ECOLE
POLYTECHNIQUE
DE BRUXELLES

Université Libre de Bruxelles

ELEC-H309 - Projet intégré

Rapport du projet integré

Auteur :
Alexandre Boving
Serge Agyemang

Matricule :
000459850
000494640

Superviseur :
Michel Osée

IRCI 3 - Module Électronique et télécommunication

$20 Mai 2022$

Table des matières

1 Introduction

Le cours *Projet intégré* consiste en la conception d'un système de contrôle d'un robot capable de se mouvoir à la suite d'un ordre sous la forme d'une trame de bits. L'existence même de cette trame se fait sous la forme d'un signal audio à l'aide de la modulation par déplacement de fréquence (modulation *FSK*) avant que celui-ci soit pris en charge et traité par le canal de communication audio du robot. Concrètement, ce système de contrôle doit permettre à notre véhicule automatique de traiter des signaux de commande audio afin d'ordonner à ce dernier des déplacements précis et de façon autonome. Le robot doit alors obéir aux ordres reçus, c'est à dire : "Avance", "Recule", "Tourne à gauche" et "Tourne à droite" accompagnés d'un paramètre indiquant la distance ou l'angle à parcourir.

Pour mener à bien cette tâche, le travail global a été subdivisé en deux parties. La première aborde toute la partie relative à l'acquisition du signal audio ainsi que son traitement numérique qu'on nomme "Communication Audio" tandis que l'autre partie fait référence principalement à la boucle de régulation implémentée au sein du robot gérant son déplacement qu'on qualifie par "Déplacement du véhicule". Le lien entre ces deux parties est finalement effectué pour obtenir le système de contrôle dans son entièreté. De même, des commentaires liés à nos codes seront laissés en fin de rapport pour plus de précision. En ce qui concerne nos codes, ils sont fournis via un lien GitHub et également accompagnés de nos traces de recherches quelles soient justes ou fausses.

2 Analyse du projet

Avant d'aborder la première partie de notre rapport s'agissant de la communication audio, une section d'analyse du projet, qu'on considère primordiale, est nécessaire afin de mieux cerner le projet ainsi que d'établir une méthode de travail.

2.1 Étude du cahier des charges

Le cahier des charges est donné dans le document *Analyse du projet* se situant sur l'UV. Cependant, pour obtenir une meilleur analyse du problème, le cahier des charges est explicité à nouveau ci-dessous.

1. Fonctions
   - Se déplacer précisément et de façon autonome.
   - Recevoir et comprendre des signaux de commande audio.
   - Obéir aux ordres "Avance", "Recule", "tourne à droite", Tourne à gauche" (accompagnés d'un paramètre indiquant la distance ou l'angle à parcourir).
2. Contraintes environnementales
   - Le robot doit être capable de se déplacer sur un terrain solide, plat et horizontal (typiquement une table).
   - Le robot doit être capable de fonctionner dans un environnement calme : à l'intérieur, sans bruit de fond (conversation, musique, ...).
3. Contraintes utilisateurs
   - La tension d'alimentation doit être inférieure ou égale à 24V, pour éviter tout risque d'électrocution.
   - La vitesse de translation du robot doit être telle que celui-ci puisse être facilement "maitrisé" par les étudiants si nécessaire durant les tests de leurs programmes, indépendamment de la manière dont les moteurs sont commandés.
   - Le robot doit être facilement tranportable pour laisser aux étudiants la possibilité de l'emporter chez eux.

On observe à partir de la rubrique *Fonctions* du cahier des charges qu'on peut directement scinder le projet en deux parties comme il est indiqué en introduction. Le fait de recevoir et de comprendre

les signaux de commande peut-être traité indépendamment de ses agissements physiques suite aux ordres reçus. Dans la rubrique *Contraintes environnementales*, on impose les hypothèses de terrain à travers lesquelles le robot évolue. L'environnement dit "calme" donne une indication sur les éventuel(s) test(s) à effectuer pour respecter cette contrainte. Une expérimentation qu'on réalise est de régler la sensibilité de détection des signaux de commande audio du robot pour qu'il puisse détecter la trame de bits dans des circonstances bruyantes avec musique de fond. Par conséquent, un environnement calme ne devrait pas poser de problèmes particuliers. La dernière catégorie de contraintes, *Contraintes utilisateurs*, est respectée dès le début du projet car la tension d'alimentation (7V à vérifier), la vitesse de translation (0.5 m/s) et la géométrie du robot (...) sont directement fournis.

2.2 Les différents modules fonctionnels

Au sein même des deux grands axes de ce rapport, se trouvent plusieurs modules qu'on peut isoler individuellement pour vérifier leur validité. En l'occurrence, cette sous-section sert à mettre en place une méthodologie de travail existante parmi tant d'autres. L'équipe d'ingénieurs chargé de ce projet à décidé de travailler suivant la méthode **Agile**. En effet, en se fixant des objectifs à court terme, il est plus facile de fragmenté le projet en plusieurs sous-catégories dont on se doit d'atteindre progressivement. Ces modules qu'on essaye de clôturer et de valider à la fin d'une séance voir deux sont les objectifs de ce qu'on appelle un "sprint". Au total, plusieurs "sprints" ont été réalisés pour les modules suivant : **Circuit d'amplification/Filtre de garde**, **Filtres numériques**, **Comparateur/FSK detector**, **Démarrage des moteurs**, **Boucle de régulation** et **communication entre les deux dsPIC.**

2.3 Stratégie de validation

L'étape de validation à l'aide d'expérimentations constitue une avancée majeure dans le projet car elle permet de clôturer et valider un module avant de l'intégrer dans l'ensemble du robot. Tout au long de ce rapport, les modules sont présentés avec le(s) test(s) qui les accompagne. Pour achever une sous-partie ou une partie entière, un test général regroupant les modules fonctionnels adéquats est effectué pour confirmé une sous-étape importante du projet.

3 Communication audio

La communication audio a pour objectif de récupérer le signal modulé et de le numériser afin d'être traité digitalement. A l'issue de ce traitement numérique, on obtient un message binaire interprétable par le robot. Les composants utilisés dans cette partie sont le dsPIC33FJ128MC802 et l'UART qui sert à visualiser à l'aide d'un code Python les résusltats des tests pour cette partie.

3.1 La chaîne d'acquisition audio

L'ADC du dsPIC33FJ128MC802 s'occupe de la conversion analogique/digital dans une plage de conversion allant de $0V$ jusqu'à $3.3V$. Compte tenu de cette plage de conversion, il est impératif d'installer un étage d'amplification entre le micro et l'ADC avec un **offset** (une tension continue). En effet, la tension de sortie du micro servant à recevoir est assez faible et est centré autour de 0V. On aborde ci-dessous la conception et la mise en oeuvre du montage analogique en allant de la trame de commande jusqu'au filtre de garde du montage.

3.1.1 La trame de commande

A travers ce projet, on utilise un mode de modulation de fréquence dans lequel la fréquence du signal modulé varie entre des fréquences prédéterminées. On parle alors de la modulation par déplacement de fréquence (modulation *FSK*). Le signal modulé du projet est un signal variant entre deux fréquences autour d'une valeur centrale et peut être formulé de la manière suivante :

$$\text{signal modulé } = \begin{cases} A_0 \sin\left(2\pi\left(f_p - \Delta_f\right)t\right) \Leftrightarrow \text{ signal modulant } = 0 \\ A_0 \sin\left(2\pi\left(f_p + \Delta_f\right)t\right) \Leftrightarrow \text{ signal modulant } = 1 \end{cases} \quad \text{où} \qquad (1)$$

$f_p = 1kHz$ la fréquence porteuse.
$\Delta f = 100Hz$ le déplacement de fréquence.

A une période de $T_s = 100ms$, le signal modulant présente une trame de bits composée d'un

préambule, de données et d'un postambule. Les données sont composées d'un ordre (2 bits) accompagné de son paramètre (8 bits). Ces deux derniers représentent le message sur 10 bits que l'unité centrale du robot (le dsPIC33FJ128MC802) doit pouvoir décoder à l'aide de son montage analogique suivit du traitement numérique adéquat.

3.1.2 Le montage analogique

Pour recueillir le signal modulé, il est important de dimensionner et de mettre en oeuvre une chaîne d'acquisition audio capable de retracer la forme du signal reçu. Pour analyser ce montage analogique, on propose d'aborder le premier composant du circuit qui est le micro.

Le micro : Imposé, celui-ci est un *"Electret Condenser Microphone"* KECG2740PBJ. A partir de sa datasheet disponible sur l'UV, on valide directement certains critères d'utilisation tels que le *"Standard Operating Voltage"* ***2 V***, l'impédance de sortie R_L = **2,2 kHz** et le courant de polarisation I_{pol} = **0.5 mA**. Vu que le circuit est alimenté par le dsPIC (3,3V), lui-même alimenté par du 5V, on peut tout de suite conclure qu'on se situe dans la gamme d'opération de ce micro.

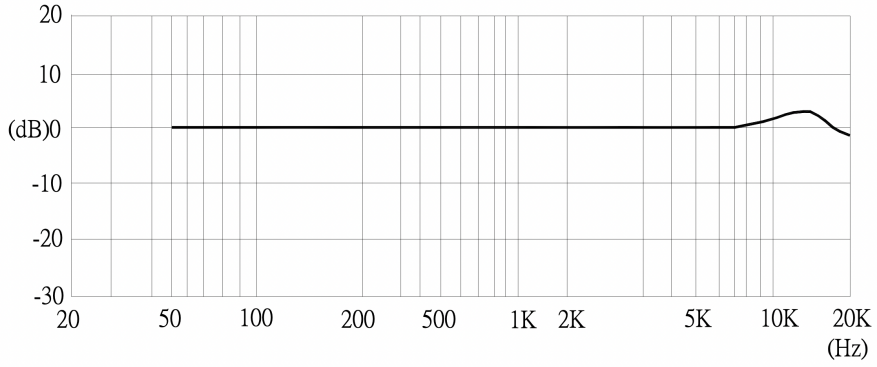

FIGURE 1 – Courbe de réponse en fréquence du micro KECG2740PB

En commençant par analyser sa réponse en fréquence (fig. 1), on s'aperçoit qu'au delà de 7kHz, le module de la fonction de transfert entre la sortie du micro et le signal modulé est supérieur à 1. Cependant, en se limitant aux fréquences du projet c'est à dire $1,1kHz$ pour la fréquence la plus élevée, on conclut que le gain du micro n'est pas à prendre en compte puisqu'il vaut O dB.

De même on suppose la sortie du microphone étant égale à 1mV au maximum e situant dans un environnement calme.

En deuxième, on peut attaquer la représentation du circuit du micro à l'aide de sa représentation à la figure 2. Les microphones à condensateur électret requièrent une alimentation externe fournie par le dsPIC. Son courant de polarisation n'excédant pas les 0.5 mA, on peut à présent calculer la résistance R1 à partir de la fonction suivante :

$$I_{pol} = \frac{V_{cc}}{2 \times (R1 + R2)} \quad \text{où} \tag{2}$$

$V_{cc} = 3.3V$.
$R2 = R_L = 2.2k\Omega$.
$I_{pol} = 0.5mA$.

Par conséquent, R1 doit prendre une valeur supérieur à $1.1k\Omega$. Cette résistance ensemble avec une capacité C1 forme un circuit passe-bas et permet de filtrer les résidus ou autres laissant uniquement la tension continue alimenter le micro.

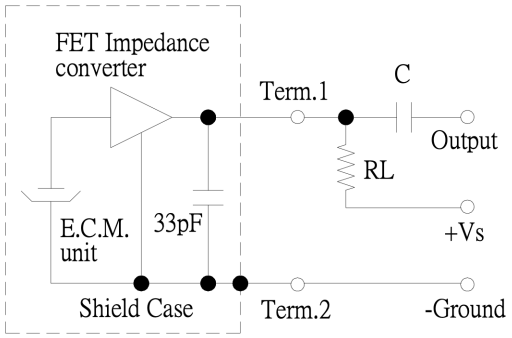

FIGURE 2 – Circuit électrique du micro

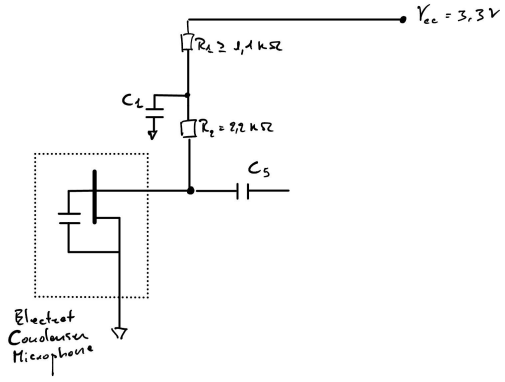

FIGURE 3 – Avec son filtre passe-bas

Pour la conception du condensateur, on fixe une fréquence de coupure $f_{Coup.1} = \frac{1}{2\pi \times R1 \times C1} = 90Hz$. Pour rappel, la résistance R1 est supérieur à 1.1 kΩ. Au labo UA5, on a choisit une résistance R1 de 1.5 kΩ donnant un condensateur C1 = 1.17 $\mu F \approx 1\mu F$. Le condensateur sur la figure 2 correspond au condensateur C5 sur la figure 4 qui permet de filtrer l'offset provenant du circuit et de fournir uniquement la tension alternative en sortie du micro vers le reste du circuit.

L'étage d'amplification : L'amplification est assurée par un Amp-op fournit directement par le laboratoire en UA5. Il s'agit du **MCP6271** qu'on alimente par du 3.3V et 0V en sortie. Du à son alimentation, un offset de 1.65V est introduit atour duquel on peut pleinement profiter de la plage d'amplification. Celui-ci est assuré par un étage avant l'entrée non-inverseuse composé d'un diviseur résistif. Le voici juste ci-dessous :

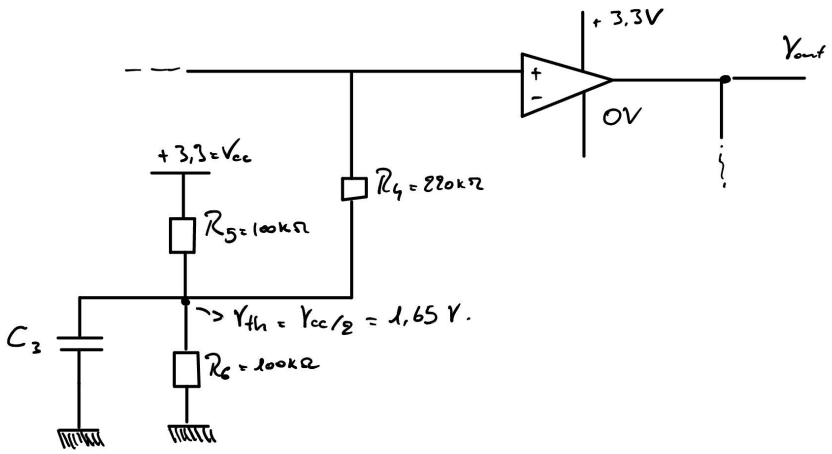

FIGURE 4 – Diviseur résistif

Les valeurs de R5 et R6 doivent être égales, on a choisit celles proposées par la datasheet sur l'UV. La valeur de R4 est 220k Ω pour obtenir une impédance d'entrée assez élevée à l'entrée de l'Amp-Op. C'est pour éviter l'effondrement du signal provenant du micro lorsqu'on travaille en tension alternative. Tandis que C3 est choisit comme filtre passe-bas pour éviter les hautes fréquences provenant du circuit intégré. Celui-ci est déterminé de la même manière que C1 avec une fréquence de coupure $f_{Coup.3} = 90Hz$. Cela nous donne 17nF qu'on approche par un condensateur à 15nF. C5 est alors déterminé en faisant office de passe-haut. Sa fréquence de coupure est assurée à 90Hz et celui-ci vaut approximativement 7.9nF qu'on approche par un condensateur de 10nF au laboratoire UA5.

Le gain de l'amplificateur est calculée en supposant la superposition de celui de la tension continue avec celle alternative. Il est donné par la formule suivante :

$$G = (1 + \frac{R8}{R7}) \times (\frac{R4}{R4 + R2}) \quad \text{où} \tag{3}$$

$R8 \approx R4 + \frac{R5 \times R6}{R4+R6}$

En respectant celui, c'est à dire un Gain de 1650 donné par $G = \frac{V_{out}}{V_{in}} = \frac{1.65V}{1mV}$, on parvient à déterminer R7 qui vaut une valeur de 200Ω. Ce gain calculé est le seul étage d'amplification puisque le gain de l'étage du filtre de garde est un gain unitaire. Par conséquent, l'Amp-Op du premier et seul étage doit assurer toute l'amplification.

Le filtre de garde : Un filtre de garde est nécessaire afin d'éviter le repliement spectrale. Celui-ci est implémenté via un filtre actif, de type Butterworth. Il a été établi via le logiciel "Analog Filter Wizard" et a la forme suivante :

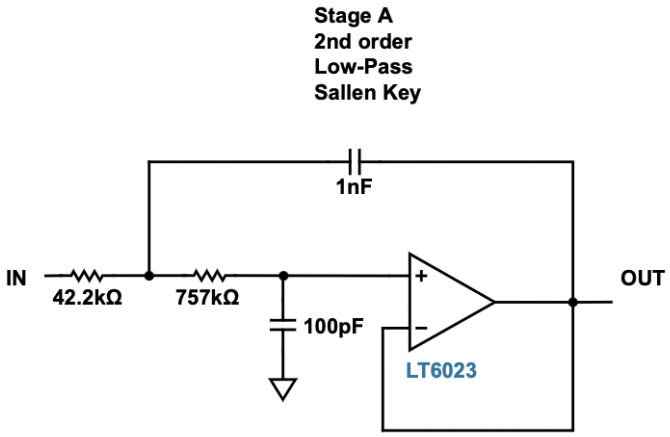

FIGURE 5 – Circuit électronique du filtre de garde

3.1.3 Validation de la chaîne d'acquisition audio

Comme visible sur les figures 11 et 12, les signaux sortant du circuit analogique ont une amplitude crête-à-crête de 3,369V et un offset avoisinant 1,68V ce qui est satisfaisant compte tenu des attentes du cahier de charge.

Voici une représentation totale du circuit électrique auquel on ajoute le filtre de garde et toutes les résistances et capacités calculées précédemment. Les valeurs des composants et des condensateurs sont récapitulées ci-dessous.

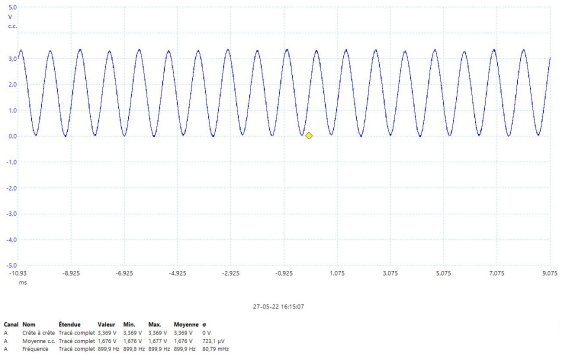

FIGURE 6 – Signal analogique à 900 Hz

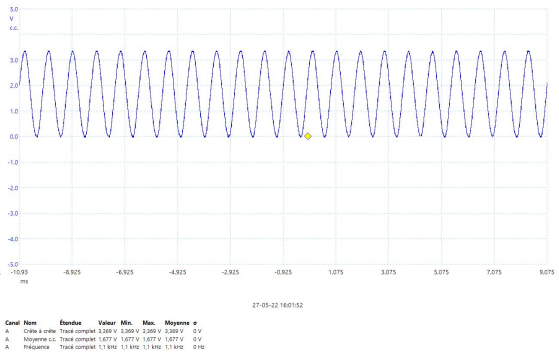

FIGURE 7 – Signal analogique à 1100 Hz

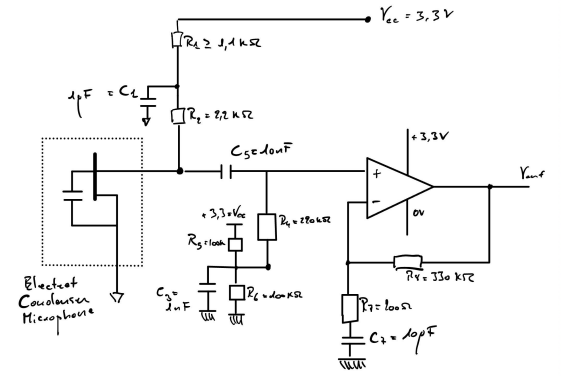

FIGURE 8 – Montage analogique

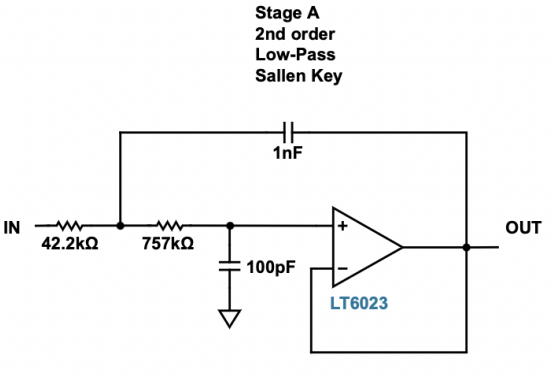

FIGURE 9 – Filtre de garde

Liste de composants électroniques		
Résistances/Capacités	Valeurs théorique	Valeurs pratique
R1	$1.1k\Omega$	$1.5k\Omega$
R2	$2.2k\Omega$	$2.2k\Omega$
R4	$220k\Omega$	$220k\Omega$
R5	$100k\Omega$	$100k\Omega$
R6	$100k\Omega$	$100k\Omega$
R7	$200k\Omega$	$200k\Omega$
R8	$320k\Omega$	$320k\Omega$
C1	$1.17\mu F$	$1\mu F$
C3	$17nF$	$15nF$
C5	$7.9nF$	$10nF$
C7	$8.9\mu F$	$10\mu F$

3.2 Traitement numérique du signal

3.2.1 Les filtres numériques

Le signal numérisé par l'ADC est passé dans 2 filtres passe-bandes dont l'un est centré à la fréquence de 900 Hz et l'autre à la fréquence de 1100 Hz. Si le signal ne comporte pas la fréquence centrale du filtre, ce dernier sera donc relativement fortement atténué. En réalité, les 2 filtres d'ordre 8 ont été chacun subdivisés en 4 filtres d'ordre 2. Les filtres ont été implémentés via l'équation de récurrence suivante :

$$\begin{cases} y_1(n) = \sum_{i=0}^{p} b_i.x(n-i) - \sum_{i=0}^{m} a_i.y_1(n-i) \\ y(n) = G.y_1(n) \end{cases}$$

Les coefficients caractérisant le filtre ont été calculés via le code Python fourni. C'est ainsi que les 2 filtres fonctionnels ont pu être établi et leur sortie peut être apprécié sur la Figure 14. La virgule fixe a du également également être implémentée afin de d'obtenir un temps d'exécution inférieur à la période d'échantillonage de l'ADC.

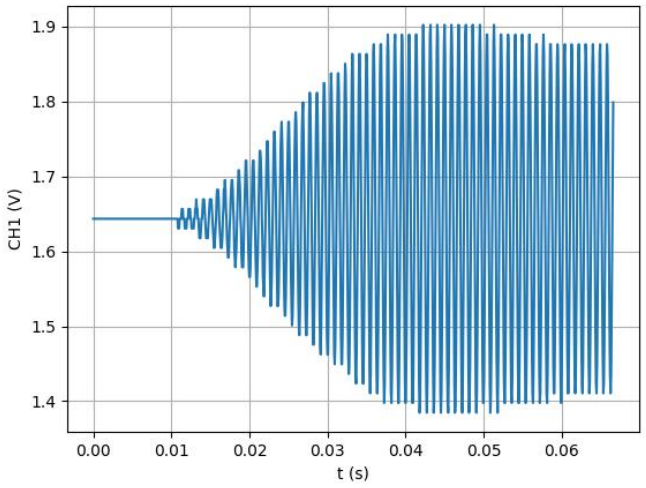

FIGURE 10 – Sortie du filtre pour une fréquence à 1100 Hz

La validation des filtres numériques s'est faite via une transformation de Fourrier pour afficher

son spectre de fréquence. Comme observable sur les Figures suivante, le signal à 1200 Hz est fortement atténué tandis que le signal à 1100 Hz ne l'est pas.

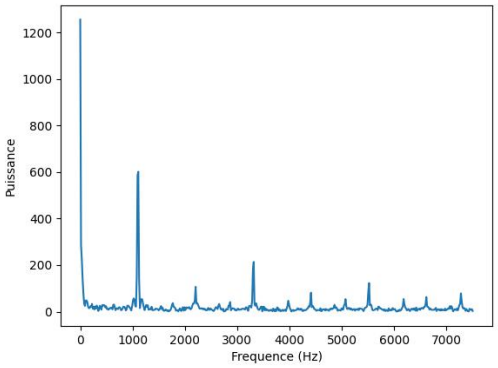

FIGURE 11 – Sortie du filtre pour 1100 Hz.

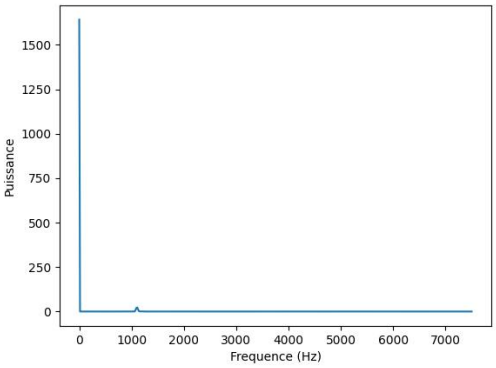

FIGURE 12 – Sortie du filtre pour 1200 Hz.

3.2.2 Le Comparateur et le fskDetector

La prochaine étape venant après les filtres numériques est le comparateur. Ce dernier permet de déterminer si le signal reçu présente une fréquence de type f_0) ou f_1) respectivement pour un bit 0 et un bit 1. Pour cela, on procède par imposer un seuil de comparaison établi par essais-erreurs qu'on impose autour de 0.4-0.5V. La raison principale est que l'amplitude des signaux atténués par nos filtres numériques ne dépassent pas ces valeurs général. Concrètement, le principe de ce comparateur est de comparer l'amplitude des échantillons pour déterminer si une amplitude dépasse le seuil imposée. Un certain nombre de bits étant nécessaires pour déterminer cette comparaison d'amplitude, on le détermine grâce à la période d'échantillonnage qui vaut 15 000Hz. Afin de couvrir l'entièreté d'une période de n échantillons, on établit la formule suivante :

$$n \cdot T_s > T_{900Hz} \Leftrightarrow n \times 6,6667 \cdot 10^{-5} > 1,1111 \cdot 10^{-3} \Leftrightarrow n > 16,66 \tag{4}$$

$$n \cdot T_s > T_{1100Hz} \Leftrightarrow n \times 6,6667 \cdot 10^{-5} > 9,09 \cdot 10^{-4} \Leftrightarrow n > 13,64 \tag{5}$$

Le nombre n = 15 est alors choisi. On procède dernièrement par appeler un détecteur d'enchaî-

nements de fréquences qu'on nomme le *fskDetector*. Ce dernier décode le Start Bit, la trame et pour finir le Stop Bit. A la sortie, la fonction renvoie l'ordre (2bits) ainsi que le paramètre (10bits).

3.2.3 Échange d'information entre les *dsPIC* via l'*UART*

L'UART est un protocole de communication asynchrone permettant à deux dispositifs de communiquer. Il suffit de relier la patte TX (de transmission) d'un des dispositifs à la patte RX (de réception) d'un autre dispositif et vice-versa. Ceci peut notamment s'avérer utile entre le dsPIC utilisé pour la communication et un ordinateur afin de pouvoir notamment valider les sorties des filtres. C'est dans cette optique que le code Python "oscilloscope" a été donc employé. En ce qui concerne la communication entre le dsPIC de la communication et celui de la partie déplacement, on adopte un raisonnement similaire en envoyant l'information via la patte TX de la communication vers la patte RX du déplacement. Ceci est effectué via le code suivant :

```
message = fskDetector(detLow, detHigh);
if (message != 0){
    int16_t message2bits = (message & 0xFF00)>>8;
    int16_t message8bits = message & 0x00FF;
    printf("Message");
    while (U1STAbits.UTXBF){} // On attend que ça soit libre.
    U1TXREG = message2bits;
    while (U1STAbits.UTXBF){} // On attend à nouveau que ça soit libre.
    U1TXREG = message8bits;
}
```

FIGURE 13 – Envoie de l'information TX

```
while (1){ // On va vérifier en premier si l'UART1 a reçu un octet avec lequel on peut travailler. Code reprit de l'OSCILLOSCOPE.
    if(U1STAbits.URXDA){
        type = U1RXREG;
        //printf("type");
        if ((0 <= type) && (type < 4))
        {
            while (!U1STAbits.URXDA) {} // On attend que l'UART soit à nouveau disponible.
            parametre = U1RXREG;
            occupied = 1;
        }
    }
```

FIGURE 14 – Reception de l'information RX

3.2.4 Validation de la chaine d'acquisition audio

La validation a été filmée pour des mesures de facilitées et la vidéo est proposée sur le lien Github ainsi que les codes et leurs commentaires.

4 Déplacement du véhicule

4.1 Description des composants principaux

Les composants nécessaires au déplacement du robot sont les suivants :

— 2 moteurs à courant continu. Ceux-ci sont équipés d'un réducteur de vitesse et d'encodeurs en quadrature. Alimentés en 5V, ces derniers fournissent deux signaux logiques compatibles avec la logique CMOS.

— 1 driver DRI004. Ce driver fera office d'interface entre le dsPIC et les moteurs. En outre, il possède des pattes DIR1 et DIR2 imposant la direction des deux roues ainsi que des pattes PWM1 et PWM2 s'occupant de régler la vitesse de ces dernières.

— 1 dsPIC33FJ128MC802. Ce microcontrôleur possède deux modules "Output Compare" qui permettront de régler la vitesse ainsi que deux " Quadrature Encoder Interface" permettant de récupérer les registres associés aux encodeurs nécessaires à la régulation en boucle fermée.

— 1 batterie NiMh de 7,2V.

4.2 Régulation polaire

La régulation polaire consiste en la combinaison d'une régulation en translation et une régulation en rotation. Celle-ci est nécessaire afin que le robot ne parte pas en arc de cercle lors d'une simple régulation en translation par exemple. En effet, lors d'une translation, la régulation polaire se charge de garder l'erreur angulaire nulle. Le raisonnement est analogue pour la rotation. Le schéma de la régulation polaire est repris sur la Figure 15.
L'erreur, soit la différence entre la consigne et la position mesurée par les encodeurs, entre dans un régulateur proportionnel de gain K_{pr} pour la rotation et K_{pl} pour la translation. La sortie

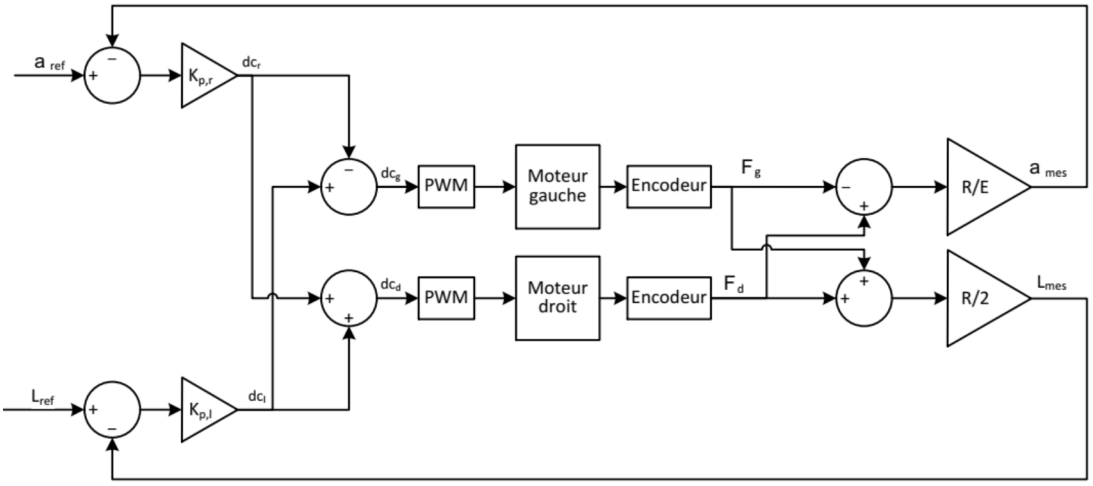

FIGURE 15 – Schéma de la régulation polaire

du régulateur est exprimée sous la forme d'un rapport cyclique d_{cr} pour la rotation et d_{cl} pour la translation. Dès lors, leur somme constituent en le signal réglant du moteur gauche tandis que leur différence règle le moteur de droite et ce, via les modules PWM. Les encodeurs permettent d'obtenir la positon du robot et ainsi de calculer l'erreur. Enfin, une rétroaction permet d'ajuster la vitesse de manière à atteindre la consigne.

Précision et zone morte

Par sa nature, cette régulation ne permettra jamais en pratique de réellement atteindre la consigne. En effet, le profil de vitesse inclus une décélération, une fois le robot proche de sa destination. Ainsi, au plus ce dernier avance, au moins il ira vite ; le rapport cyclique chute. Ce dernier se traduit par une tension très faible appliquée aux roues. Celle-ci n'est alors plus suffisante pour vaincre l'inertie des roues, le couple résistif, les frottements,... Le robot ne bouge plus. La différence entre sa positon finale et la consigne est communément appelée zone morte. Il est alors intéressent de l'évaluer et voir ce qui peut être mis en place pour la réduire au mieux et ce, dans une optique d'évaluation des *performances* du robot.

4.2.1 Génération de la consigne

Le profil de vitesse imposé a la forme décrite sur la figure 16. Il s'agit d'un profil trapé-

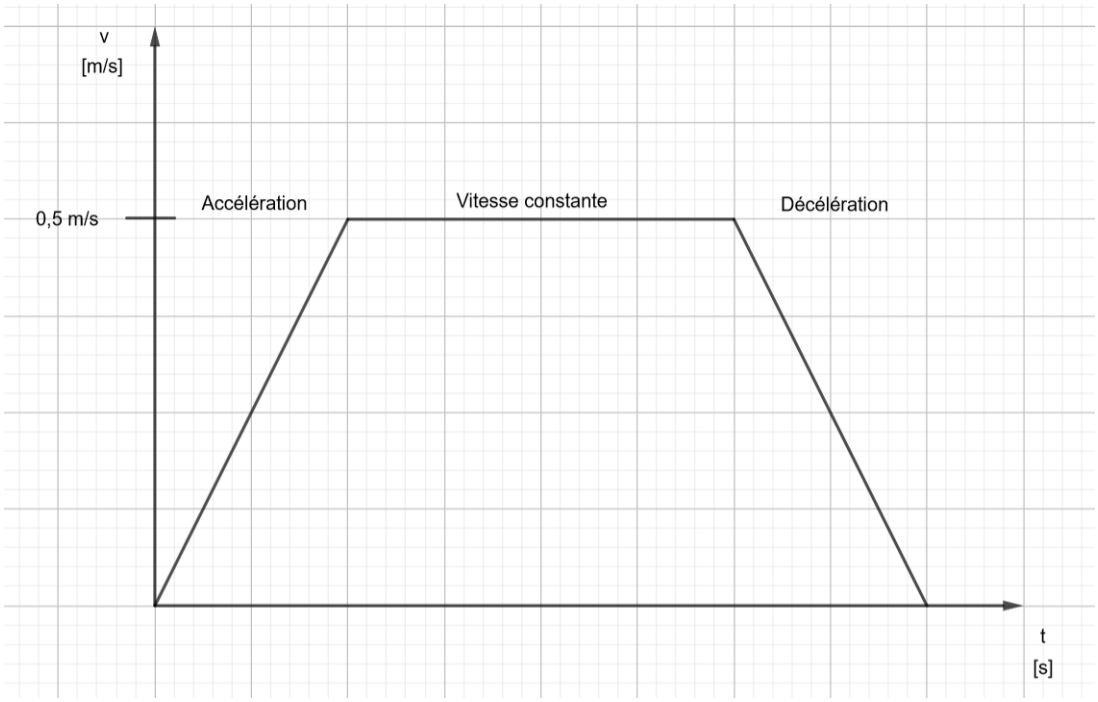

FIGURE 16 – Profil de vitesse du robot

zoïdal composé d'une phase d'accélération , une phase de vitesse constante et une décélération.

Méthode 1

La première méthode consiste en suivi de trajectoire. Elle nécessite ainsi d'intégrer le profil de position. Dès lors, à intervalle régulier (d'où la nécessité d'un timer en interruption), la consigne est générée en fonction du temps écoulé. Ceci a comme avantage d'avoir un facteur cyclique qui augmente graduellement car la différence entre la position du robot à l'instant t et la consigne à ce même instant t sera relativement faible. Elle demande toutefois plus de ressources au dsPIC.

Méthode 2

Afin de simuler ce profil, une autre possibilité est de définir le rapport cyclique comme proportionelle à une erreur de position dont la référence est cette fois-ci fixée à sa valeur finale. Toutefois, cela veut dire que le rapport cyclique sera très élevé dès le début car l'erreur de positon sera maximale dès le début. Afin de palier à ce problème, il convient de limiter le rapport cyclique de translation et de rotation à 0,5. Ceci laisse notamment place à la possibilité de réguler l'angle au

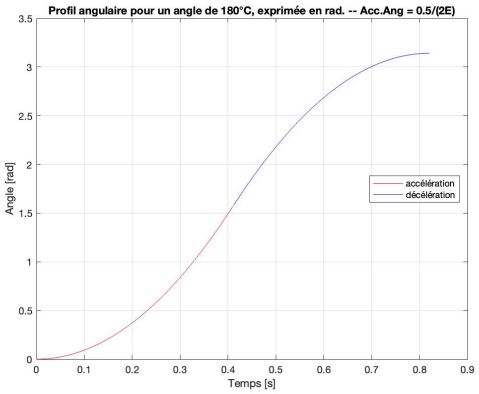

FIGURE 17 – Profil angulaire

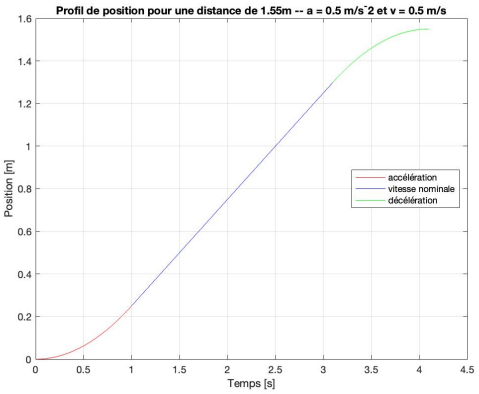

FIGURE 18 – Profil de position

cours d'une translation car le facteur cyclique total (soit la somme des 2) ne sera pas saturé à 1 et ainsi, jouir pleinement de la régulation polaire.

4.2.2 Régulateur proportionnel

Comme explicité précédemment, le régulateur mis en place est un régulateur proportionnel de gain K_{pr} pour la rotation et K_{pl} pour la translation. Les gains ont été obtenus grâce à la marche de phase et de gain qui ont été fixées respectivement à 30 degré et 6 dB. Dès lors le gain K_{pl} ayant été fourni vaut 3,69 m^{-1}. Il faut alors convertir ce gain en radian pour K_{pr}. Sachant

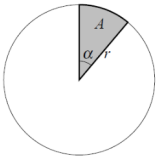

FIGURE 19 – Arc de cercle

que le rayon de l'arc de cercle formée par la rotation du robot vaut la moitié de l'empattement E, le lien entre l'angle et l'arc de cercle formée est le suivant :

$$\alpha = \frac{2}{E} * A \tag{6}$$

Et donc le lien entre K_{pr} et K_{pl} est :

$$K_{pr} = \frac{E}{2} * K_{pl} \tag{7}$$

4.2.3 Encodeurs et rétroaction

Conversion des valeurs numériques données par les registres

Les données fournies par les encodeurs se trouvent dans deux registres POS1CNT et POS2CNT qui s'incrémentent au fur et à mesure du mouvement du robot. Données en hexadécimal, il convient de convertir les valeurs stockées dans ces registres afin d'obtenir les mesures dans des unités appropriées. Premièrement, il faut utiliser un "cast" permettant de convertir ces valeurs en nombres entiers. Comme il s'agit d'encodeurs en quadrature, il convient de diviser la valeur des registres par 4 également. La mesure obtenue est alors en degré et correspond aux grandeurs réglées F_g et F_d sur la Figure 15.

Les "mesures" fournies à la rétroaction s'expriment donc de la manière suivante :

$$L_{mes} = \frac{R}{2}\frac{\pi}{180}(F_d + F_g) \tag{8}$$

$$\alpha_{mes} = \frac{R}{E}\frac{\pi}{180}(F_d - F_g) \tag{9}$$

4.3 Validation de la partie déplacement

La régulation via la "Méthode 2" a été aboutie. Celle-ci donne des résultats satisfaisants. Toutefois, le robot a tendance à partir légèrement en arc de cercle pour une translation malgré la régulation polaire. Plusieurs hypothèses sont possibles à cela comme un gain lié à la rotation trop faible ou une saturation du à la consigne en translation.

4.3.1 Traitement de la zone morte et précision

Afin de palier au problèmes de la zone morte, il convient par exemple, d'ajouter un "offset" à la consigne de départ. Ceci permet d'atteindre la vrai consigne de départ. Cet offset a été défini de manière itérative et empirique pour la translation et rotation. Il consiste en la multiplication de la consigne de départ par un certain facteur tel qu'on puisse obtenir une précision de l'ordre du centimètre pour la translation et du degré pour la rotation.

5 Conclusion

En conclusion, la réalisation de ce robot a permis de développer une multitude de connaissances essentiels dans le milieu de l'ingénieurie éléctronique. De l'éléctronique analogique, à l'automatique en passant par la télécommunication et l'instrumentation, ce projet à permis de souligner l'importance de l'ensemble des disciplines apprises au cours de cette formation d'ingénieur.

6 Annexe

6.1 Lien Github

https ://github.com/SergeAgM/ProjetIntegr-.git

6.2 Déplacement du robot (Méthode 2)

```
/*
 * File:   main.c
 * Author: User
 *
 * Created on 4 mai 2022, 22:23
 */

//Right  wheel = 1
//Left wheel = 2
#include "main.h"
#include "math.h"

int main(){
    frcPll40MHzConfig();
    int16_t Tram = 0b1100001111;
    //Instruction
    int16_t Intruction = 0b00;
    float Dist = 100;  //[cm]
    float Angle = 0; //[ ]
    //You'll need to make a condition on the dist/angle to be sure to have [m] or
    [  ]
    float error[2] ={Dist,Angle};
    //setups
    setup();

while(1){
    POS1CNT = 0x0000;
    POS2CNT = 0x0000;
    //Get the register of the wheel (1 and 2)

    if(Intruction == 0b00){
```

```
31          _LATB12 = 1; //Dir1 (forward with 3.3)
32          _LATB11 = 0; //Dir2 (forward with 0)
33
34          while(error[0] > 0.5f){
35
36              polar(error[0], error[1]);
37              error[0] = errorLength(Dist);
38              error[1] = errorAngular(Angle);
39              if(error[0]<1.0f)
40              {
41                  error[0]=0;
42                  error[1]=0;
43                  polar(error[0],error[1]);
44                      LATBbits.LATB4  = 1;
45              }
46          }
47      }
48          if(Intruction == 0b01){
49          _LATB12 = 0; //Dir1 (forward with 3.3)
50          _LATB11 = 1; //Dir2 (forward with 0)
51
52          while(error[0] < -0.5f){
53              polar(error[0], error[1]);
54              error[0] = errorLength(Dist);
55              error[1] = errorAngular(Angle);
56              if(error[0] > -1.5f){
57                  error[0]=0;
58                  error[1]=0;
59                  polar(error[0],error[1]);
60
61              }
62          }
63      }
64              if(Intruction == 0b11){
65          _LATB12 = 1; //Dir1 (forward with 3.3)
66          _LATB11 = 1; //Dir2 (forward with 0)
67
68          while(error[1] > 0.5f){
69              polar(error[0], error[1]);
70              error[0] = errorLength(Dist);
71              error[1] = errorAngular(Angle);
```

```
72
73              if(error[1] < 1.0f)
74              {
75                  error[0]=0;
76                  error[1]=0;
77                  polar(error[0],error[1]);
78
79              }
80
81
82          }
83      }
84
85      if(Intruction == 0b10){
86          _LATB12 = 0; //Dir1 (forward with 3.3)
87          _LATB11 = 0; //Dir2 (forward with 0)
88
89          while(error[1] < -0.5f){
90              polar(error[0], error[1]);
91              error[0] = errorLength(Dist);
92              error[1] = errorAngular(Angle);
93
94              if(error[1] > -5.0f){
95                  error[0]=0;
96                  error[1]=0;
97                  polar(error[0],error[1]);
98              }
99
100
101         }
102     }
103 }
104     return 0;
105 }
106
107 void polar(float errorLength, float errorAngle){
108
109
110     float dcr = errorAngle * Kpr;
111     float dcl = errorLength * Kpl;
112
```

```
113         if (dcl > 0.5f){
114         dcl = 0.5f;
115         }
116         else if (dcl < -0.5f){
117             dcl = -0.5f;
118 
119         }
120         if (dcr > 0.5){
121         dcr = 0.5;
122         }
123         else if (dcl < -0.5f){
124             dcr = -0.5f;
125 
126         }
127 
128 
129 
130     float dcd = dcl + dcr;
131     float dcg = dcl - dcr;
132 
133     if(dcd < 0.0f){
134         _LATB12= 0;
135         OC1RS = -(dcd)*PR2;
136     }
137     else{
138         OC1RS = (dcd)*PR2;
139 
140     }
141     if(dcg < 0.0f){
142         _LATB11= 1;
143         OC2RS = -(dcg)*PR2;
144     }
145     else{
146         OC2RS = (dcg)*PR2;
147     }
148 }
149 
150 float errorAngular(float angleRef){
151     angleRef = 1.5*angleRef;
152     float encod1 = ((int)POS1CNT)/(4);
153     float encod2 = ((int)POS2CNT)/(4);
```

```
154
155     float angleMes = ((-encod1+encod2) *R) / E ;
156     float errorAngle = angleRef + angleMes;
157     return errorAngle;
158 }
159
160 float errorLength(float lengthRef){
161     float encod1 = ((int)POS1CNT*PI*R)/(4*180);
162     float encod2 = ((int)POS2CNT*PI*R)/(4*180);
163
164     float lengthMes = ((encod2+encod1) ) / 2.0f ;
165     float errorLength = lengthRef - lengthMes;
166     return errorLength;
167 }
168
169 //
        ////////////////////////////////////////////////////////////////////////////////////////////////////
170
171 float errorwheelB(float dist, float errorL[]){
172
173     //return the error of 1
174
175
176         float encod1 = (int)POS1CNT*PI*R/(4*180);
177         float encod2 = (int)POS2CNT*PI*R/(180*4);
178         float moyencod = (encod1 + encod2) / 2.0f;
179         float error = dist - moyencod;
180
181
182         if (errorL[1] < 2.0f ){//
183             error = 0;
184         }
185
186
187
188
189           return error;
190
191 }
192
```

```
193
194
195
196 void setup(){
197
198     AD1PCFGL = 0xFFFF; //set all the pins on 1
199
200
201     //Set the pins of the encoder
202
203     //QEI 1
204     QEI1CONbits.QEIM = 0b111; //Fait registre pour data
205     RPINR14bits.QEA1R = 3;
206     RPINR14bits.QEB1R = 2;
207
208     //QEI2
209     QEI2CONbits.QEIM = 0b111;
210     RPINR16bits.QEA2R = 14;
211     RPINR16bits.QEB2R = 13;
212
213     //Pin Dir
214     TRISBbits.TRISB12 = 0;
215     TRISBbits.TRISB11 = 0;
216     TRISBbits.TRISB4 = 0;
217     TRISBbits.TRISB3 = 0;
218     _LATB12 = 1; //Dir1 (forward with 3.3)
219     _LATB11 = 1; //Dir2 (forward with 0)
220     _LATB4 = 0;
221     _LATB3 = 0;
222
223     // Initialize and enable Timer2
224     PR2 = 7999; // Load the period value; 40MHz*200us -1 = PR2 ->7999
225
226     //PWM1
227
228     //Driver setup
229     OC1RS = 0; // Write the duty cycle for the second PWM pulse
230     OC1R = 0;
231     OC1CONbits.OCM = 0b110; // Select the Output Compare mode
232     OC1CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
233         // Initialize Output Compare Module
```

```
234     _RP9R = 0b10010; //configure the pin to setup the pwm1 (for the second pwm,
        set the LSB at 1)
235     T2CONbits.TON = 1; // Start Timer
236
237
238     //PWM2
239
240     //Driver setup
241     OC2RS = 0; // Write the duty cycle for the second PWM pulse
242     OC2R = 0;
243     OC2CONbits.OCM = 0b110; // Select the Output Compare mode
244     OC2CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
245         // Initialize Output Compare Module
246     _RP10R = 0b10010; //configure the pin to setup the pwm1 (for the second pwm,
        set the LSB at 1)
247     T2CONbits.TON = 1; // Start Timer
248
249
250 }
251
252 // DSPIC33FJ128MC802 Configuration Bit Settings
253 // 'C' source line config statements
254 // FBS
255 #pragma config BWRP = WRPROTECT_OFF     // Boot Segment Write Protect (Boot
        Segment may be written)
256 #pragma config BSS = NO_FLASH           // Boot Segment Program Flash Code
        Protection (No Boot program Flash segment)
257 #pragma config RBS = NO_RAM             // Boot Segment RAM Protection (No Boot
        RAM)
258
259 // FSS
260 #pragma config SWRP = WRPROTECT_OFF     // Secure Segment Program Write Protect (
        Secure segment may be written)
261 #pragma config SSS = NO_FLASH           // Secure Segment Program Flash Code
        Protection (No Secure Segment)
262 #pragma config RSS = NO_RAM             // Secure Segment Data RAM Protection (No
        Secure RAM)
263
264 // FGS
265 #pragma config GWRP = OFF               // General Code Segment Write Protect (
        User program memory is not write-protected)
```

```
266 #pragma config GSS = OFF                // General Segment Code Protection (User
        program memory is not code-protected)
267
268 // FOSCSEL
269 #pragma config FNOSC = FRC              // Oscillator Mode (Internal Fast RC (FRC)
        )
270 #pragma config IESO = OFF               // Internal External Switch Over Mode (
        Start-up device with user-selected oscillator source)
271
272 // FOSC
273 #pragma config POSCMD = NONE            // Primary Oscillator Source (Primary
        Oscillator Disabled)
274 #pragma config OSCIOFNC = OFF           // OSC2 Pin Function (OSC2 pin has clock
        out function)
275 #pragma config IOL1WAY = ON             // Peripheral Pin Select Configuration (
        Allow Only One Re-configuration)
276 #pragma config FCKSM = CSECMD           // Clock Switching and Monitor (Both Clock
         Switching and Fail-Safe Clock Monitor are disabled)
277
278 // FWDT
279 #pragma config WDTPOST = PS32768        // Watchdog Timer Postscaler (1:32,768)
280 #pragma config WDTPRE = PR128           // WDT Prescaler (1:128)
281 #pragma config WINDIS = OFF             // Watchdog Timer Window (Watchdog Timer
        in Non-Window mode)
282 #pragma config FWDTEN = OFF             // Watchdog Timer Enable (Watchdog timer
        enabled/disabled by user software)
283
284 // FPOR
285 #pragma config FPWRT = PWR128           // POR Timer Value (128ms)
286 #pragma config ALTI2C = OFF             // Alternate I2C  pins (I2C mapped to SDA1
        /SCL1 pins)
287 #pragma config LPOL = ON                // Motor Control PWM Low Side Polarity bit
         (PWM module low side output pins have active-high output polarity)
288 #pragma config HPOL = ON                // Motor Control PWM High Side Polarity
        bit (PWM module high side output pins have active-high output polarity)
289 #pragma config PWMPIN = ON              // Motor Control PWM Module Pin Mode bit (
        PWM module pins controlled by PORT register at device Reset)
290
291 // FICD
292 #pragma config ICS = PGD1               // Comm Channel Select (Communicate on
        PGC1/EMUC1 and PGD1/EMUD1)
```

```
293 #pragma config JTAGEN = OFF              // JTAG Port Enable (JTAG is Disabled)
294
295 // #pragma config statements should precede project file includes.
296 // Use project enums instead of #define for ON and OFF.
297
298 #include <xc.h>
299
300
301 /* Configures PLL prescaler, PLL postscaler, PLL divisor to obtain Fosc = 80MHz
302  * with the FRC oscillator (Fin = 7.37MHz).
303  * We obtain Fosc = 7.37MHz*65/(*2) = 79.96MHz */
304 void frcPll40MHzConfig(void) {
305   // Fosc = Fin*M/(N1*N2), where :
306   //     M = PLLFBD + 2
307   //     N1 = PLLPRE + 2
308   //     N2 = 2 x (PLLPOST + 1)
309     PLLFBD = 63;
310     CLKDIVbits.PLLPRE = 1;
311     CLKDIVbits.PLLPOST = 0;
312
313   // Initiate Clock Switch to FRC with PLL
314   __builtin_write_OSCCONH( 0x01 );
315   __builtin_write_OSCCONL( OSCCON | 0x01 );
316   // Wait for Clock switch to occur
317     while (OSCCONbits.COSC != 0b001);
318     // Wait for PLL to lock
319     while(OSCCONbits.LOCK != 1);
320 }
```

Listing 1 – Code exemple pour le déplacement du robot