé des ajustements supplémentaires pour assurer que les robots puissent suivre les commandes avec une précision acceptable. Aucune n'a vraiment résolu le problème.

CONCLUSION

En conclusion, plusieurs améliorations peuvent être apportées à notre projet. L'amélioration de l'asservissement est une priorité. On peut mettre en place un asservissement adaptatif. Celui-ci ajustera automatiquement les gains en fonction des conditions de fonctionnement et des erreurs mesurées. Il est également possible d'intégrer des filtres plus avancés. Cela permettra une meilleure estimation de l'état et une réduction des bruits de mesure. L'ajout de capteurs supplémentaires, tels que l'ajout de capteurs sonar ou infrarouges permettra de détecter les obstacles, offrant ainsi une meilleure navigation et évitant les collisions.

Notre robot pourrait avoir plusieurs applications industrielles. L'une des plus probables serait des robots d'exploration de type rover, similaires à ceux utilisés sur Mars, qui permettraient d'accéder à des zones accidentées ou inexplorées. Le robot pourrait également faire de l'exploration minière, pétrolière ou encore participer à des missions de recherche et de sauvetage.

ANNEXE

```
/* Includes ------------------------------------------------------------
---*/
#include "main.h"

/* Private includes ----------------------------------------------------
---*/
/* USER CODE BEGIN Includes */
#include <stdlib.h>
#include <stdio.h>
/* USER CODE END Includes */

/* USER CODE BEGIN PV */
#define AVANCE  GPIO_PIN_SET
#define RECULE  GPIO_PIN_RESET
#define V1 10
#define V2 20
#define V3 25
#define SEUIL_BATT 148 // Seuil des 3V sur 5.2 V
#define Attente 100
#define Kp_D 300
#define Kp_G 300
#define Ki_D 100
#define Ki_G 100
#define V1_top 36
#define WINDOW_SIZE 10

enum REQUETE {START,STOP,AVANT,ARRIERE,DROITE,GAUCHE};
volatile enum REQUETE REQUETE;
enum ETAT
{IDLE,UNMOVE,Avance_V1,Avance_V2,Avance_V3,Recule_V1,Recule_V2,Recule_V3,Dr
oite_V1,Droite_V2,Droite_V3,Gauche_V1,Gauche_V2,Gauche_V3};
volatile enum ETAT Etat;

volatile unsigned char Nouvelle_Requete = 0;
volatile uint8_t Bouton_Poussoir = 0;
volatile uint8_t Temps_Asser = 0;

volatile uint32_t Vit_D = 0;
volatile uint32_t Vit_G = 0;
volatile float EpsilonD = 0;
volatile float EpsilonG = 0;
volatile float Somme_err_D = 0;
volatile float Somme_err_G = 0;
volatile float EpsilonD_nouv = 0;
volatile float EpsilonG_nouv = 0;
volatile uint32_t Vit_Corr_D = 0;
volatile uint32_t Vit_Corr_G = 0;
volatile float Distance_EG_nouv = 0;
volatile float Distance_ED_nouv = 0;
volatile float Distance_EG = 0;
volatile float Distance_ED = 0;
volatile float Variation_D = 0;
volatile float Variation_G = 0;
volatile float Variation_err_D = 0;
volatile float Variation_err_G = 0;
volatile uint32_t Vit_D_liss = 0;
```

```
volatile uint32_t Vit_G_liss = 0;

// Buffers circulaires pour la moyenne glissante
float window_d[WINDOW_SIZE] = {0};
float window_g[WINDOW_SIZE] = {0};
int index_d = 0;
int index_g = 0;
float sum_d = 0;
float sum_g = 0;

uint16_t DIR_G, DIR_D;
uint8_t Vbatt;
/* USER CODE END PV */

/* Private user code ---------------------------------------------------------
---*/
/* USER CODE BEGIN 0 */
volatile uint8_t BLUE_RX = 'C';
/* USER CODE END 0 */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* USER CODE BEGIN 2 */
  HAL_SuspendTick();
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  REQUETE = STOP;
  Nouvelle_Requete = 1;
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_Encoder_Start(&htim3, TIM_CHANNEL_ALL);
  HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);
  HAL_UART_Receive_IT(&huart3, &BLUE_RX, 1);
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */

      // On vérifie la batterie
      HAL_ADC_Start(&hadc1);
      HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
      Vbatt = HAL_ADC_GetValue(&hadc1);
      if(Vbatt < SEUIL_BATT){
          HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin,
GPIO_PIN_RESET);
      }
      else {
          HAL_GPIO_WritePin(LED_Batt_GPIO_Port, LED_Batt_Pin,
GPIO_PIN_SET);
      }

      //On Bouge le robot
      Detecteur_Requete();
      // On démarre l'asservissement
```

```
      if(Temps_Asser >= Attente){
          Temps_Asser = 0;
          asservissement();
      }
  }
  /* USER CODE END 3 */
}


/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    if (Bouton_Poussoir){
        REQUETE = STOP;
    }
    else{
        REQUETE = START;
    }

    Bouton_Poussoir = ~Bouton_Poussoir;
    Nouvelle_Requete = 1;
}

void Detecteur_Requete(void) {

    if (Nouvelle_Requete) {
        Nouvelle_Requete = 0;
        switch (REQUETE) {
            case STOP: {
                Vit_D = Vit_G = 0;
                Etat = IDLE;
                break;
            }

            case START: {
                Etat = UNMOVE;
                break;
            }

            case AVANT: {
                switch (Etat) {
                case IDLE: {
                    Etat = IDLE;
                    break;
                }
                case UNMOVE: {
                    DIR_G = DIR_D = AVANCE;
                    Vit_G = Vit_D = V1;
                    Etat = Avance_V1;
                    break;
                }
                case Avance_V1: {
                    DIR_G = DIR_D = AVANCE;
                    Vit_G = Vit_D = V2;
                    Etat = Avance_V2;
                    break;
                }
                case Avance_V2: {
                    DIR_G = DIR_D = AVANCE;
                    Vit_G = Vit_D = V3;
                    Etat = Avance_V3;
```

```
        break;
    }
    case Avance_V3: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V3;
        Etat = Avance_V3;
        break;
    }
    case Recule_V1: {
        DIR_G = DIR_D = RECULE;
        Vit_G = Vit_D = 0;
        Etat = UNMOVE;
        break;
    }
    case Recule_V2: {
        DIR_G = DIR_D = RECULE;
        Vit_G = Vit_D = V1;
        Etat = Recule_V1;
        break;
    }
    case Recule_V3: {
        DIR_G = DIR_D = RECULE;
        Vit_G = Vit_D = V2;
        Etat = Recule_V2;
        break;
    }
    case Droite_V1: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V1;
        Etat = Avance_V1;
        break;
    }
    case Droite_V2: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V2;
        Etat = Avance_V2;
        break;
    }
    case Droite_V3: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V3;
        Etat = Avance_V3;
        break;
    }
    case Gauche_V1: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V1;
        Etat = Avance_V2;
        break;
    }
    case Gauche_V2: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V2;
        Etat = Avance_V2;
        break;
    }
    case Gauche_V3: {
        DIR_G = DIR_D = AVANCE;
        Vit_G = Vit_D = V3;
        Etat = Avance_V3;
        break;
```

```
    }
    }
break;
}

    case ARRIERE: {
        switch (Etat) {
        case IDLE: {
            Etat = IDLE;
            break;
        }
        case UNMOVE: {
            DIR_G = DIR_D = RECULE;
            Vit_G = Vit_D = V1;
            Etat = Recule_V1;
            break;
        }
        case Avance_V1: {
            DIR_G = DIR_D = AVANCE;
            Vit_G = Vit_D = 0;
            Etat = UNMOVE;
            break;
        }
        case Avance_V2: {
            DIR_G = DIR_D = AVANCE;
            Vit_G = Vit_D = V1;
            Etat = Avance_V1;
            break;
        }
        case Avance_V3: {
            DIR_G = DIR_D = AVANCE;
            Vit_G = Vit_D = V2;
            Etat = Avance_V2;
            break;
        }
        case Recule_V1: {
            DIR_G = DIR_D = RECULE;
            Vit_G = Vit_D = V2;
            Etat = Recule_V2;
            break;
        }
        case Recule_V2: {
            DIR_G = DIR_D = RECULE;
            Vit_G = Vit_D = V3;
            Etat = Recule_V3;
            break;
        }
        case Recule_V3: {
            DIR_G = DIR_D = RECULE;
            Vit_G = Vit_D = V3;
            Etat = Recule_V3;
            break;
        }
        case Droite_V1: {
            DIR_G = DIR_D = RECULE;
            Vit_G = Vit_D = V1;
            Etat = Recule_V1;
            break;
        }
        case Droite_V2: {
            DIR_G = DIR_D = RECULE;
```

```
                Vit_G = Vit_D = V2;
                Etat = Recule_V2;
                break;
            }
            case Droite_V3: {
                DIR_G = DIR_D = RECULE;
                Vit_G = Vit_D = V3;
                Etat = Recule_V3;
                break;
            }
            case Gauche_V1: {
                DIR_G = DIR_D = RECULE;
                Vit_G = Vit_D = V1;
                Etat = Recule_V1;
                break;
            }
            case Gauche_V2: {
                DIR_G = DIR_D = RECULE;
                Vit_G = Vit_D = V2;
                Etat = Recule_V2;
                break;
            }
            case Gauche_V3: {
                DIR_G = DIR_D = RECULE;
                Vit_G = Vit_D = V3;
                Etat = Recule_V3;
                break;
            }
        }
        break;
}
        case DROITE: {
            switch (Etat) {
            case IDLE: {
                Etat = IDLE;
                break;
            }
            case UNMOVE: {
                DIR_G = AVANCE;
                DIR_D = RECULE;
                Vit_G = Vit_D = V1;
                Etat = Droite_V1;
                break;
            }
            case Avance_V1: {
                DIR_G = AVANCE;
                DIR_D = RECULE;
                Vit_G = Vit_D = V1;
                Etat = Droite_V1;
                break;
            }
            case Avance_V2: {
                DIR_G = AVANCE;
                DIR_D = RECULE;
                Vit_G = Vit_D = V2;
                Etat = Droite_V2;
                break;
            }
            case Avance_V3: {
                DIR_G = AVANCE;
                DIR_D = RECULE;
```

```
        Vit_G = Vit_D = V3;
        Etat = Droite_V3;
        break;
    }
    case Recule_V1: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V1;
        Etat = Droite_V1;
        break;
    }
    case Recule_V2: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V2;
        Etat = Droite_V2;
        break;
    }
    case Recule_V3: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V3;
        Etat = Droite_V3;
        break;
    }
    case Droite_V1: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V2;
        Etat = Droite_V2;
        break;
    }
    case Droite_V2: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V3;
        Etat = Droite_V3;
        break;
    }
    case Droite_V3: {
        DIR_G = AVANCE;
        DIR_D = RECULE;
        Vit_G = Vit_D = V3;
        Etat = Droite_V3;
        break;
    }
    case Gauche_V1: {
        DIR_G = DIR_D = RECULE;
        Vit_G = Vit_D = 0;
        Etat = UNMOVE;
        break;
    }
    case Gauche_V2: {
        DIR_G = RECULE;
        DIR_D = AVANCE;
        Vit_G = Vit_D = V1;
        Etat = Gauche_V1;
        break;
    }
    case Gauche_V3: {
        DIR_G = RECULE;
```

```
            DIR_D = AVANCE;
            Vit_G = Vit_D = V2;
            Etat = Gauche_V2;
            break;
        }
        }
    break;
    }
    case GAUCHE: {
        switch (Etat) {
        case IDLE: {
            Etat = IDLE;
            break;
        }
        case UNMOVE: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V1;
            Etat = Gauche_V1;
            break;
        }
        case Avance_V1: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V1;
            Etat = Gauche_V1;
            break;
        }
        case Avance_V2: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V2;
            Etat = Gauche_V2;
            break;
        }
        case Avance_V3: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V3;
            Etat = Gauche_V3;
            break;
        }
        case Recule_V1: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V1;
            Etat = Gauche_V1;
            break;
        }
        case Recule_V2: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V2;
            Etat = Gauche_V2;
            break;
        }
        case Recule_V3: {
            DIR_G = RECULE;
            DIR_D = AVANCE;
            Vit_G = Vit_D = V3;
            Etat = Gauche_V3;
```

```
                    break;
                }
                case Droite_V1: {
                    DIR_G = DIR_D = RECULE;
                    Vit_G = Vit_D = 0;
                    Etat = UNMOVE;
                    break;
                }
                case Droite_V2: {
                    DIR_G = AVANCE;
                    DIR_D = RECULE;
                    Vit_G = Vit_D = V1;
                    Etat = Droite_V1;
                    break;
                }
                case Droite_V3: {
                    DIR_G = AVANCE;
                    DIR_D = RECULE;
                    Vit_G = Vit_D = V2;
                    Etat = Droite_V2;
                    break;
                }
                case Gauche_V1: {
                    DIR_G = RECULE;
                    DIR_D = AVANCE;
                    Vit_G = Vit_D = V2;
                    Etat = Gauche_V2;
                    break;
                }
                case Gauche_V2: {
                    DIR_G = RECULE;
                    DIR_D = AVANCE;
                    Vit_G = Vit_D = V3;
                    Etat = Gauche_V3;
                    break;
                }
                case Gauche_V3: {
                    DIR_G = RECULE;
                    DIR_D = AVANCE;
                    Vit_G = Vit_D = V3;
                    Etat = Gauche_V3;
                    break;
                }
                }
            break;
            }
        }
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
    if (huart->Instance == USART3) {

        switch (BLUE_RX) {
        case 'F': {
            REQUETE = AVANT;
            Nouvelle_Requete = 1;
            break;
        }

        case 'B': {
```

```
            REQUETE = ARRIERE;
            Nouvelle_Requete = 1;
            break;
        }

        case 'L': {
            REQUETE = GAUCHE;
            Nouvelle_Requete = 1;
            break;
        }

        case 'R': {
            REQUETE = DROITE;
            Nouvelle_Requete = 1;
            break;
        }
        default:
            // Empty
        }

        HAL_UART_Receive_IT(&huart3, &BLUE_RX, 1);

    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim) {

    if ( htim->Instance == TIM2) {
        Temps_Asser++;
    }
}

void asservissement(void) {

    Distance_EG_nouv = __HAL_TIM_GET_COUNTER(&htim4);
    Distance_ED_nouv = __HAL_TIM_GET_COUNTER(&htim3);
    Variation_G = abs(Distance_EG_nouv - Distance_EG);
    Variation_D = abs(Distance_ED_nouv - Distance_ED);
    Distance_ED = Distance_ED_nouv;
    Distance_EG = Distance_EG_nouv;

    if (Etat != IDLE && Etat != UNMOVE){
        // Utilisation des valeurs filtrées
        Vit_D_liss = filtre_moyenne_glissante(window_d, &index_d, &sum_d,
Vit_D);
        Vit_G_liss = filtre_moyenne_glissante(window_g, &index_g, &sum_g,
Vit_G);
        EpsilonD_nouv = (Vit_D_liss*4) - Variation_D;
        EpsilonG_nouv = (Vit_G_liss*4) - Variation_G;
        Somme_err_D += EpsilonD_nouv;
        Somme_err_G += EpsilonG_nouv;
        EpsilonD = EpsilonD_nouv;
        EpsilonG = EpsilonG_nouv;
        Vit_Corr_D = (unsigned int)Kp_D * (int)EpsilonD + (unsigned
int)Ki_D * (int)Somme_err_D;
        Vit_Corr_G = (unsigned int)Kp_G * (int)EpsilonG + (unsigned
int)Ki_G * (int)Somme_err_G;

        if (Vit_Corr_D < 0)
            Vit_Corr_D = 0;
        if (Vit_Corr_G < 0)
```

```
            Vit_Corr_G = 0;
        if (Vit_Corr_D > 160000)
            Vit_Corr_D = 160000;
        if (Vit_Corr_G > 160000)
            Vit_Corr_G = 160000;
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, (uint32_t )
Vit_Corr_G);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, (uint32_t )
Vit_Corr_D);
        HAL_GPIO_WritePin(Cmde_DirD_GPIO_Port, Cmde_DirD_Pin, DIR_D);
        HAL_GPIO_WritePin(Cmde_DirG_GPIO_Port, Cmde_DirG_Pin, DIR_G);
    }
    else {
        Vit_Corr_D = 0;
        Vit_Corr_G = 0;
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, (uint32_t )
Vit_Corr_G);
        __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, (uint32_t )
Vit_Corr_D);
        HAL_GPIO_WritePin(Cmde_DirD_GPIO_Port, Cmde_DirD_Pin, DIR_D);
        HAL_GPIO_WritePin(Cmde_DirG_GPIO_Port, Cmde_DirG_Pin, DIR_G);
    }

}
float filtre_moyenne_glissante(float* window, int* index, float* sum, float
new_value) {
    // Retirer l'ancienne valeur de la somme
    *sum -= window[*index];
    // Ajouter la nouvelle valeur à la somme
    window[*index] = new_value;
    *sum += new_value;
    // Avancer l'index
    *index = (*index + 1) % WINDOW_SIZE;
    // Retourner la moyenne
    return *sum / WINDOW_SIZE;
}
/* USER CODE END 4 */
```