



Coupe de France de Robotique

Base roulante

Lucie GARNIER

Tuteur : M. NOIZETTE



**UNIVERSITÉ
DE LORRAINE**

LORRAINE INP
vos talents se lèvent à l'Est

Juin 2022



Sommaire

I - Introduction.....	4
II - Configuration de STM.....	5
II.a. - Timers.....	5
II.b. - Direct Memory Access (DMA)	8
III - Gestion des moteurs	9
III.a. - Electronique des moteurs.....	9
III.b. - Gestion des déplacements.....	10
III.c. - Gestion de la vitesse et des accélérations, décélérations.....	12
IV - Problèmes rencontrés	14
V - Conclusion et perspectives	15
V.a. - Gestion des accélérations, décélérations.....	15
V.b. - Correction des déplacements en temps réel	15
V.c. - Conclusion	16



Figure 1 : Schéma du montage pour piloter un moteur.....	4
Figure 2 : Principe de fonctionnement d'un timer.....	5
Figure 3 : Capture d'écran du logiciel STM32, pour configurer le timer 1.....	7
Figure 4 : Capture d'écran du logiciel STM32, pour activer les interruptions du timer 1.....	7
Figure 5 : Capture d'écran du logiciel STM32, pour paramétrer le DMA du timer 1	8
Figure 6 : Pinout du driver TMC2208.....	9
Figure 7 : Schéma des pins utilisé sur la STM pour élaborer la carte moteur	9
Figure 8 : Schéma des différentes directions et rotations de la base roulante	11
Figure 9 : Signaux du timer et des trois moteurs	12
Figure 10 : Fréquence généré par le timer maître du contrôle moteur d'une accélération	13
Figure 11 : Schéma d'une direction alternative.....	15
Tableau 1 : Paramétrage du timer 1 pour la base roulante	6
Tableau 2 : Paramétrage des micro-steps.....	9
Tableau 3 : Trame déplacement.....	10
Tableau 4 : Trame moteur	11



I - Introduction

L'objectif de ce pôle base roulante est de réaliser la partie informatique qui permettra au robot de se déplacer. Nous avons recommencé depuis le début toute la partie base roulante, l'objectif est d'avoir une base fiable en fin d'année et de l'améliorer au fil des ans.

Cette année, nous avons choisi de réaliser une base roulante avec trois roues, contre quatre l'année précédente. Cette solution nous permet de ne pas avoir de problèmes d'hyperstaticité. Nous utilisons des roues holonomes, ce sont des roues omnidirectionnelles. Leur principal avantage est que nous pouvons aller dans six directions différentes sans effectuer de rotations, ce qui est un gain de temps et d'énergie. Chaque roue est motorisée par un moteur *NEMA 17*, lui-même piloté par un driver *TMC2208*. On gère les pilotes des moteurs avec un microcontrôleur *STM32* (modèle de carte nucléo *L432KCU3*). Ce microcontrôleur est commandé par la stratégie, qui envoie les actions que la base roulante doit effectuer via le bus CAN.

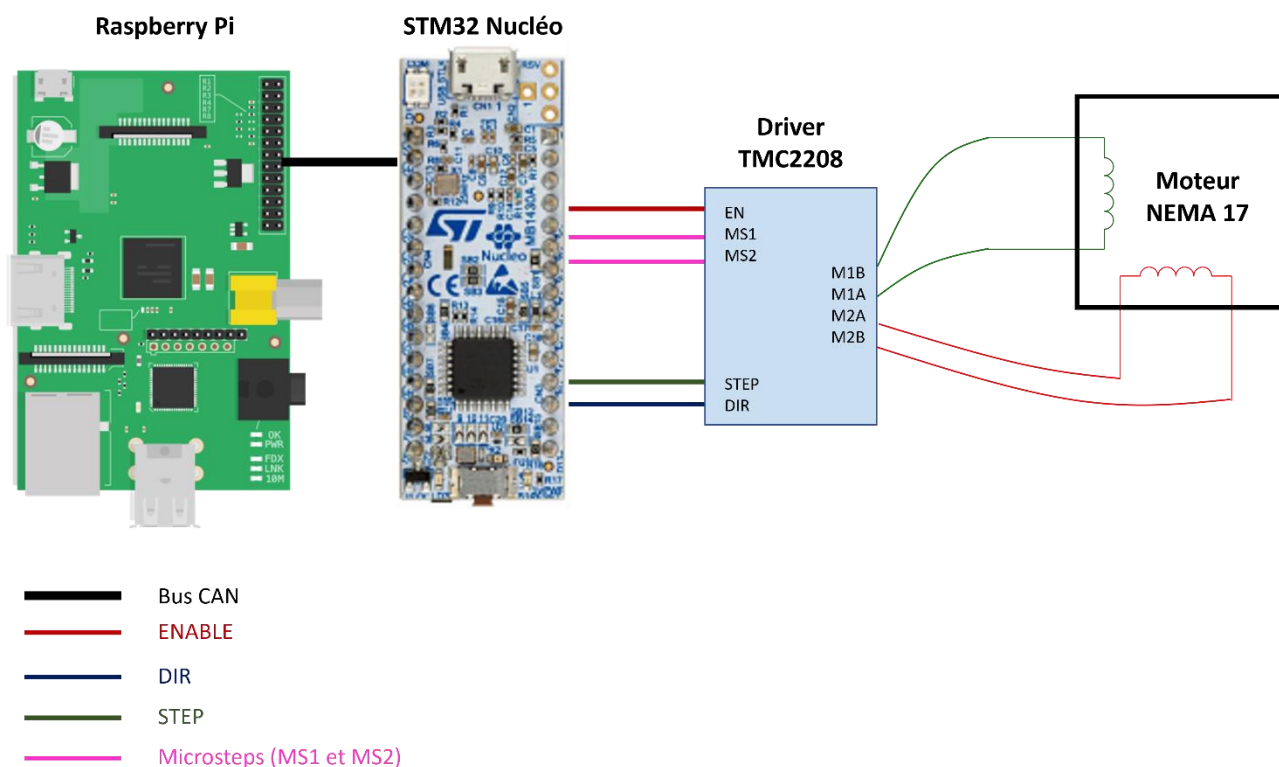


Figure 1 : Schéma du montage pour piloter un moteur



II - Configuration de STM

II.a. - Timers

Pour programmer les *timers*, il faut ouvrir le fichier IOC et sélectionner *timer* dans la colonne de gauche. L'Autoreload Register (ARR) est commun à toutes les *channels* du *timer*, alors que le Capture/Compare Register (CCR) est spécifique à chaque *channel*. L'ARR va définir la période à laquelle l'interruption va avoir lieu, tandis que le CCR définira la largeur de l'impulsion. Nous avons ensuite deux modes : haut ou bas. Sur la *Figure 2*, c'est le mode haut qui est représenté, le mode bas est son complémentaire. Chaque *channel* de *timer* est reliée à une unique pin, en revanche une pin peut être connectée à plusieurs *channel* de différents *timers*.

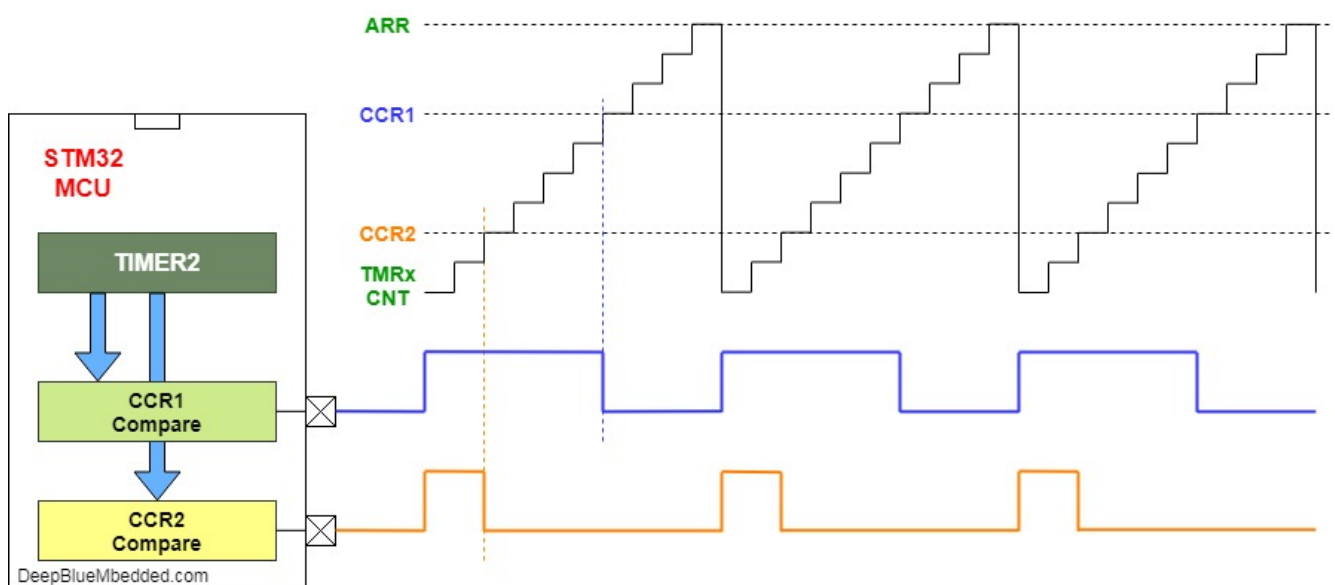


Figure 2 : Principe de fonctionnement d'un timer

On commence par sélectionner le *timer* souhaité et on active une horloge (*clock source*). On utilise l'horloge interne mais on pourrait aussi utiliser une horloge externe. Cependant, toutes les cartes ne disposent pas de cette source externe il faut donc s'assurer qu'elle est bien présente. Puis, on choisit le mode de fonctionnement du *channel*, dans notre cas on choisit *PWM Generation Channel 1*. Ce mode nous permet d'avoir un signal de sortie sur la pin PA8, ce qui est pratique pour le débogage. Ensuite, il faut paramétrer le *timer*, dans *configuration* et *parameter settings* :

- *Prescaler* : permet de diviser la fréquence source. Il faut soustraire 1 pour que la sortie soit bien à la valeur souhaitée, sinon il y aura un léger décalage.

$$\text{fréquence de sortie prescaler} = \frac{\text{fréquence clock}}{\text{prescaler}}$$

- *Counter mode* : en mode *up* commence à compter à partir de 0 et incrémente. A l'inverse, *down* va partir de la valeur de l'ARR et décrémenter jusqu'à 0. Il existe un troisième mode, *center align*, celui-ci va démarrer à 0, incrémenter jusqu'à l'ARR au lieu de recommencer à 0 comme le mode *up*, il va décrémenter jusqu'à 0.
- *Counter period (AutoReload Register)* : c'est l'ARR, il faut soustraire 1, car le compteur commence à 0 et non à 1, sans la soustraction, on aurait une fréquence qui ne serait pas tout à fait celle que l'on souhaite. On lui donnera une valeur de départ pas très grande pour



éviter qu'il ne mette sa valeur maximum par défaut et que la première interruption ait lieu après une longue attente. Cependant, cette valeur sera changée par la suite lors des phases d'accélération et de décélérations, puisque c'est ce qui nous permet de changer de vitesse.

$$\text{fréquence de sortie ARR} = \frac{\text{fréquence de sortie prescaler}}{\text{ARR}}$$

- *Pulse* : c'est le CCR, la durée du temps haut, il doit toujours être inférieur à l'ARR. Ici, on lui donnera une valeur fixe, car on joue sur la durée de la période et non la largeur de l'impulsion pour définir la vitesse.

Nous laisserons les autres paramètres avec leur valeur par défaut.

Il faut faire attention aux nombres de bits sur lequel les nombres sont codés. Ici, on a des valeurs faibles, mais si on venait à avoir besoin de nombre plus grands, il faudrait choisir le *timer* en conséquence. Par exemple, nous utilisons *timer 1*, dont l'ARR est codé sur 16 bits, alors que celui du *timer 2* est sur 32 bits. Certains *timers*, sur la carte n'ont qu'une seule *channel*, si on venait à avoir besoin plusieurs *channel* et qu'elles devaient être sur le même *timer*, c'est un élément qu'il faudrait aussi prendre en compte.

<i>Parameter Setting</i>	Valeur
Timer	TIM1
Clock Source	Internal clock
Channel 1	PWM Generation CH1
Prescaler	16-1
Counter mode	Up
Counter Period (Autoreload Register)	100-1
Pulse	1

Tableau 1 : Paramétrage du timer 1 pour la base roulante

Calcul de la fréquence obtenue :

$$\text{fréquence de sortie prescaler} = \frac{\text{fréquence clock}}{\text{prescaler}} = \frac{16 \times 10^6}{16} = 1\text{MHz}$$

$$\text{fréquence de sortie ARR} = \frac{\text{fréquence de sortie prescaler}}{\text{ARR}} = \frac{1 \times 10^6}{100} = 10\,000\text{Hz}$$

Ici, la valeur de l'ARR correspond à sa valeur lors du maintien de vitesse.

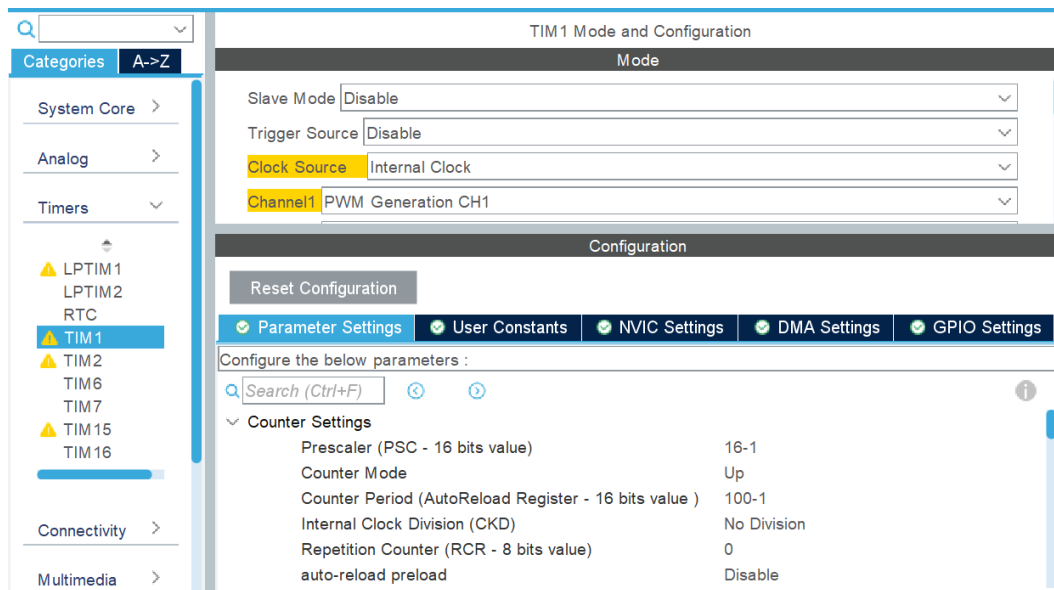


Figure 3 : Capture d'écran du logiciel STM32, pour configurer le timer 1

Cependant, même après avoir entré tous ces paramètres correctement, il ne se passera rien. En effet, il ne faut pas oublier d'activer les interruptions. Pour cela, il faut aller dans l'onglet *NVIC Setting* (toujours dans la configuration du timer utilisé), est cocher les différentes cases comme sur la *Figure 4* ci-dessous. Sur certaines versions, il est possible de n'avoir qu'une seule case à cocher et qui activera toutes les interruptions.

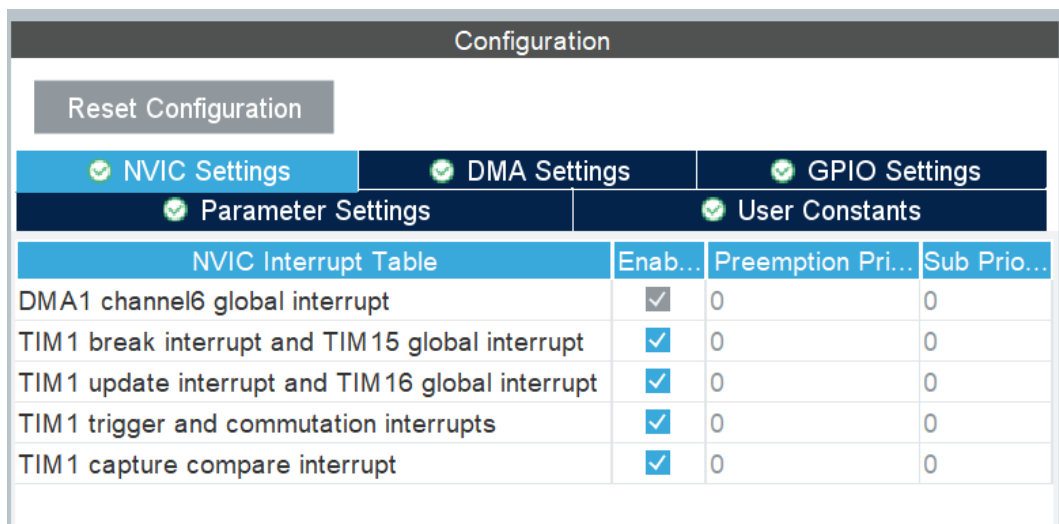


Figure 4 : Capture d'écran du logiciel STM32, pour activer les interruptions du timer 1



II.b. - Direct Memory Access (DMA)

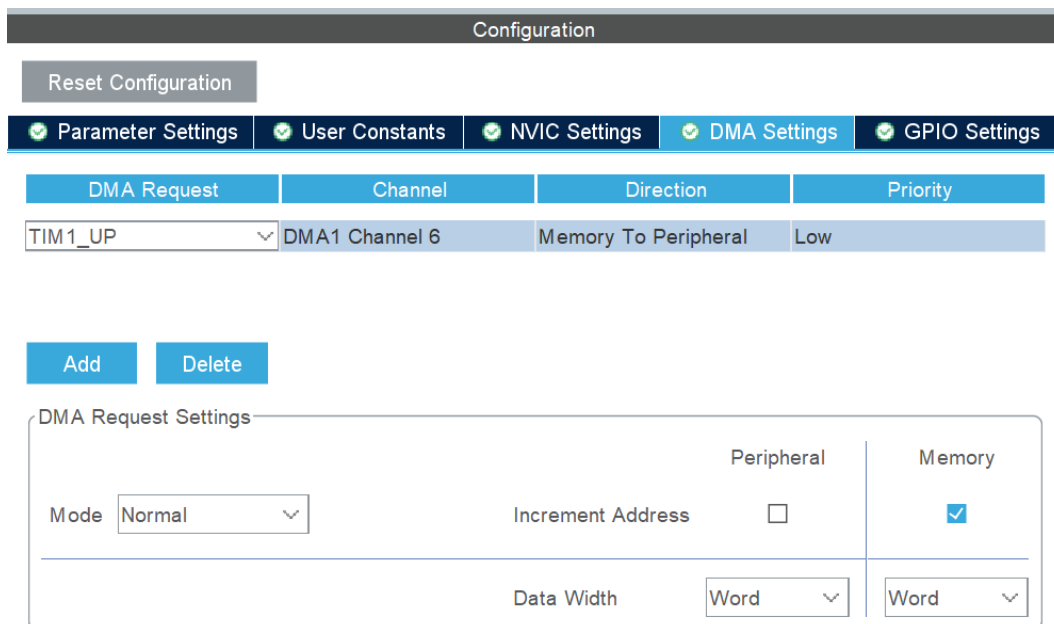
L'Accès direct à la mémoire est un procédé qui permet à des données de circuler depuis, ou vers, un périphérique et d'être transférées vers la mémoire principale, grâce à un contrôleur adapté et avec une intervention minimale du microprocesseur (pour commencer et terminer l'opération). Ce qui permet au microprocesseur d'effectuer d'autres actions pendant le processus.

Nous allons utiliser ce procédé pour charger les tableaux d'accélérations et de décélérations (la génération de ces tableaux sera vu dans une prochaine partie). Pour cela, toujours dans la partie configuration du *timer*, on va dans l'onglet *DMA*. Par défaut aucun *DMA* n'est actif, nous allons donc en ajouter un avec *add*. On va ensuite le paramétrer :

- *DMA request* : on sélectionne l'option avec *up*
- *Direction* : nous avons deux options *memory to peripheral* (on envoie des données vers un périphérique) ou *peripheral to memory* (on récupère les données d'un périphérique et on les envoie à la mémoire). Nous choisissons la première option.

Ensuite dans *DMA request settings* :

- *Mode* : nous avons le choix entre *circular*, dans le cas d'un tableau les cases de ce dernier seront envoyées une à une, et lorsqu'on arrivera à la fin du tableau on recommencera à renvoyer les données à partir de la première case du tableau. Alors qu'avec le mode *normal*, la première fois que l'on arrive à la fin du tableau, on garde la dernière valeur envoyée, on ne recommence pas. C'est cette deuxième option que nous choisissons.
- *Increment address* : *peripheral* ou *memory*, nous cochoons *memory*, car c'est l'adresse du tableau qui se trouve dans la mémoire que nous voulons incrémenter.
- *Data width* : on transmet un *word* à chaque fois.



The screenshot shows the 'Configuration' window of the STM32CubeMX software. The 'DMA Settings' tab is selected. A table lists the configured DMA requests:

DMA Request	Channel	Direction	Priority
TIM1_UP	DMA1 Channel 6	Memory To Peripheral	Low

Below the table are 'Add' and 'Delete' buttons. The 'DMA Request Settings' dialog is open, showing the following configuration:

- Mode:** Normal (selected)
- Increment Address:** ☐ (unchecked)
- Data Width:** Word (selected)
- Peripheral:** ☐ (unchecked)
- Memory:** ☒ (checked)

Figure 5 : Capture d'écran du logiciel STM32, pour paramétrer le DMA du timer 1

III - Gestion des moteurs

III.a. - Electronique des moteurs

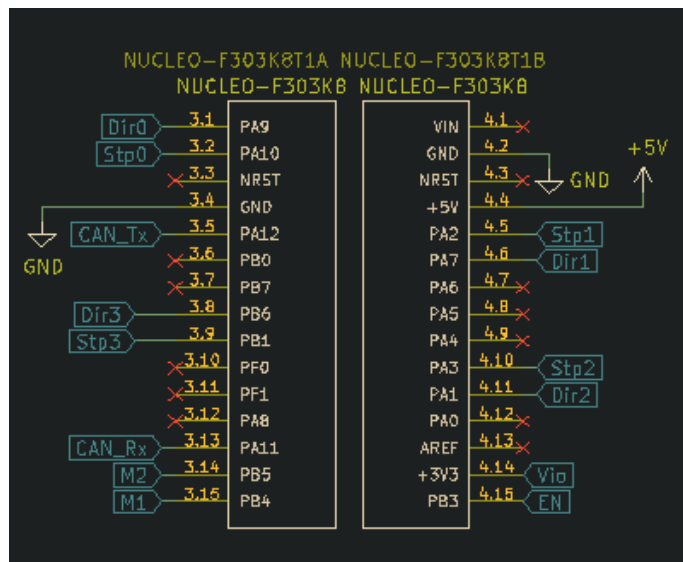


Figure 7 : Schéma des pins utilisés sur la STM pour élaborer la carte moteur

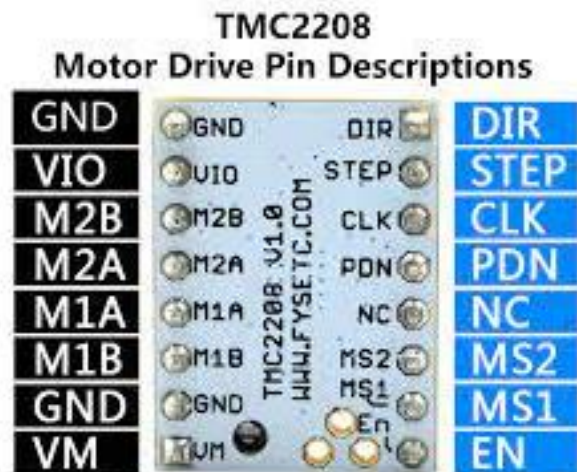


Figure 6 : Pinout du driver TMC2208

Tout d'abord, nous configurons la taille des pas. Avec la nouvelle version de la carte moteur, le paramétrage de la taille des pas se fait maintenant informatiquement, alors qu'avec l'ancienne carte cela dépendait d'une combinaison d'interrupteurs.

Sans diviseur, il faut deux cents pas pour faire un tour complet. Dans l'IOC, ces pins sont en mode *Push Pull*. Ci-dessous, le [Tableau 2](#) indique le paramétrage des pins pour obtenir la taille de pas souhaitée. Nous choisissons de configurer les drivers avec des demi-steps. Pour cela, nous utilisons la fonction `motor_micro_step()`, qui prend en entrée l'état des pin *MS1* et *MS2*, et définit la même taille de pas pour tous les moteurs.

MS1	MS2	Diviseur des micro-steps
GND	GND	8
VCC_IO	GND	2
GND	VCC_IO	4
VCC_IO	VCC_IO	16

Tableau 2 : Paramétrage des micro-steps



Pour que les moteurs puissent tourner, il faut qu'un courant électrique passe dans les bobines. Cela est permis avec la pin *enable*, qui est en mode *Push Pull*. Comme les micro-steps, sur la carte précédente, *enable* était activé, ou non, grâce à un interrupteur. La fonction *motor_en()*, permet d'activer ou de désactiver tous les moteurs en même temps. Si la fonction a pour paramètre *SET*, les moteurs ne sont pas actifs, au contraire si le paramètre est *RESET* alors les moteurs sont actifs. Lorsque les moteurs sont actifs et à l'arrêt ils sont clampés, c'est-à-dire qu'il faut forcer pour leur faire passer un pas.

Le sens de rotation du moteur dépend de l'état de la pin *dir*, qui est aussi en mode *Push Pull*. Nous commandons la direction d'une unique roue avec la fonction *motor_dir()*. Dans la configuration actuelle, c'est-à-dire lorsque le fil noir est branché sur *A1*, et lorsque la pin est à l'état haut, le moteur tourne dans le sens trigonométrique, à l'inverse, lorsqu'elle est à l'état bas, le moteur tourne dans le sens anti-trigonométrique. Cependant, si nous venions à inverser le branchement, alors le moteur tournerait dans le sens opposé. Il est donc important de toujours relier les moteurs aux drivers de la même manière.

Enfin, pour que les moteurs puissent tourner, il faut leur envoyer des impulsions, c'est la partie la plus compliquée. Nous générons des impulsions grâce au *timer* que nous avons à paramétrer précédemment. De plus, chaque moteur possède un compteur et un diviseur qui lui sont propres et qui sont indépendants des autres moteurs. A chaque impulsion générée par le *timer*, cela déclenche une interruption, qui va incrémenter le compteur de tous les moteurs. Lorsqu'un compteur atteint la valeur du diviseur associé à ce même moteur, alors le moteur va effectuer un pas et le compteur se réinitialisera à zéro et recommencera à s'incrémenter dès la prochaine interruption.

III.b. - Gestion des déplacements

La base roulante fonctionne avec des unités qui sont trop abstraites pour qu'une personne ou un autre système puissent les interpréter. C'est pour cela qu'il faut convertir les unités du système international en unités de la base roulante. Nous réalisons ces calculs, sur la Raspberry Pi, grâce à une fonction qui va convertir une « trame déplacement », telle que sur le Tableau 3, en une « trame moteur », telle que sur le Tableau 4.

On donne en entrée de la fonction une trame déplacement, qui est composée de trois valeurs :

- *Vitesse* : en millimètre par seconde, ce qui nous permet de toujours donner un nombre entier et de ne pas avoir de nombre à virgule ce qui serait plus compliqué à gérer.
- *Distance* : en millimètre, comme précédemment nous donnons toujours un nombre entier.
- *Direction* : est un nombre entre 1 et 8. Si le nombre est compris entre 1 et 6, le robot avancera dans l'une des directions comme indiquée sur la Figure 8. Si l'on donne 7 ou 8 en argument le robot effectuera une rotation sur lui-même.

Nombre de bits	16	16	8
Description	Vitesse [mm/s]	Distance [mm]	Direction

Tableau 3 : Trame déplacement
Trame à convertir, envoyée par la stratégie

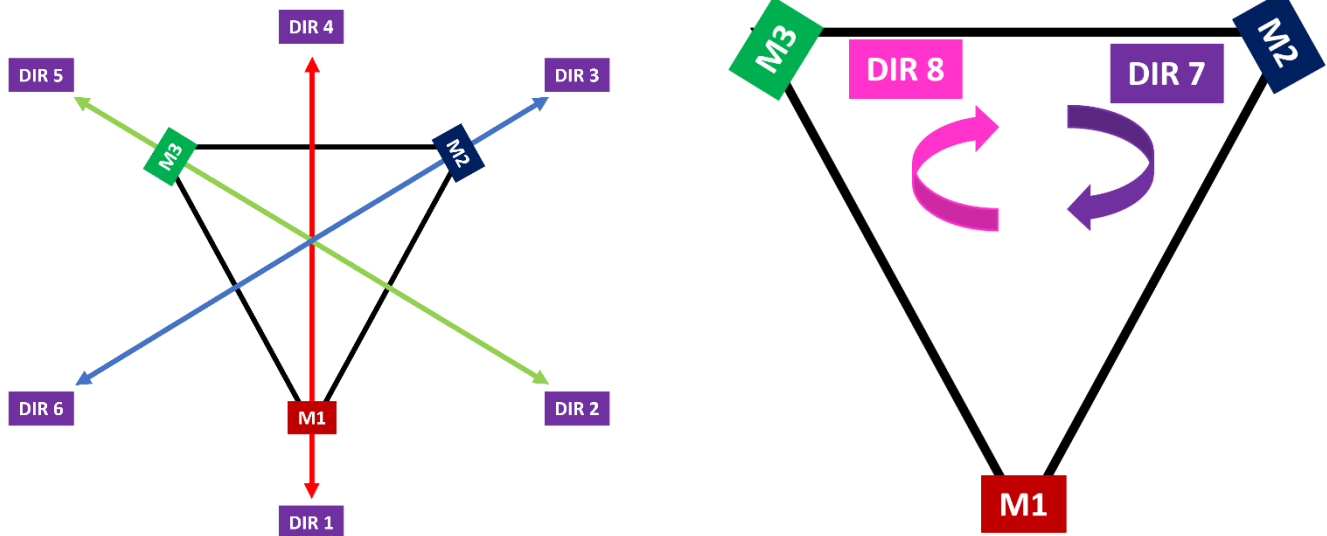


Figure 8 : Schéma des différentes directions et rotations de la base roulante

La fonction de conversion retourne une trame moteur, cette dernière est composée de sept valeurs :

- *Nombre de steps* : c'est le nombre de pas que le robot devra effectuer, le nombre d'impulsions qui seront émises par le timer.
- *Trois diviseurs* (un pour chaque moteur) : c'est ce qui permet de gérer la vitesse. Plus le diviseur est grand moins le robot va vite.
- *Trois directions* (une pour chaque moteurs) : c'est un booléen qui indique dans quel sens la roue va tourner. Par exemple, si le robot tourne sur lui-même les trois roues tourneront dans le même sens. Alors que si le robot effectue une ligne droite, il y aura une roue à l'arrêt, une roue qui tourne dans un sens et la dernière roue qui tournera dans le sens opposé à la deuxième.

Nombre de bits	32	8	8	8	1	1	1
Description	Nombre de steps	Diviseur moteur 1	Diviseur moteur 2	Diviseur moteur 3	Direction moteur 1	Direction moteur 2	Direction moteur 3

*Tableau 4 : Trame moteur
Trame convertie et envoyée à la base roulante*



III.C. - Gestion de la vitesse et des accélérations, décélérations

Pour gérer la vitesse, on change la valeur de la période entre deux impulsions, c'est-à-dire on change l'ARR. Cependant, nous avons un unique *timer* pour trois roues et nous voulons qu'elles puissent avoir des vitesses différentes. Pour cela on ajoute deux variables, un « compteur » et un « diviseur », pour chaque moteur. Le principe est simple, le *timer* émet des impulsions, à chaque impulsion, le compteur de chaque moteur s'incrémente de 1. Lorsque qu'un compteur atteint la valeur du diviseur de son moteur, alors il y aura une impulsion sur la pin *STEP* associée au moteur et il avancera d'un pas, et son compteur recommencera à s'incrémenter à partir de 0. Ainsi, plus le diviseur est grand, plus le compteur mettra du temps à l'atteindre, et donc à effectuer un pas. Pour qu'un moteur reste à l'arrêt, son diviseur doit être égal à 0. On obtient trois diviseurs différents, donc trois vitesses différentes. Sur la *Figure 9*, on voit que les impulsions des moteurs sont synchronisées avec celles du *timer*. De plus, c'est le moteur vert qui ira le plus vite et le moteur violet ira plus lentement.

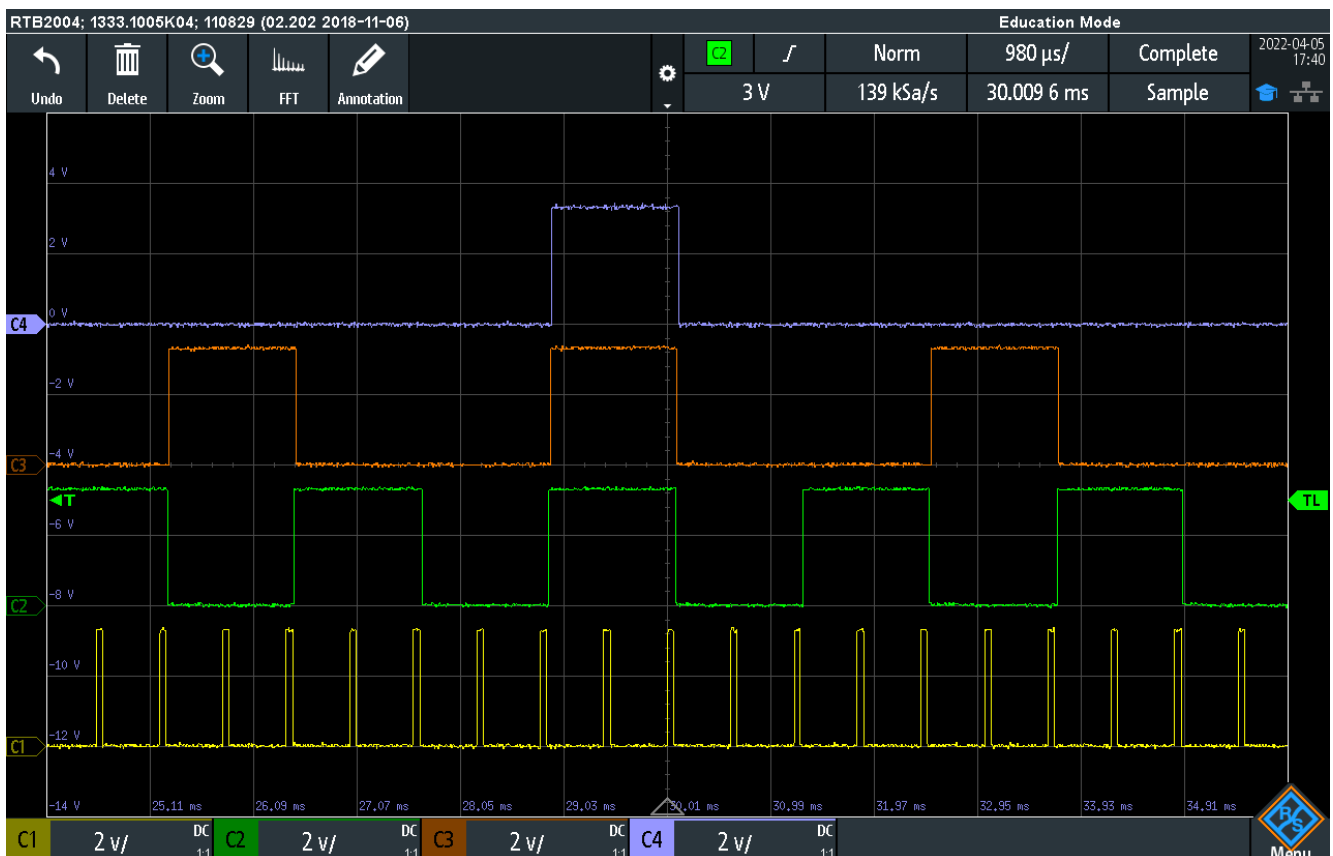


Figure 9 : Signaux du timer et des trois moteurs

En jaune le timer, en vert le moteur 1, en orange le moteur 2 et en violet le moteur 3

Les accélérations et décélérations sont générées avec un programme à part qui ne se trouve ni sur la Raspberry Pi, ni sur la carte STM. Le programme crée un tableau, qui aura comme données toutes les valeurs d'ARR du *timer*, pour réaliser les accélérations en fonctions des paramètres que nous lui auront fourni précédemment, telle que la fréquence de l'horloge de la STM, le *prescaler* utilisé, la fréquence de départ et d'arrêt, et le temps d'accélération souhaité. Ensuite, le tableau est inversé



pour avoir les décélérations. Finalement, nous récupérons la longueur des tableaux, qui est la même car les accélérations et décélérations sont symétriques, ainsi que les deux tableaux générés précédemment. On copie ces résultats et on les colle dans les fichiers utilisés pour les variables dans le code de la base roulante sur la STM. Sur la Figure 10, nous avons tracé la courbe sigmoïde de l'accélération, grâce à une fonction logistique. La fréquence de départ est de 1000Hz, et on atteint 10 000Hz en vitesse de maintien.

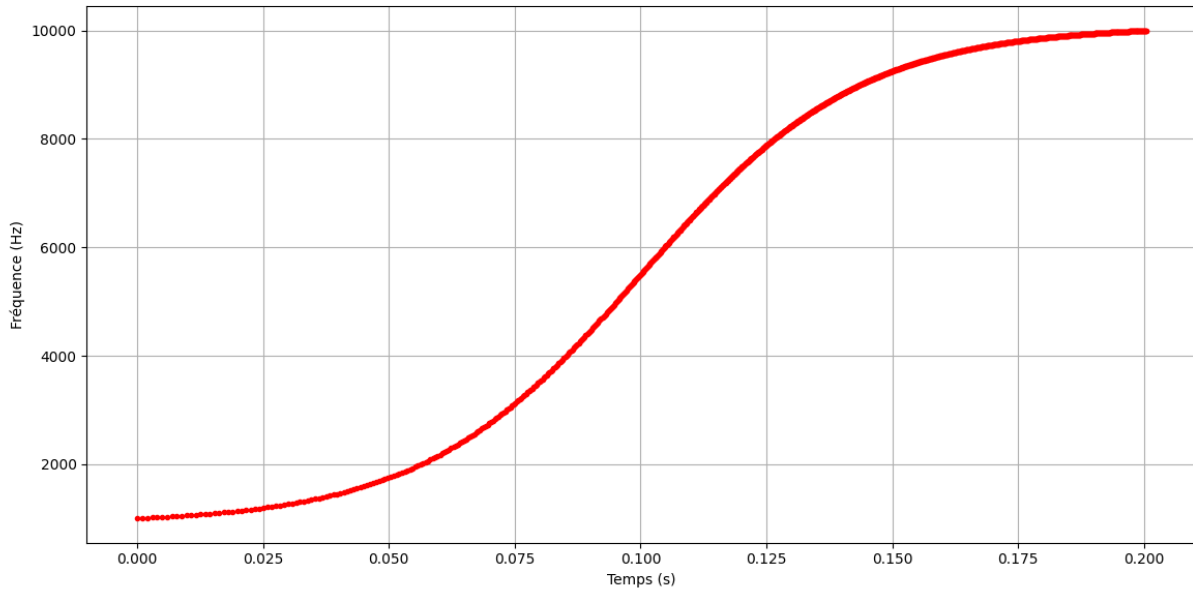


Figure 10 : Fréquence généré par le timer maître du contrôle moteur d'une accélération



IV - Problèmes rencontrés

Cette année, nous avons rencontré quelques problèmes liés aux moteurs et à leur branchement. Avant de chercher les potentiels erreurs dans le programme, il faut vérifier certains points directement sur le robot. Dans un second temps, on pourra vérifier le programme.

Premièrement, nous devons nous assurer que tous les moteurs soient bien clampés lorsque que la carte de la base roulante est active. Si un des moteurs n'est pas clampé, alors le robot n'effectuera pas le bon déplacement. Par exemple, il va effectuer un cercle au lieu d'une ligne droite ou il ne bougera pas du tout si aucun moteur n'est clampé.

Deuxièmement, il faut vérifier que le branchement des moteurs est correct, que les fils n'ont pas n'ont pas été inversés, sinon cela modifiera le déplacement. Par exemple, la trajectoire sera un cercle au lieu d'une ligne droite ou lorsqu'il effectuera une rotation sur lui-même elle sera dans le sens opposé à celle que l'on souhaite.

De plus, lorsque nous avons effectué des tests en continu il s'est avéré que plus nous faisons rouler le robot moins il était précis. C'était un problème mécanique, au fil du temps, les roues se désolidarisaient du moteur, il y avait un peu de jeu et les trajectoires devinaient moins précises. Il faut donc penser à vérifier de temps en temps que toutes les pièces du robot sont bien maintenues.

Ensuite, au niveau du programme, s'il ne se passe rien, il faut vérifier que des interruptions sont bien générées. Pour cela, on regarde, avec un oscilloscope, la pin qui est associée au *channel* du *timer* que l'on a choisi précédemment, il doit y avoir des impulsions.

Lorsque nous avons eu monté le robot nous avons constaté qu'il effectuait 20% de la distance en plus lors d'un trajet en ligne droite. Cela vient de la fonction de conversion de trame déplacement en trame moteur, il doit y a avoir une erreur de calcul dans le nombre de pas mais ne l'ayant pas trouvé et étant pressés par le temps (nous avons pu effectuer ces tests seulement à la compétition), nous avons décidé d'enlever les 20% qui étaient en trop, grâce à une simple soustraction. Nous avons pu faire ceci car l'erreur était constante, elle était toujours de 20% peu importe la distance parcourue ou la direction choisie. A l'inverse lors d'une rotation sur lui-même le robot ne tournait pas assez, mais là encore l'erreur était toujours constante peu importe le sens de rotation ou l'angle choisi, nous avons donc ajouté 18,75% au nombre steps final.

Enfin, si le robot ne prend pas en compte un nouvel ordre de déplacement qui lui a été envoyé et qu'il renvoie 00, c'est qu'il est déjà en train d'effectuer un trajet. Il faut soit attendre qu'il ait fini son déplacement, soit lui ordonner de s'arrêter et renvoyer l'ordre une fois qu'il est à l'arrêt. Lorsque l'on interrompt un trajet avec le code fonction *STOP*, la base roulante renvoie immédiatement une trame moteur, c'est-à-dire au début de la phase de décélération, mais elle va encore parcourir une certaine distance avant d'être totalement à l'arrêt. Le nombre de pas qui est renvoyé, c'est le nombre de pas qu'il restait à effectuer pour faire complètement le trajet moins le nombre de pas qui seront nécessaire pour s'arrêter (on connaît ce nombre puisque la déclaration est toujours la même). Cependant, puisque la base roulante n'est pas à l'arrêt, si on renvoie un ordre déplacement immédiatement après la réception de la trame envoyée par la base roulante alors l'ordre ne sera pas pris en compte. Il faut attendre le temps de la décélération (temps que l'on connaît également puisque c'est toujours le même).



V - Conclusion et perspectives

V.a. - Gestion des accélérations, décélérations

Une des premières améliorations à apporter serait de rendre les accélérations et décélérations asymétriques. Actuellement, elles sont symétriques, ce qui pose un problème lorsque nous atteignons des vitesses élevées, de l'ordre du mètre par seconde. A cette vitesse, l'accélération se passe encore bien mais lors de la décélération le robot commence à basculer, si l'on augmentait encore la vitesse, il se renverserait complètement. En effet, le poids du robot entre en jeu, notamment celui de la batterie qui est assez important et elle est placée en hauteur ce qui augmente son impact. Il faudrait donc pouvoir rendre la décélération indépendante de l'accélération, et la rendre plus longue. Mais si nous faisons cela il faudra aussi revoir le périmètre de la détection de collision, c'est-à-dire l'agrandir car si la décélération est plus longue alors le robot parcourra une plus grande distance avant de s'arrêter.

Dans un deuxième temps, on pourra faire en sorte que les accélérations et décélérations soient calculées en temps réel. Pour cela, il faudra implémenter sur la STM le programme qui génère les tableaux d'accélérations. Cependant cela sera plus compliqué et plus long à faire que de rendre les accélérations et décélérations asymétriques.

V.b. - Correction des déplacements en temps réel

Une autre amélioration possible est la correction en temps réel de la trajectoire avec un correcteur sur la Raspberry Pi.

Cependant, pour réaliser cela il faudra que la base roulante puisse accepter un nouveau déplacement alors qu'elle n'est pas arrêtée. Actuellement, si le robot n'est pas totalement arrêté, il ne peut débuter un nouveau déplacement, il prendra en compte uniquement les instructions pour s'arrêter. C'est une sécurité qui a été mise en place afin d'éviter que deux instructions de trajet ne se mélangent et que le programme plante. Le robot devra aussi pouvoir changer de vitesse sans s'arrêter.

Il faudra également aller dans une direction différente des six directions possibles actuellement, comme sur la *Figure 11*. Pour cela, il faut que les trois roues aient une vitesse différente. Cette partie est déjà réalisée, on peut donner trois vitesses différentes mais le robot irait dans une direction qui ne serait pas celle que l'on souhaite. Il manque les calculs qui permettront de faire rouler le robot avec la bonne vitesse générale et le bon angle.

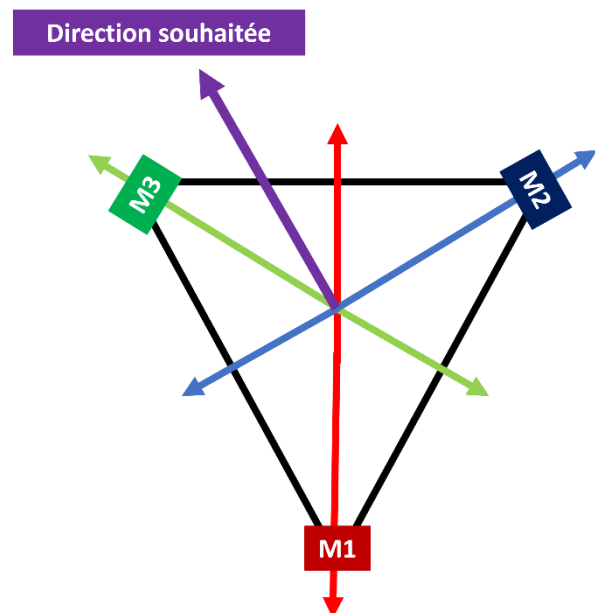


Figure 11 : Schéma d'une direction alternative



V.c. - Conclusion

Pour conclure, cette année la base roulante a été rendue fonctionnelle et relativement fiable. Il est important de conserver ce qui a été fait et de la transmettre pour l'année suivante. En effet, la base roulante est loin d'être parfaite et nous pouvons l'améliorer de nombreuses manières.

Parmi les problèmes cités précédemment, certains étaient récurrents, mais nous perdions quand même du temps à chercher d'où le problème venait. Il faudrait en dresser une liste « problème, source du problème et solution » sur laquelle nous pourrions nous appuyer et afin de pouvoir vérifier plus facilement et plus rapidement.

Pour les années suivantes, il faudrait aussi que plusieurs personnes soient capables de comprendre le fonctionnement de la base roulante, pas forcément dans les détails mais que ces personnes puissent régler des problèmes simples et la commander.