



# École Nationale Supérieure des Mines de Paris

CYCLE INGÉNIEUR

## RAPPORT DE PROJET

*Bolide*

Mecatrol - Groupe 5

Samara Gabriela Cespedes Afanador  
Clair de Saint Méloir  
Marianne Pinault  
Clément Hervé  
Jingyi Liu

14 novembre 2024  
Paris

## Table des matières

<b>1 Fabrication</b>	<b>2</b>
1.1 Support du Moteur . . . . .	2
1.2 Roues . . . . .	2
1.3 Fixation de l'aimant . . . . .	3
1.4 Support de l'encodeur . . . . .	3
1.5 Fixation de la batterie . . . . .	3
1.6 Fixation du switch . . . . .	4
1.7 Support pour les charges amovibles . . . . .	4
1.8 Support du suiveur de ligne . . . . .	5
1.9 Châssis . . . . .	5
1.10 Conclusion . . . . .	7
<b>2 Assemblage</b>	<b>7</b>
<b>3 Configuration des moteurs</b>	<b>8</b>
3.1 Détails de la fonction <code>setMotorDutyCycle</code> . . . . .	8
<b>4 Interfaçage des encodeurs avec l'Arduino</b>	<b>8</b>
4.1 Mesures de position . . . . .	9
4.2 Vitesse en temps réel . . . . .	10
<b>5 Interfaçage du suiveur de ligne avec l'Arduino</b>	<b>11</b>
5.1 Méthode de réglage de sensibilité . . . . .	11
<b>6 Identification de paramètres</b>	<b>12</b>
6.1 Explication de la formule de la vitesse de translation et de la vitesse de rotation du véhicule . . . . .	13
6.2 Prise en compte des phénomènes de frottement et de viscosité . . . . .	14
<b>7 Résultats des tests</b>	<b>15</b>
7.1 Pas sur le sol . . . . .	15
7.1.1 $V_r$ en fonction de $U_r$ . . . . .	15
7.1.2 $V_l$ en fonction de $U_l$ . . . . .	15
7.1.3 $A_r$ en fonction de $U_r$ . . . . .	16
7.1.4 $A_l$ en fonction de $U_l$ . . . . .	16
7.1.5 $U_{plus}$ vs $\delta_u$ . . . . .	17
7.1.6 $U_{moins}$ vs $\delta_r$ . . . . .	18
7.2 Sur le sol . . . . .	18
7.2.1 $V_r$ en fonction de $U_r$ . . . . .	18
7.2.2 $V_l$ en fonction de $U_l$ . . . . .	19
7.2.3 $A_r$ en fonction de $U_r$ . . . . .	19
7.2.4 $A_l$ en fonction de $U_l$ . . . . .	20
7.2.5 $U_{plus}$ vs $\delta_u$ . . . . .	21
7.2.6 $U_{moins}$ vs $\delta_r$ . . . . .	21
<b>8 Résultats finaux</b>	<b>22</b>
8.1 Contrôleur pour le Système de Translation . . . . .	23
8.2 Contrôleur pour le Système de Rotation (Système Robuste) . . . . .	23
8.3 Différence entre la théorie et la pratique pour les valeurs de gain . . . . .	23
8.4 Contrôleur pour un Système plus Rapide . . . . .	23
8.5 Résultat Final de la Trajectoire . . . . .	23
<b>9 Résultats expérimentaux des performances individuelles des contrôleurs</b>	<b>25</b>
9.1 Performance du contrôleur PI pour le système de translation . . . . .	25
9.2 Performance du contrôleur PID pour le système de rotation . . . . .	26
<b>10 Code du parcours le plus rapide</b>	<b>27</b>
<b>11 Code identification paramètres</b>	<b>30</b>
<b>12 Code MatLab</b>	<b>35</b>

## 1 Fabrication

Une grande partie des choix faits lors de la fabrication a été détaillée dans le rapport de construction mécanique du 29 octobre 2024. Ce qui sera présenté dans cette section est donc un résumé de ce qui a été écrit dans ce rapport, ainsi que des compléments pour les pièces modifiées ou faites après le rapport.

### 1.1 Support du Moteur

La pièce finale est un support en forme de L, auquel sont ajoutées des équerres pour avoir le plus de solidité. Le moteur est fixé à l'aide des vis et placé grâce à l'axe moteur qui est inséré en force dans la pièce. Le MIP/MAP de cette pièce avec le chevet est fait à l'aide de 4 vis et de 2 pions de centrage.

Une piste d'amélioration pour cette pièce est de la faire légèrement moins grande. Il y a une hauteur de pièce au-dessus qui n'est pas nécessaire. La longueur pour le fixer au châssis pourrait également être réduite. Cela aurait permis de gagner du temps d'impression, d'économiser des matériaux comme le PLA et de gagner en place sur le châssis principal.

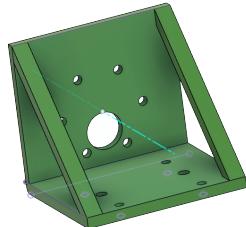


FIGURE 1 – Support du moteur

### 1.2 Roues

Les roues sont une reproduction de celles qui avaient été fournies en début de projet pour pouvoir commencer les tests. Le choix de prendre les mêmes dimensions a permis de reprendre les pneus fournis, pour avoir une bonne adhérence au sol. La pièce est attachée à l'axe moteur avec une vis de serrage qui est mise dans un insert sur une extrusion autour du centre.

Une autre modification a été de vider la pièce, pour ne plus avoir un disque plein, mais plutôt une roue avec 8 rayons. L'objectif est le même que celui qui a été évoqué pour une possibilité d'amélioration du support moteur, celui de gagner du temps d'impression et d'économiser des matériaux. Il y a aussi un avantage physique, valable aussi pour le support moteur, celui d'alléger le robot et donc de diminuer son inertie.



FIGURE 2 – Roues

### 1.3 Fixation de l'aimant

Cette pièce a pour but de maintenir l'aimant à la roue. C'est un disque dans lequel l'aimant est inséré en force puis fixé à la roue par 3 vis. L'avantage est que l'aimant n'a pas de possibilité de basculer d'un côté ou de l'autre, il est complètement immobile.

Cependant, cette pièce n'aurait pas forcément été nécessaire, il était possible de prévoir un espace similaire dans les roues puis d'insérer l'aimant en force. L'avantage de cette technique aurait été de ne pas faire de pièce supplémentaire, en économisant du temps et du PLA mais aussi d'avoir un peu moins d'assemblage à réaliser et d'utiliser moins de vis. Un autre inconvénient majeur à cette pièce est qu'une partie se situe entre l'encodeur et l'aimant, et que l'encodeur est donc un peu plus éloigné de l'aimant pour ne pas frotter lors de la rotation des roues.



FIGURE 3 – Fixation de l'aimant

### 1.4 Support de l'encodeur

Ce support a pour but de fixer et de placer l'encodeur en face de l'aimant. La pièce est en forme de U large vu du dessus, faisant le tour de la roue, le MIP/MAP est fait de chaque côté avec 2 pions de centrage ainsi que 4 vis. Il y a de plus des évidements de fait pour faire passer les câbles ainsi que certains composants électroniques. Des extrusions permettent de maintenir l'encodeur en place.

Pour ce qui est des possibilités d'amélioration, elles sont nombreuses sur cette pièce. Il aurait été possible de maintenir l'encodeur par une seule pâte passant au dessus de la roue. Une autre idée est de faire une conception complètement différente du châssis, ne situant pas les roues complètement à l'extérieur, mais dans un trou. Cela aurait permis de créer un support encodeur directement à la verticale, sur le châssis et non pas à l'extérieur.

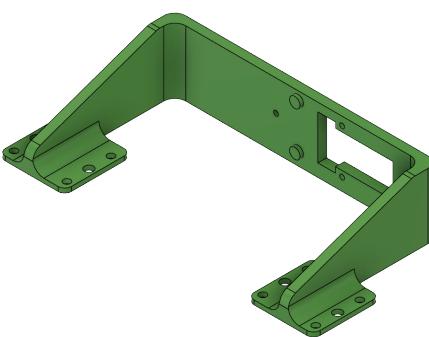


FIGURE 4 – Support de l'encodeur

### 1.5 Fixation de la batterie

Ce qui a été réalisé pour cette pièce est une sorte de boîte, pas complètement fermée qui bloque la pièce pour qu'elle ne puisse pas bouger selon un seul axe. La pièce est accrochée au châssis pour 4 vis et 2 pions de centrage.

La pièce finale est légèrement différente de ce qui a été présenté dans le rapport de construction mécanique, car nous avons finalement choisi de fixer le switch sur la batterie pour faciliter l'assemblage. Les raisons précises seront abordées dans la section de l'assemblage. La partie du dessus a donc été faite plus grande, ce qui n"était qu'une sorte de poutre de 5mm est devenu un plateau de 70 cm avec des inserts pour fixer le switch. Une piste d'amélioration est de faire une pièce légèrement plus économique en PLA en faisant un plateau sur le dessus plus petit. Une autre idée est de rajouter une poutre dans la longueur pour éviter une potentielle chute de la pièce par l'avant si l'on décidait de mettre cette pièce dans une orientation différente (à la verticale ou sous le châssis, par exemple).

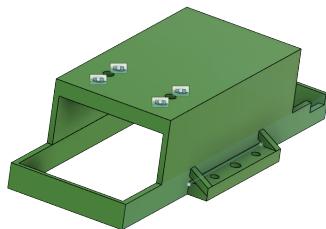


FIGURE 5 – Fixation de la batterie

## 1.6 Fixation du switch

Cette pièce a le même principe que la fixation de la batterie. Elle est cependant adaptée pour laisser passer les câbles et pour pouvoir placer le switch dedans, qui est assez particulier.

La pièce est fixée sur le dessus du support pour la batterie pour faciliter le câblage. Cette fixation est réalisée avec 2 pions de centrage et 4 vis.

Une possibilité d'évolution de cette pièce est de la supprimer en la remplaçant par un trou dans le châssis, qui, en étant juste de la forme d'un cercle, permettrait d'accrocher le switch. Il suffirait de le faire passer par dessous puis de raccrocher le dessus avec les boulons et le cache ON/OFF.

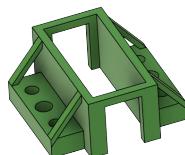


FIGURE 6 – Fixation du switch

## 1.7 Support pour les charges amovibles

Cette pièce est un support en forme de cylindre sur lequel les disques viennent s'empiler. La pièce est fixée juste au-dessus de la bille folle pour être plus stable. Le MIP/MAP est fait avec 2 pions de centrage et 4 vis. La forme très haute de cette pièce est faite pour respecter la condition : les poids doivent être à 10 cm du sol. Il y a un pilier central pour bien transférer le poids, tout en minimisant la quantité de plastique imprimé.

Cette pièce pourrait être améliorée en étant raccourcie en hauteur si le châssis était plus haut du sol. Beaucoup de plastique serait économisé, mais il y aurait aussi plus de stabilité, car le poids serait plus proche du châssis. En étant plus court, il y aurait aussi moins de contraintes sur le support.

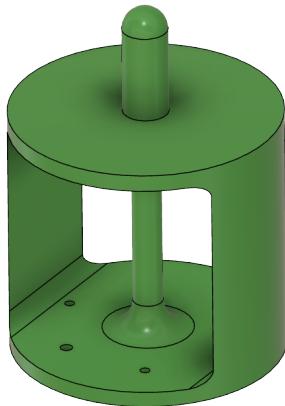


FIGURE 7 – Support pour les charges amovibles

## 1.8 Support du suiveur de ligne

L'idée de base était de fixer directement le capteur au châssis principal. Cependant, pour faciliter le travail pour la partie électronique, il était avantageux de le mettre un peu plus devant les roues, raison pour laquelle nous avons rajouté une pièce en bois qui a permis de prolonger le châssis vers l'avant. Il a ensuite fallu abaisser cette pièce pour être au plus proche du sol. C'est donc pour cela que nous avons rajouté des pièces en PLA qui se mettent entre les plaques et dans lesquelles nous avons vissé directement. Un évidement est prévu pour permettre à l'utilisateur de voir le capteur du dessus afin de voir les LEDs pour faciliter les tests et l'identification des paramètres.

Il aurait été possible de faire une unique pièce, plus simple, qui prenait directement en compte ces deux contraintes. Une autre idée est de placer les routes légèrement plus à l'arrière du robot.



FIGURE 8 – Support du suiveur de ligne

## 1.9 Châssis

Le principe de conception du châssis vise à garantir une connexion optimale des composants, un déplacement fluide du chariot sous différentes charges, ainsi qu'un fonctionnement optimal des capteurs dans des conditions idéales. Nous avons opté pour un chariot à un seul niveau afin de simplifier la structure tout en répondant à ces exigences. Pour garantir une fixation solide de tous les composants et connexions, les dimensions finales du châssis ont été définies à 220 mm x 275 mm. En tenant compte de l'ordre et de la manière de connecter les différents composants, nous avons placé les deux microcontrôleurs au centre du châssis. Les roues, les encodeurs magnétiques et leurs supports sont positionnés de chaque côté, tandis que les capteurs de suivi de ligne (line follow array) sont situés à l'avant du chariot.

Pour assurer la stabilité structurelle, nous avons installé la bille folle à l'arrière, formant ainsi une structure robuste à trois points d'appui. Les charges sont disposées au-dessus de la bille folle afin de minimiser leur impact

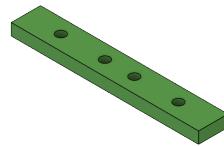


FIGURE 9 – Pièce pour diminuer la hauteur

sur le déplacement du chariot. Par ailleurs, la batterie, qui constitue le composant le plus lourd, a été placée au centre du châssis afin de garantir une stabilité optimale, que le chariot soit chargé ou non.

La position des capteurs constitue également un aspect crucial de la conception du châssis. Pour les encodeurs magnétiques, la distance optimale de travail est comprise entre 3 et 5 mm par rapport aux roues. Cela permet de collecter des données précises sans entraver la rotation des roues. En fonction des dimensions des structures de support du moteur et des capteurs, nous avons défini une distance de 11 mm et 8 mm par rapport au bord du châssis. Concernant les capteurs de suivi de ligne, ils ont été placés sous le châssis pour réduire au maximum l'influence des variations de lumière ambiante sur leur sensibilité. Cette disposition garantit une détection fiable tout en préservant la précision des données collectées.

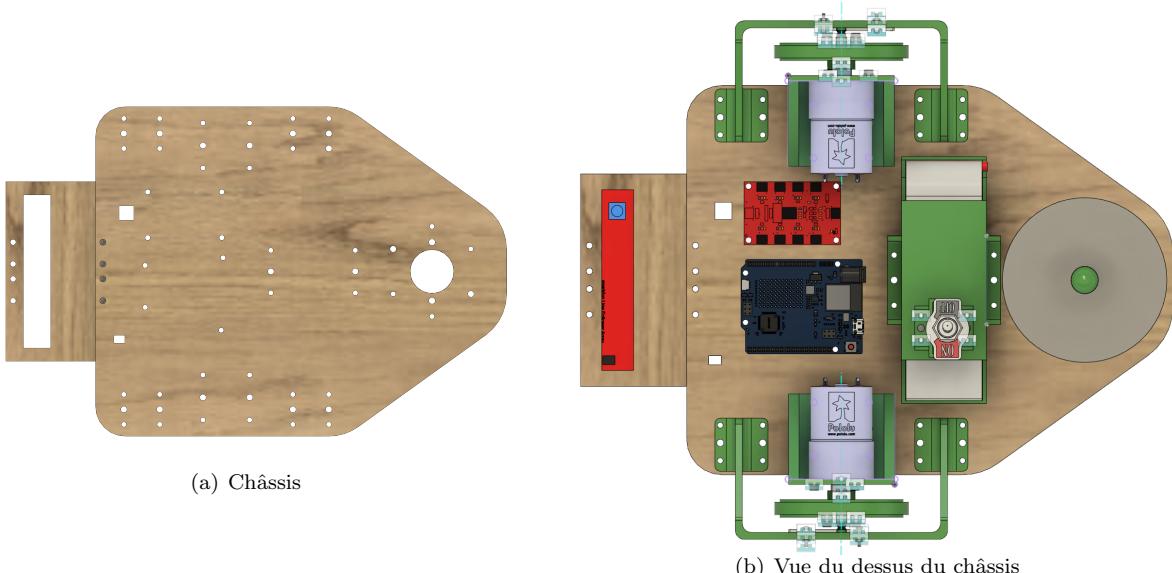


FIGURE 10 – Schéma du châssis et disposition des composants

En fonction du choix de la taille des roues et de la position du moteur, nous avons constaté que la garde au sol normale du châssis est d'environ 17 mm, tandis que la hauteur de la bille folle atteint environ 19 mm. Cette légère différence de hauteur, combinée à la longueur totale du châssis de 275 mm, n'a qu'un impact négligeable sur la stabilité globale du chariot, même lors de son déplacement. Après analyse, nous avons donc décidé de fixer directement la bille folle sous le châssis, sans ajout de structures supplémentaires.

Cette configuration simplifie la conception tout en maintenant une répartition stable des forces. De plus, la position centrale des autres composants et la répartition homogène des charges garantissent un déplacement fluide, même avec cette légère différence de hauteur.

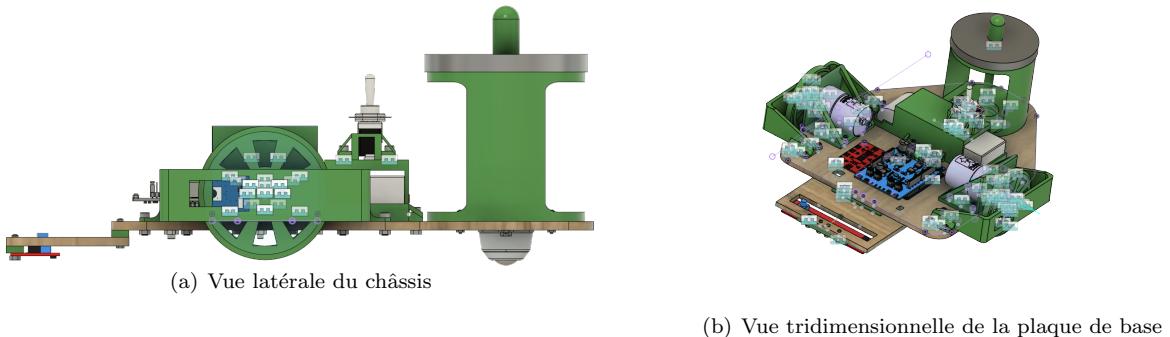


FIGURE 11 – Vue multidimensionnelle du châssis

## 1.10 Conclusion

Le robot que nous avons conçu respecte bien toutes les contraintes du projet et permet bien de suivre la ligne du circuit. Cependant, il y a beaucoup de pistes d'amélioration possibles pour le robot. Certaines d'entre elles auraient pu être mises en place en ayant plus de recul sur la conception globale ou en prenant en compte absolument toutes les contraintes dès le début, comme la position du capteur ou la hauteur des disques. Une des possibilités serait de reprendre la position actuelle, mais en inversant certaines pièces, par exemple en mettant les moteurs sous le châssis et non pas dessus, tout en faisant un support très haut pour le suiveur de ligne et pour la bille folle. Cela permettrait aussi de mettre la batterie en dessous et donc de faire un châssis plus petit, car il y aurait des pièces des deux côtés. Le support des charges amovibles serait alors moins haut également.

## 2 Assemblage

Pour ce qui est de l'assemblage, certaines choses avaient été pensées dès la conception du robot sur Fusion. Beaucoup de pièces étant fixées sur le chevet, il n'y avait pas de problème d'ordre d'assemblage ou de désassemblage pour la plupart d'entre elles.

En revanche, il n'y avait pas de possibilité de démonter les roues sans enlever le support de l'encodeur, ce qui était un problème, car les roues ont dû être refixées plusieurs fois. En effet, l'insert qui était sur le côté ne tenait pas bien et il a fallu plusieurs fois le remettre en refaisant fondre le plastique. Une autre solution aurait pu être de refaire des roues avec un diamètre plus petit, mais elle était plus coûteuse en temps et nécessitait de consommer à nouveau du plastique.

Un autre problème d'assemblage est la position de la batterie vis-à-vis des composants électroniques. Le câble qui se branchait entre la carte Arduino et l'ordinateur, nécessitait une place sur le côté de la carte, à l'endroit même où avait été choisi de mettre la batterie. La solution choisie fut celle de fixer la batterie uniquement au dernier moment, en gardant donc une batterie qui n'était pas fixée au support durant toute la phase de test et de conception.

Par ailleurs, les câbles de la batterie étant épais et très peu souples, nous avons dû revoir la position du switch qui était censé être lui aussi sur le chevet à la base. La position initiale n'était pas envisageable car les câbles bloquaient cette possibilité.

En ce qui concerne les autres câbles, plus petits et souples, nous n'avons eu aucune contrainte excepté la longueur. Nous avons donc fait les raccordements nécessaires pour rallonger ce qui devait l'être. Un espace avait été prévu pour ces câbles dans le support encodeur et dans la plaque du support du suiveur de ligne, donc il n'y avait aucun problème supplémentaire pour les fixer aux composants.

Pour ce qui est de la fixation des composants, la stratégie choisie est la même pour tous, en prenant 4 vis et 2 pions de centrage. Cela permettrait d'être fixe et de ne pas bouger. Certaines pièces, comme la fixation de la batterie, n'ont pas nécessité les pions, car il n'y avait pas besoin d'être précis et complètement immobile.

Les pions de centrage ont tous été mis en force dans les pièces en plastique. Cependant, pour le chevet, la découpe laser dans le bois n'étant pas aussi précise, les trous étaient parfois légèrement trop grands et la pièce pouvait légèrement bouger, sans que cela pose de problème pour le robot.

### 3 Configuration des moteurs

Nous avons choisi cette configuration 12 après avoir vérifié que les deux moteurs se déplaçaient vers l'avant lorsqu'une entrée de commande  $U$  positive était appliquée, et inversement, ils se déplaçaient en arrière pour une entrée  $U$  négative. Pour générer le signal de commande, nous avons utilisé la fonction fournie par la bibliothèque `Mecatronics`, `setMotorDutyCycle(Ul, Ur)`, qui reçoit comme paramètres les valeurs de duty cycle comprises entre  $-1$  et  $1$  pour chaque moteur, où  $Ul$  correspond au moteur gauche et  $Ur$  au moteur droit.

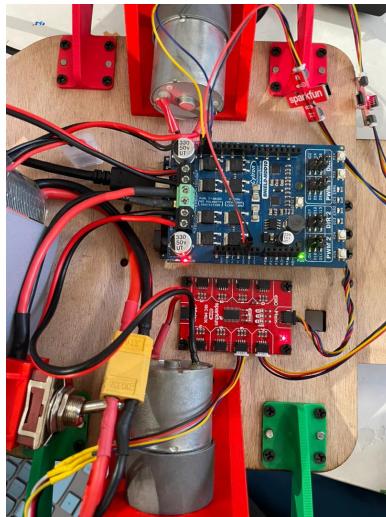


FIGURE 12 – Emplacement câbles des moteurs

#### 3.1 Détails de la fonction `setMotorDutyCycle`

La fonction `setMotorDutyCycle` contrôle la vitesse et la direction des moteurs en utilisant la modulation par largeur d'impulsion (PWM) avec les minuteries GPT2 et GPT7. Elle détermine la direction de rotation à l'aide de broches numériques, en fonction du signe du duty cycle, et calcule la valeur de comparaison pour les minuteries en adaptant le cycle de travail à une plage permise allant de  $0$  à  $GTPR = 2399$ . La fréquence d'horloge utilisée est de  $48\text{ MHz}$ . Par conséquent, la fréquence du signal PWM est définie comme  $f = \frac{48000000}{2399+1} = 20\text{ kHz}$ , c'est-à-dire, une période de  $0.05\text{ ms}$ .

Le duty cycle est limité à des valeurs admissibles avant d'être assigné aux registres des minuteries correspondants, qui génèrent le signal PWM pour chaque moteur.

```

1 void setMotorDutyCycle(float const& leftMotorDC, float const& rightMotorDC)
2 {
3     // Set direction
4     digitalWrite(DIRECTION_MOTOR_RIGHT, rightMotorDC > 0);
5     digitalWrite(DIRECTION_MOTOR_LEFT, leftMotorDC > 0);
6     // Set duty cycle: the timer counts up to R_GPT2->GTPR.
7     int counterValue = floor(abs(rightMotorDC) * R_GPT2->GTPR);
8     if (counterValue > R_GPT2->GTPR)
9         counterValue = R_GPT2->GTPR;
10    R_GPT2->GTCCR[0] = counterValue;
11
12    counterValue = floor(abs(leftMotorDC) * R_GPT7->GTPR);
13    if (counterValue > R_GPT7->GTPR)
14        counterValue = R_GPT7->GTPR;
15    R_GPT7->GTCCR[1] = counterValue;
16 }
```

Listing 1 – Fonction `setMotorDutyCycle` pour envoyer un PWM.

### 4 Interfaçage des encodeurs avec l'Arduino

L'encodeur rotatif Grove AS5600, choisi comme capteur d'angle de rotation des moteurs, utilise l'effet Hall pour détecter les variations de la direction d'un champ magnétique. Il s'agit d'un codeur absolu simple offrant une résolution de 12 bits, permettant de mesurer jusqu'à 4096 positions par tour. Il dispose de deux sorties configurables : via l'interface I<sup>2</sup>C pour les données brutes ou en signal PWM/analogique via une broche dédiée

[2]. Dans notre cas, la communication I2C a été choisie. Cependant, étant fixe la sortie I2C il n'est pas possible d'utiliser deux codeurs sur le même réseau sans un multiplexeur I2C. De plus, un câble d'interface a été implémenté car son connecteur est de type Grove et non QWIIC.

Pour répondre aux contraintes mécaniques ou logicielles, le module intègre un switch permettant d'inverser l'interprétation des signaux de rotation captés, soit au sens horaire (CW) ou antihoraire (CCW) sans modification de l'installation (voir image 13). Par défaut, une rotation horaire est interprétée comme positive. Lorsque le switch est activé, cette même rotation est considérée comme négative, inversant ainsi le sens perçu. Ainsi, une fois les encodeurs assemblés dans leur position de fonctionnement, ce switch a été ajusté sur chaque encodeur pour correspondre à une lecture positive lorsque les moteurs tournent dans le sens antihoraire, c'est-à-dire vers l'avant dans le système global de la voiture, et à une lecture négative lorsqu'ils tournent dans le sens horaire.

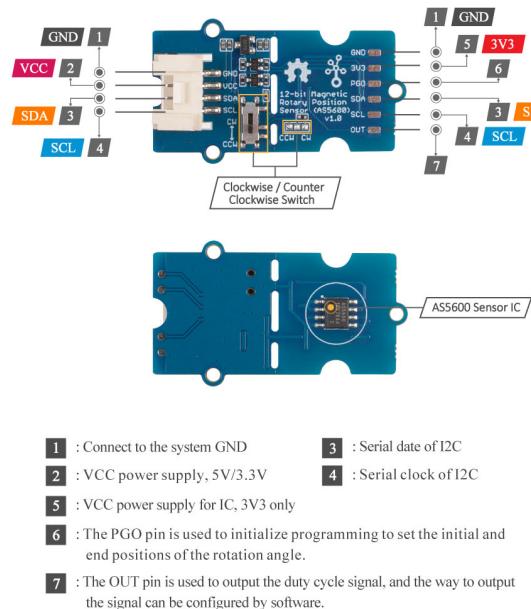


FIGURE 13 – L'encodeur rotatif Grove AS5600

#### 4.1 Mesures de position

Lors de l'obtention des mesures de position des encodeurs, nous avons constaté des variations brusques et un bruit important dans les données brutes. Pour améliorer la qualité du signal, un filtre passe-bas de type moyenne exponentielle, qui consiste, en termes généraux, à lisser le signal en donnant un poids plus important aux valeurs récentes tout en conservant une certaine mémoire des valeurs passées. Ce filtre agit comme un passe-bas, a été appliqué, suivant l'équation :

$$V_i = \epsilon \cdot V_{\text{encoder}} + (1 - \epsilon) \cdot V_{i-1}$$

Ici,  $V_i$  représente la valeur filtrée à l'instant  $i$ ,  $V_{\text{encoder}}$  la valeur brute mesurée par l'encodeur, et  $\epsilon$  est le facteur d'atténuation.

Nous avons choisi la valeur de  $\epsilon$  égale à 0,8 car, en appliquant ce filtre passe-bas, le temps de stabilisation est de 0,15 seconde, et le signal en régime permanent est moins oscillatoire, améliorant le lissage du signal. La valeur de  $\epsilon$  peut varier entre 0 et 1 ; nous avons sélectionné 0,8 pour ne pas altérer les données originales ni affecter le temps de stabilisation. Une valeur plus faible rendrait le signal plus lisse mais avec un temps de stabilisation plus long. Par exemple, avec une valeur de 0,0015, nous avons observé ce phénomène, tandis qu'avec des valeurs supérieures à 0,95, le signal n'était pas suffisamment filtré.

Pour garantir que les mesures de vitesse étaient cohérentes et adaptées aux performances attendues, nous avons d'abord analysé les spécifications maximales du moteur trouvés auprès du datasheet [1], avec charge 330 RPM  $\approx$  34.5 rad/s et sans charge 280 RPM  $\approx$  29.3 rad/s, ces valeurs correspondent à un fonctionnement à une tension maximale de 12V. Nous avons ensuite comparé la sortie de notre fonction de vitesse (expliquée dans la section suivante) avec la fonction 'angularSpeed' fournie par le professeur. Les résultats montrent que les deux fonctions donnent des valeurs très proches. Cependant, notre implémentation produit un signal moins bruité et plus lisse en régime permanent réduisant les fluctuations rapides dans le signal d'entrée mais toujours conforme à la réalité physique d'une vitesse d'environ 30 rad/s.

## 4.2 Vitesse en temps réel

Ensuite, des tests supplémentaires ont été réalisés sur le signal de vitesse en utilisant un second filtre qui réduit les pics dans le signal, comme on peut le voir dans les images suivantes pour différents types de signaux, avec des comparaisons entre chaque moteur. Les valeurs obtenues montrent une meilleure condition des données avec un double filtrage. Le deuxième filtre utilisé est appelé filtre de moyenne mobile, qui consiste à calculer la moyenne des valeurs sur une fenêtre glissante pour lisser les variations rapides et réduire le bruit dans le signal.

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]$$

$y[n]$  : la valeur filtrée du signal à l'instant  $n$ . Elle correspond au résultat du filtre obtenu en prenant la moyenne des  $M$  dernières valeurs de  $x[n]$ .

$M$  : la longueur de la fenêtre du filtre. Il s'agit du nombre de valeurs utilisées pour calculer la moyenne. Une fenêtre plus grande ( $M$  élevé) lisse davantage les variations rapides, mais introduit un retard dans la réponse du filtre.

$x[n]$  : la valeur d'entrée du signal à l'instant  $n$ . Ce sont les données brutes ou non filtrées.

Il est important de noter que les signaux d'entrée représentés dans les graphiques  $U_r$  et  $U_l$  ont été augmentés par un facteur de 1875 (soit 1500/0,8) pour que la forme des signaux soit plus visible, voir figures 14 et 16. Cela signifie que les signaux ne sont pas à l'échelle réelle, et les valeurs maximales de ces signaux se situent entre -0,8 et 0,8 pour ces essais. Les résultats sont présentés ci-après :

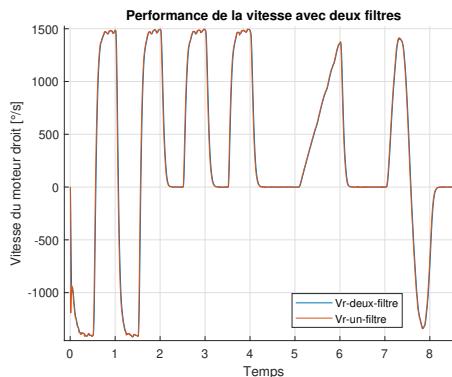


FIGURE 14 – Vitesse moteur droit en temps réel

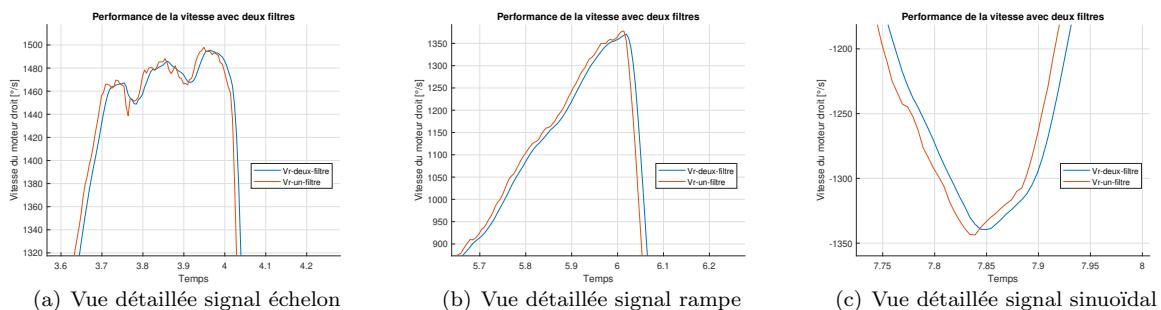


FIGURE 15 – Vues détaillées

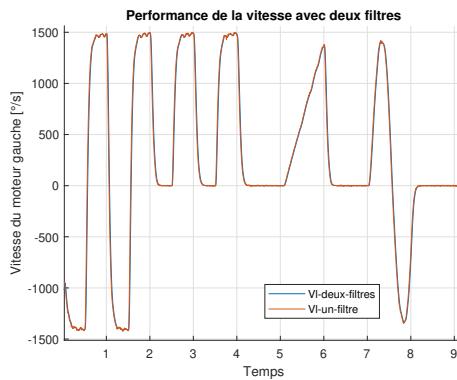


FIGURE 16 – Vitesse moteur gauche en temps réel

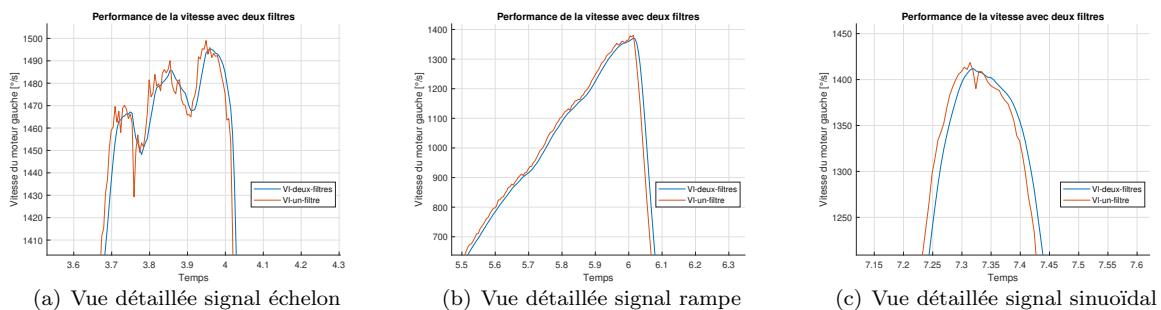


FIGURE 17 – Vues détaillées

## 5 Interfaçage du suiveur de ligne avec l’Arduino

Nous avons utilisé le code fourni pour obtenir une mesure précise du suivi de ligne, en particulier grâce à la fonction `.getPosition()`. Cette fonction, qui ne prend aucun argument, renvoie un nombre signé de 8 bits, compris entre  $-127$  et  $127$ . Ce résultat représente la moyenne des points détectés par les capteurs de la barre.

Par exemple :

- Si les deux capteurs centraux (`b4` et `b3`) détectent une ligne, la valeur moyenne sera proche de  $0$ , indiquant une position centrée.
- Si les quatre capteurs de gauche (`b7` à `b4`) détectent la ligne, la moyenne sera décalée vers la gauche, avec une valeur d'environ  $-79$ .
- Si seul le capteur le plus à gauche (`b7`) détecte la ligne, la valeur retournée sera  $-127$ .

Pour faciliter l'interprétation des données, l'axe sériographié sur la barre du capteur peut être utilisé comme référence visuelle.

Il faut savoir : Si `InvertBits` ne correspond pas aux couleurs de la ligne ou du champ, le résultat de `.getPosition()` sera incorrect. Cela s'explique par le fait que plusieurs positions détectées sont moyennées pour obtenir un vecteur. Une configuration inversée entraîne la détection de toutes les positions, ce qui donne une moyenne proche de zéro (centre).

Enfin, nous avons appliqué une mise à l'échelle de la valeur obtenue pour la convertir en sa correspondance en unité internationale, en tenant compte de la longueur totale du suiveur 105 mm. Voir la partie du code 2.

```

1 void position() {
2     pos = (mySensorBar.getPosition() * 105 / 254);
3 }
```

Listing 2 – Fonction `position()` pour obtenir des données du suiveur de ligne .

### 5.1 Méthode de réglage de sensibilité

Pour configurer la sensibilité du capteur de ligne sur le parcours, nous avons suivi les étapes suivantes :

1. Réduire la luminosité jusqu'à ce que les zones claires cessent de s'illuminer.
2. Augmenter la luminosité jusqu'à ce que les zones sombres commencent à être détectées.
3. Ajuster la luminosité à une valeur intermédiaire entre ces deux points.

## 6 Identification de paramètres

Pour l'identification des paramètres, nous avons mesuré les vitesses et les positions angulaires de chaque moteur en réponse à différents types de signaux de tension. Les tests ont été effectués selon l'ordre suivant : échelon de -1 à 1, échelon de 1 à 0, rampe, et signal sinusoïdal (Voir graphiques 18). La période de chaque signal de test a été fixée à 1 seconde.

Les réponses du système ont été obtenues à l'aide d'un code Arduino 3 et d'un code MatLab 4, qui fonctionne comme suit :

1. **Initialisation de la télémétrie** : Cette étape permet de connecter le module WiFi et de transmettre les données vers MATLAB.
2. **Acquisition des données** : La méthode `.getCumulativePosition()` est utilisée pour récupérer l'angle cumulatif de l'encodeur. Cet angle est ensuite transmis à la fonction `.trouverVitesse(angle)`.
3. **Traitement des données brutes** : Dans la fonction `.trouverVitesse(angle)`, la valeur brute de l'angle est d'abord filtrée à l'aide d'un filtre de moyenne exponentielle.
4. **Calcul de la vitesse** : La vitesse est calculée comme la dérivée de la position, en utilisant un intervalle de temps ( $\Delta t$ ) de 5 ms.
5. **Double filtrage** : Pour finaliser l'étape de conditionnement de signaux, une fois la vitesse calculée, un second filtre passe-bas, basé sur une moyenne mobile, est appliqué. Ce filtre permet de stabiliser davantage la mesure de la vitesse en lissant les variations rapides.
6. **Calcul des commandes de chaque moteur** : En utilisant la fonction `.fonction()`, nous avons créé une succession de différents types de signaux de commande. À cette étape, le code fournit la valeur du signal en temps réel.
7. **Obtention des variables d'état du système** : Une fois les vitesses de chaque moteur calculées, nous déterminons les variables d'état estimées.
8. **Télémétrie** : Enfin, nous enregistrons les données susceptibles d'être envoyées vers MATLAB afin qu'elles puissent être traitées et analysées.
9. **Activation des moteurs** : Finalement, les commandes pour les moteurs sont envoyées afin qu'ils rouent à la vitesse désirée.

Ce processus garantit des mesures à prendre pour interpréter, essentielles pour l'identification des paramètres et la modélisation du système.

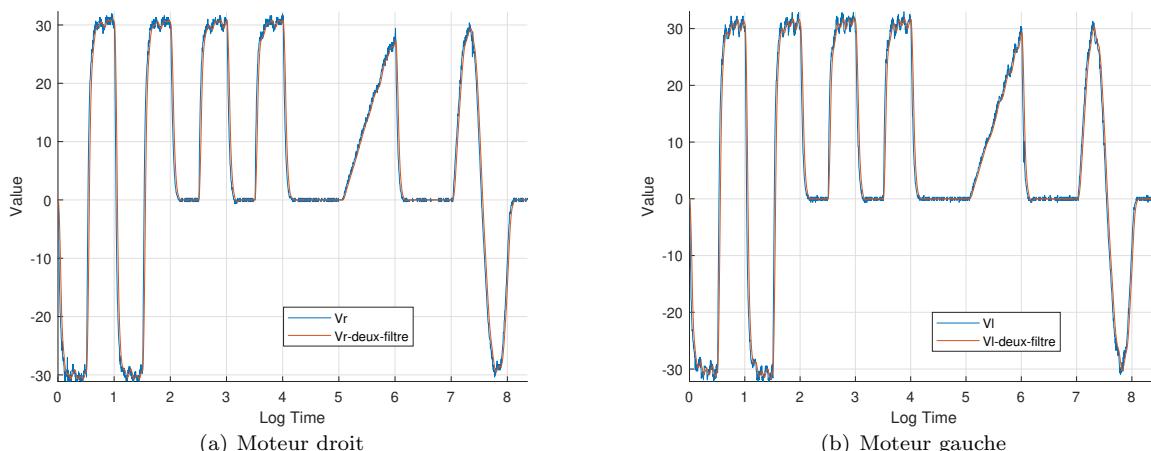


FIGURE 18 – Resultats via télémétrie

Étant donné que dans l'identification du modèle dynamique par rapport au système réel, certaines valeurs, telles que les coefficients de frottement, les inerties, etc., sont parfois perdues ou mal connues, il est essentiel de réaliser une identification des paramètres pour mieux comprendre notre système réel. Ce processus permet d'observer la réponse intégrée du moteur, du réducteur et de la roue. Il offre également des informations précieuses sur différents paramètres d'accouplement, ainsi que sur les propriétés électriques et d'inertie, grâce aux mesures obtenues via l'encodeur magnétique.

## 6.1 Explication de la formule de la vitesse de translation et de la vitesse de rotation du véhicule

Paramétrisation de la vitesse de translation du véhicule et de sa vitesse de rotation. On pose le vecteur d'état suivant :

$$X = \begin{bmatrix} x \\ y \\ u \\ \psi \\ \dot{\psi} = r \\ i_+ \\ i_- \end{bmatrix} \quad (1)$$

On a également le vecteur de commande U :

$$U = \begin{bmatrix} U_+ \\ U_- \end{bmatrix} \quad (2)$$

On trouve un état d'équilibre pour les équations de la dynamique suivantes :

$$\dot{u} = \frac{1}{\left(M + 2\frac{I_y^w}{\rho^2}\right)} \left( \frac{ki_+}{\rho} - m_b dr^2 \right) = \alpha_1 i_+ - \alpha_2 r^2 \quad (3)$$

$$\dot{r} = \frac{1}{I_\psi + m_b d^2} \left( \frac{kl}{2\rho} i_- + m_b dur \right) = \alpha_3 i_- + \alpha_4 ur \quad (4)$$

$$\frac{di_+}{dt} = \frac{U_+}{L} - R \frac{i_+}{L} - \frac{2k}{L\rho} u \quad (5)$$

Qui est le suivant :

$$\frac{di_-}{dt} = \frac{U_-}{L} - R \frac{i_-}{L} + \frac{kl}{L\rho} r \quad (6)$$

$$\bar{X} = \begin{bmatrix} \cos(\bar{\psi}) \bar{u} \\ \sin(\bar{\psi}) \bar{u} \\ \bar{u} \\ \bar{\psi} \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (7)$$

$$\bar{U} = \begin{bmatrix} \bar{U}_+ \\ 0 \end{bmatrix} \quad (8)$$

On va linéariser le système autour de cet état d'équilibre. En posant de nouveaux vecteurs d'états on peut traiter indépendamment la rotation et la translation. On a alors deux équations de la forme,

$$\dot{X}_{1,2} = A_{1,2} X_{1,2} + B_{1,2} U \quad (9)$$

Avec pour la translation :

$$X_1 = \begin{bmatrix} u \\ x \\ i_+ \end{bmatrix} \quad (10)$$

$$A_1 = \begin{bmatrix} 0 & 0 & \alpha_1 \\ 1 & 0 & 0 \\ -\frac{2k}{\rho L} & 0 & -\frac{R}{L} \end{bmatrix} \quad (11)$$

$$B_1 = \begin{bmatrix} \frac{1}{L} \\ 0 \\ 0 \end{bmatrix} \quad (12)$$

Et pour la rotation :

$$X_2 = \begin{bmatrix} y \\ \dot{\psi} \\ \ddot{\psi} \\ i_- \end{bmatrix} \quad (13)$$

$$A_2 = \begin{bmatrix} 0 & \bar{u} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \alpha_2 \bar{u} & \alpha_3 \\ 0 & -\frac{kl}{\rho L} & 0 & -\frac{R}{L} \end{bmatrix} \quad (14)$$

$$B_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} \quad (15)$$

Or on trouve des valeurs propres très éloignées de zéro qui nous permettent de simplifier le système par perturbations singulières. En effet les constantes de temps électriques sont négligeables par rapport aux constantes de temps mécaniques d'où l'approximation  $\dot{i}_+ = 0$  et  $\dot{i}_- = 0$ .

À la fin de la réduction par perturbations singulières, nos systèmes de vitesses prennent la forme d'une fonction de transfert avec un pôle et zéro zéros, comme indiqué ci-dessous :

$$\hat{u}_v(s) = \frac{\alpha}{s + \beta} \hat{U}^+(s)$$

$$\hat{r}_v(s) = \frac{\gamma}{s + \zeta} \hat{U}^-(s)$$

On implémente alors sur Matlab un correcteur PI en translation et un correcteur PID en rotation.

On obtient par le calcul, pour obtenir les polynômes caractéristiques idéaux pour le système les valeurs théoriques de gain suivantes :

Pour la translation :

$$K_i = \frac{Rp_o^2}{\alpha_1}; K_p = \frac{2\xi Rp_0 - \frac{2k\alpha_1}{\rho}}{\alpha_1} \quad (16)$$

Pour la rotation :

$$K_{pr} = \frac{R\sigma^3}{\alpha_3 \bar{u}}; K_{dr} = \frac{R\sqrt{6}\sigma^2}{\alpha_3 \bar{u}}; K_{ddr} = \frac{R}{\alpha_3} \left( \frac{\sqrt{6}\sigma}{\bar{u}} + \alpha_4 \bar{u} + \frac{kl\alpha_3}{R\rho} \right) \quad (17)$$

Ce qui donne :

$$K_{pt} = 8.283; K_{it} = 271.21; K_{pr} = 2.774; K_{dr} = 4.5299; K_{ddr} = 2.097 \quad (18)$$

## 6.2 Prise en compte des phénomènes de frottement et de viscosité

Les réducteurs des moteurs induisent des phénomènes de frottements secs et visqueux ainsi qu'une modification de l'inertie totale du bolide. Ceci modifie les équations du bolide de sorte qu'il faut identifier de façon expérimentale les valeurs des coefficients des formules écrites précédemment :

$$\hat{u}_v(s) = \frac{\alpha}{s + \beta} \hat{U}^+(s)$$

$$\hat{r}_v(s) = \frac{\gamma}{s + \zeta} \hat{U}^-(s)$$

En utilisant la Control Toolbox de Matlab, on peut à partir de données d'entrée et de sortie évaluer les valeurs de ces coefficients.

On trouve ainsi les valeurs finales des coefficients :

$$K_{pt} = 2.4906; K_{it} = 106.8; K_{pr} = 0.072456; K_{dr} = 0.11832; K_{ddr} = 0.30076 \quad (19)$$

On note notamment pour les gains du correcteur en rotation une différence d'ordre de grandeur de l'ordre de  $10^1$  ce qui montre l'influence très importante des phénomènes visqueux.

## 7 Resultats des tests

### 7.1 Pas sur le sol

Ci-dessous sont présentés les résultats de notre système lorsqu'il fonctionne sans être en contact avec une surface, c'est-à-dire sans l'intervention de facteurs physiques tels que les glissements, les frottements, etc.

#### 7.1.1 V<sub>r</sub> en fonction de U<sub>r</sub>

Test avec une valeur de  $\epsilon$  du filtre de 0.1, une période du signal de 5 secondes et -0.3 PWM pour le moteur droit. Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{374.6}{s + 11.87} \hat{U}(s)$$

Avec une estimation des données : 77.32%

Test avec  $\epsilon = 1$ , c'est-à-dire sans filtre exponentiel et 0.8 PWM . Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{735.8}{s + 24.05} \hat{U}(s)$$

Avec une estimation des données : 93.69%

Entrées de plusieurs signaux avec  $\epsilon = 0.8$  et 0.5 PWM pour le moteur droit . Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{449.8}{s + 14.52} \hat{U}(s)$$

Avec une estimation des données : 89.61%

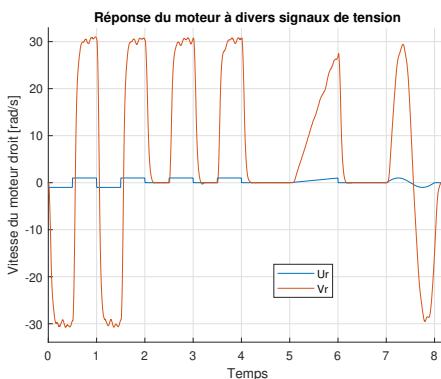


FIGURE 19 – Réponse du moteur droit à divers signaux de tension

#### 7.1.2 V<sub>l</sub> en fonction de U<sub>l</sub>

Test avec une valeur de  $\epsilon$  du filtre de 0.1, une période du signal de 5 secondes et 0.3 PWM pour le moteur gauche. Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{377.8}{s + 12.64} \hat{U}(s)$$

Avec une estimation des données : 94.64%

Test avec  $\epsilon = 1$ , c'est-à-dire sans filtre exponentiel et 0.8 PWM . Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{759.4}{s + 24.51} \hat{U}(s)$$

Avec une estimation des données : 94.22%

Entrées de plusieurs signaux avec  $\epsilon = 0.8$  et 0.5 PWM pour le moteur droit . Fonction de transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{470.9}{s + 14.7} \hat{U}(s)$$

Avec une estimation des données : 89.91%

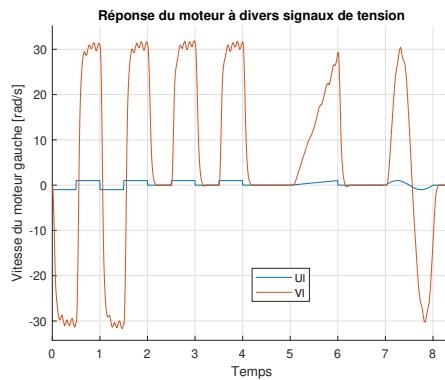


FIGURE 20 – Réponse du moteur gauche à divers signaux de tension

### 7.1.3 Ar en fonction de Ur

Position angulaire du moteur droit avec une entrée de 0.3, une valeur de  $\epsilon$  du filtre de 0.1, une période du signal de 5 secondes. Fonction de transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{388.9}{s^2 + 12.31s + 0.00169} \hat{U}(s)$$

Avec une estimation des données : 99.77%

Test avec  $\epsilon = 1$ , c'est-à-dire sans filtre exponentiel et 0.8 PWM . Fonction de transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{785.5}{s^2 + 25.86s + 0.03061} \hat{U}(s)$$

Avec une estimation des données : 99.69%

Entrées de plusieurs signaux avec  $\epsilon = 0.8$  et 0.5 PWM pour le moteur droit . Fonction de transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{717.1}{s^2 + 24.07s + 0.06272} \hat{U}(s)$$

Avec une estimation des données : 99.4%

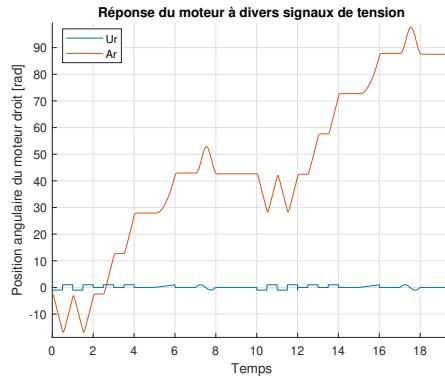


FIGURE 21 – Réponse position angulaire moteur droit

### 7.1.4 Al en fonction de Ul

Position angulaire du moteur gauche avec une entrée de 0.3, une valeur de  $\epsilon$  du filtre de 0.1, une période du signal de 5 secondes. Fonction de transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{395.1}{s^2 + 13.25s + 0.003678} \hat{U}(s)$$

Avec une estimation des données : 99.89%

Test avec  $\epsilon = 1$ , c'est-à-dire sans filtre exponentiel et 0.8 PWM . Fonction de transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{783.1}{s^2 + 25.53s + 0.01332} \hat{U}(s)$$

Avec une estimation des données : 99.75%

Entrées de plusieurs signaux avec  $\epsilon = 0.8$  et 0.5 PWM pour le moteur gauche . Fonction de transfert obtenue :

$$\hat{a}_w(s) = \frac{769}{s^2 + 25.01s + 0.02283} \hat{U}(s)$$

Avec une estimation des données : 99.74%

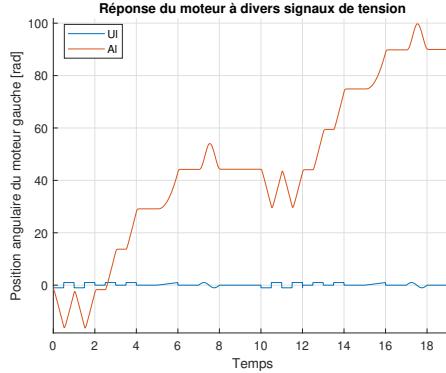


FIGURE 22 – Réponse position angulaire moteur gauche

### 7.1.5 $U_{plus}$ vs $\delta_u$

Les tests suivants ont été réalisés en appliquant des tensions positives aux moteurs, sans que le châssis du véhicule ne soit en contact avec le sol. Ainsi, on analyse la relation entre la somme des tensions des moteurs droit ( $U_r$ ) et gauche ( $U_l$ ), définie comme la variable  $U^+$ , et la vitesse de translation des moteurs, représentée par  $\hat{u}_v(s)$ . Les résultats obtenus sont les suivants :

Pour une tension de 0,3 sur les deux moteurs :

$$\hat{u}_v(s) = \frac{57,53}{s + 12,03} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 86,15 %

En augmentant la tension à 0,5 sur les deux moteurs :

$$\hat{u}_v(s) = \frac{8,856}{s + 14,01} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 89,58 %

En fixant la tension à 1 :

$$\hat{u}_v(s) = \frac{9,034}{s + 14,92} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 91,45 %

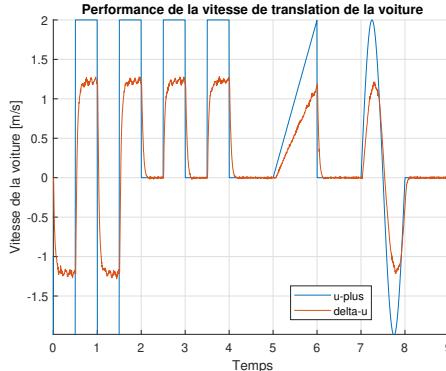


FIGURE 23 – Vitesse de la voiture en avance tout droit sans sol

### 7.1.6 $U_{\text{moins}}$ vs $\delta_r$

Dans ce système, la relation entre  $U_{\text{moins}}$  et  $\Delta_r$  a été analysée pour tensions avec de signe positives. La fonction de transfert obtenue est nulle, et le graphique correspondant met en évidence les pics de bruit présents dans les mesures ainsi qu'une inégalité des vitesses entre les deux moteurs, malgré l'application de tensions égales 24. Ce phénomène s'explique par les différences structurelles physiques entre les deux moteurs, qui influencent leur comportement dynamique et leur réponse dans le système.

$$\hat{r}_v(s) = 0\hat{U}^-(s)$$

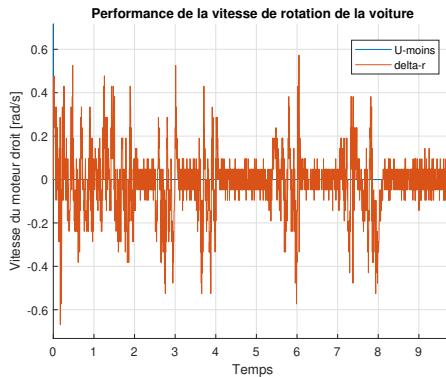


FIGURE 24 – Vitesse de rotation nulle

## 7.2 Sur le sol

Sur le sol, une expérimentation effective a été réalisée avec plusieurs types de signaux d'entrée. Cependant, comme le véhicule avait tendance à reculer lorsque les signaux prenaient une valeur négative, il a été décidé d'utiliser uniquement un signal d'entrée en échelon. Cette approche a permis d'analyser et de caractériser les sorties  $\delta_u$  et  $\delta_r$ .

### 7.2.1 $V_r$ en fonction de $U_r$

Une série de tests a été effectuée avec différents signaux, en fixant une valeur de  $\epsilon = 0.8$  et une valeur maximale de PWM de 0,3, la fonction transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{122.3}{s + 3.581}\hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 56.77% Dans les tests suivants, l'entrée de tension a été modifiée pour correspondre à un signal en échelon ne contenant que des valeurs positives, ce qui permettait au véhicule d'avancer uniquement. En fixant une valeur maximale de PWM à 0,3, la fonction de transfert obtenue est la suivante :

$$\hat{\Omega}_w(s) = \frac{240.9}{s + 8.045}\hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 79.52%

Dans le test suivant, une valeur de 0,5 PWM a été appliquée à chaque moteur.

$$\hat{\Omega}_w(s) = \frac{112.3}{s + 3.669}\hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 66.5%

Dans le cadre du test suivant, l'objectif était d'évaluer le comportement du véhicule en mode de rotation. Pour cela, des valeurs de PWM de -1 pour le moteur gauche et 1 pour le moteur droit ont été utilisées.

$$\hat{\Omega}_w(s) = \frac{608.5}{s + 22.45}\hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 92.92%

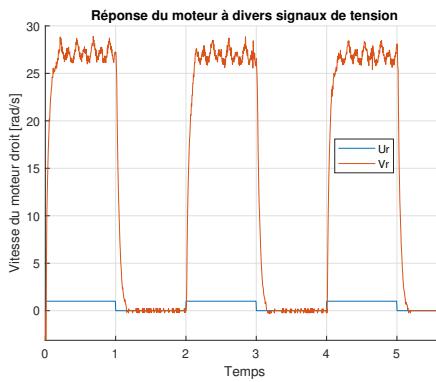


FIGURE 25 – Test sur sol, réponse du moteur

### 7.2.2 VI en fonction de UI

Une série de tests a été effectuée avec différents signaux, en fixant une valeur de  $\epsilon = 0.8$  et une valeur maximale de PWM de 0,3, la fonction transfert obtenue :

$$\hat{\Omega}_w(s) = \frac{366.6}{s + 12.65} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 89.96% Dans les tests suivants, l'entrée de tension a été modifiée pour correspondre à un signal en échelon ne contenant que des valeurs positives, ce qui permettait au véhicule d'avancer uniquement. En fixant une valeur maximale de PWM à 0,3, la fonction de transfert obtenue est la suivante :

$$\hat{\Omega}_w(s) = \frac{355.1}{s + 12.31} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 86.43%

Dans le test suivant, une valeur de 0,5 PWM a été appliquée à chaque moteur.

$$\hat{\Omega}_w(s) = \frac{352.1}{s + 11.81} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 87.96% Dans le cadre du test suivant, l'objectif était d'évaluer le comportement du véhicule en mode de rotation. Pour cela, des valeurs de PWM de -1 pour le moteur gauche et 1 pour le moteur droit ont été utilisées.

$$\hat{\Omega}_w(s) = \frac{592.5}{s + 19.23} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 91.25%

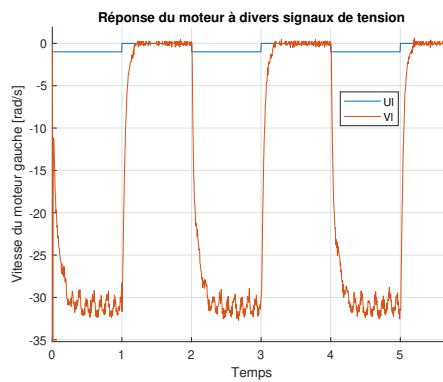


FIGURE 26 – Test sur sol, réponse du moteur gauche

### 7.2.3 Ar en fonction de Ur

Une série de tests a été effectuée avec différents signaux, en fixant une valeur de  $\epsilon = 0.8$  et une valeur maximale de PWM de 0,3, la fonction transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{108.7}{s^2 + 3.979s + 0.05764} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 92.59%

Dans les tests suivants, l'entrée de tension a été modifiée pour correspondre à un signal en échelon ne contenant que des valeurs positives, ce qui permettait au véhicule d'avancer uniquement. En fixant une valeur maximale de PWM à 0,3, la fonction de transfert obtenue est la suivante :

$$\hat{\alpha}_w(s) = \frac{298}{s^2 + 9.722s + 0.2569} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 98.95% Dans le test suivant, une valeur de 0,5 PWM a été appliquée à chaque moteur.

$$\hat{\alpha}_w(s) = \frac{132.1}{s^2 + 4.438s + 0.08677} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 97.13%

Dans le cadre du test suivant, l'objectif était d'évaluer le comportement du véhicule en mode de rotation. Pour cela, des valeurs de PWM de -1 pour le moteur gauche et 1 pour le moteur droit ont été utilisées.

$$\hat{\alpha}_w(s) = \frac{606.6}{s^2 + 22.63s + 8.35e - 13} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 99.26%

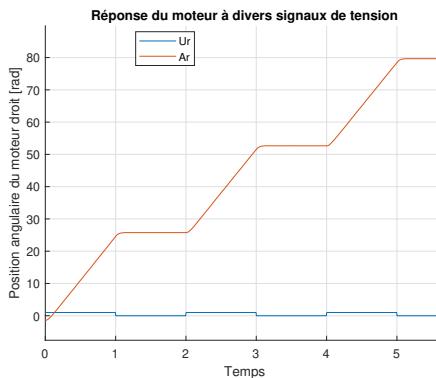


FIGURE 27 – Test sur sol, réponse angulaire moteur droit

#### 7.2.4 Al en fonction de Ul

Une série de tests a été effectuée avec différents signaux, en fixant une valeur de  $\epsilon = 0.8$  et une valeur maximale de PWM de 0,3, la fonction transfert obtenue :

$$\hat{\alpha}_w(s) = \frac{541.7}{s^2 + 19.85s + 0.05716} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 98.28%

Dans les tests suivants, l'entrée de tension a été modifiée pour correspondre à un signal en échelon ne contenant que des valeurs positives, ce qui permettait au véhicule d'avancer uniquement. En fixant une valeur maximale de PWM à 0,3, la fonction de transfert obtenue est la suivante :

$$\hat{\alpha}_w(s) = \frac{540.9}{s^2 + 19.24s + 4.29e - 11} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 97.52% Dans le test suivant, une valeur de 0,5 PWM a été appliquée à chaque moteur.

$$\hat{\alpha}_w(s) = \frac{518.3}{s^2 + 17.51s + 0.09473} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 99.45% Dans le cadre du test suivant, l'objectif était d'évaluer le comportement du véhicule en mode de rotation. Pour cela, des valeurs de PWM de -1 pour le moteur gauche et 1 pour le moteur droit ont été utilisées.

$$\hat{\alpha}_w(s) = \frac{639.4}{s^2 + 20.86s + 4.375e - 09} \hat{U}(s)$$

Pourcentage d'ajustement aux données d'estimation : 99.8%

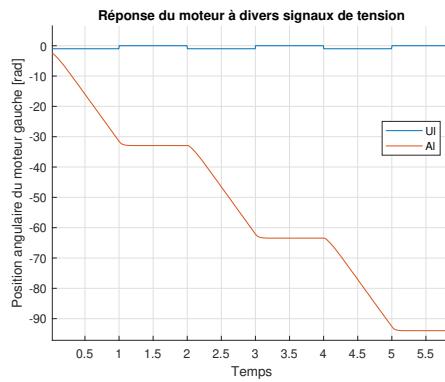


FIGURE 28 – Test sur sol, réponse angulaire moteur gauche

### 7.2.5 $U_{plus}$ vs $\delta_u$

Dans les tests suivants, l'entrée de tension a été modifiée pour correspondre à un signal en échelon ne contenant que des valeurs positives, ce qui permettait au véhicule d'avancer uniquement. En fixant une valeur maximale de PWM à 0,3, la fonction de transfert obtenue est la suivante :

$$\hat{u}_v(s) = \frac{5.455}{s + 9.286} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 86.48%

Dans le test suivant, une valeur de 0,5 PWM a été appliquée à chaque moteur.

$$\hat{u}_v(s) = \frac{3.408}{s + 5.757} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 83.6% Dans ce cas, le test a été réalisé sur le tapis de la piste de course afin que les résultats soient représentatifs de l'environnement où le véhicule serait testé. Par conséquent, la fonction de transfert la plus appropriée est la suivante :

$$\hat{u}_v(s) = \frac{11.47}{s + 20.93} \hat{U}^+(s)$$

Pourcentage d'ajustement aux données d'estimation : 85.25

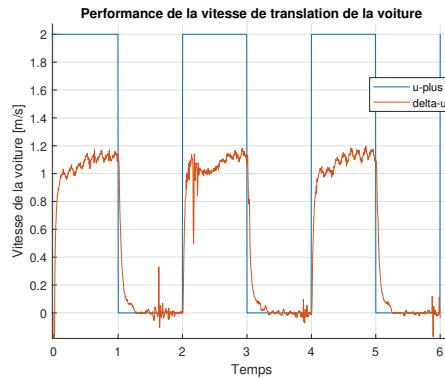


FIGURE 29 – Test sur sol, vitesse de translation

### 7.2.6 $U_{moins}$ vs $\delta_r$

Ensuite, deux tests du véhicule sont fait sur le sol, le premier avec la valeur maximale de PWM à 0,8 et le deuxième à 1. La fonction de transfert a été obtenue pour un système du premier ordre et pour un système du second ordre, dans le but de déterminer lequel représentait le mieux le système, selon le code Simulink.

$$\hat{r}_v(s) = \frac{93.16}{s + 20.67} \hat{U}^-(s)$$

Pourcentage d'ajustement aux données d'estimation : 95.32

$$\hat{r}_v(s) = \frac{2.777e04}{s^2 + 295.8s + 6353} \hat{U}^-(s)$$

Pourcentage d'ajustement aux données d'estimation : 91.67

Le test suivant est considéré comme le plus représentatif car le véhicule se trouvait sur le tapis de la piste de course, ce qui a permis d'obtenir des résultats représentatifs de l'environnement dans lequel le véhicule serait testé.

$$\hat{r}_v(s) = \frac{11.47}{s + 20.93} \hat{U}^-(s)$$

Pourcentage d'ajustement aux données d'estimation : 85.25%

$$\hat{r}_v(s) = \frac{2.696e04}{s^2 + 288.1s + 6325} \hat{U}^-(s)$$

Pourcentage d'ajustement aux données d'estimation : 94.55%

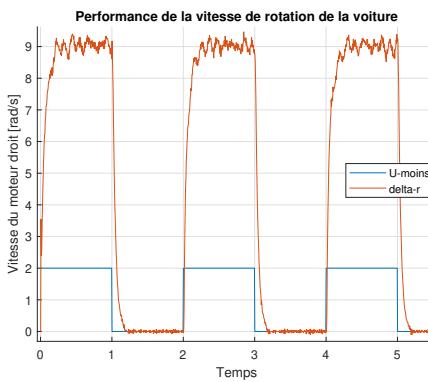


FIGURE 30 – Support moteur, première version

## 8 Résultats finaux

Le robot a réussi à compléter tout le parcours avec une bonne performance, en suivant la ligne sans pics ni oscillations notables et en maintenant une vitesse de référence stable. La vitesse maximale atteinte a été de 0,5 m/s, grâce à deux contrôleurs PID, l'un pour le système de translation et l'autre pour le système de rotation.

La partie électronique comportait les étapes suivantes : d'abord, la lecture des encodeurs et de l'array de suiveur de ligne a été effectuée pour vérifier le bon fonctionnement de chaque capteur. Ensuite, les signaux de sortie ont été captés et, dans le cas des encodeurs, un conditionnement a été appliqué aux signaux produits, car ils étaient bruités et comportaient des pics qu'il fallait lisser. Après avoir conditionné ces signaux pour calculer la vitesse angulaire de chaque moteur et en assurer la cohérence avec les valeurs du datasheet, nous avons procédé à l'identification des systèmes de rotation et de translation et à l'établissement de leurs coefficients dans la fonction de transfert.

Une fois le système identifié, une mise en œuvre automatique a été réalisée et intégrée. Nous avons commencé par intégrer le contrôleur pour le système de translation, qui ne comprenait que les termes proportionnel et intégrateur. Dans ce cas, une réponse rapide et stable du système était nécessaire, donc le terme dérivatif a été exclu : malgré le conditionnement des signaux, il restait des pics, et le terme dérivatif aurait ralenti le contrôleur et amplifié le bruit. Après avoir obtenu des résultats satisfaisants pour la vitesse en direction droite, nous avons introduit le contrôleur pour le système de rotation. L'erreur de ce système a été définie par la lecture de l'encodeur en mètres. Ce contrôleur a été appliqué avec précaution, car les constantes obtenues dans la phase de test initiale montraient que le terme intégrateur induisait une instabilité dans le système.

Par conséquent, nous avons commencé par ajouter uniquement le Kp . À ce stade, le robot suivait la ligne à une vitesse faible de 0,3 m/s. Ensuite, pour diminuer le dépassement, nous avons implémenté le terme Kd, qui nous a également aidés avec l'inertie du système. Après avoir ajusté ce terme, nous avons pu augmenter la vitesse de référence à 0,4 m/s. Pour continuer à l'augmenter, nous avons ajouté le terme Ki avec une technique anti-windup : lorsque l'erreur était égale à zéro, le terme intégrateur était réinitialisé. De plus, nous avons observé que ce terme était plus utile dans les virages serrés de la piste, c'est-à-dire lorsque les lectures de l'encodeur dépassaient 30 mm. Ce dispositif aidait le système à suivre la ligne sans conserver l'erreur passée lorsque le robot devait se déplacer en ligne droite. Ainsi, le contrôleur PID du système de translation n'était pas affecté.

## 8.1 Contrôleur pour le Système de Translation

Les constantes trouvées pour le système de translation sont les suivantes :  $K_{pt} = 0,3077$ ,  $K_{it} = 6,807$ , et  $K_{dt} = 0,0$ . Ce contrôleur a été conçu pour un système nécessitant une stabilité robuste. Un terme proportionnel modéré et un terme intégrateur élevé ont été utilisés pour assurer que le robot maintienne sa position de référence sans déviations à long terme. Le terme dérivatif n'a pas été inclus afin d'éviter que le bruit ne nuise à la stabilité du système.

## 8.2 Contrôleur pour le Système de Rotation (Système Robuste)

Les constantes trouvées pour le système de rotation sont les suivantes :  $K_{pr} = 1,8$ ,  $K_{ir} = 0,0$ ,  $K_{dr} = 20,0$ . Pour le système de rotation, un terme proportionnel plus élevé que celui du système de translation a été choisi pour garantir une réponse précise et rapide aux déviations angulaires. Le terme dérivatif, relativement élevé, aide à réduire le dépassement et à améliorer la réponse dynamique du système aux changements de direction. Le terme intégrateur n'a pas été utilisé pour éviter l'instabilité dans la rotation.

## 8.3 Différence entre la théorie et la pratique pour les valeurs de gain

En translation, on avait trouvé théoriquement les gains suivants :  $K_{pt} = 2.4906$ ;  $K_{it} = 106.8$

Et on trouve en pratique les gains suivants :  $K_{pt} = 0,3077$ ;  $K_{it} = 6,807$

La différence d'ordre de grandeur est de  $10^1$  pour le premier et  $10^2$  pour le deuxième.

Pour la rotation, on avait trouvé :  $K_{pr} = 0.072456$ ;  $K_{dr} = 0.11832$ ;  $K_{ddr} = 0.30076$

Et on a finalement implémenté :  $K_{pr} = 1.8$ ;  $K_{dr} = 20.0$ ;  $K_{ddr} = 0$ .

Ici on a une différence d'ordre de grandeur de  $10^2$  puis  $10^2$  puis un système invitant à avoir en réalité un simple contrôleur PD.

Cela peut s'expliquer par des erreurs dans les valeurs de paramètres pris en compte dans le calcul des gains, ou par des erreurs de choix de  $p_0$  et de  $\sigma$  lors des tests sur Matlab. Il faut aussi noter que ces gains ont été trouvés à l'aide du linéarisation et non d'une modélisation parfaite de la réalité.

De plus, les informations des codeurs ont été filtrées plusieurs fois ce qui entraîne une perte d'information importante par rapport au système réel.

Enfin, les nombreux tests effectués sur des surfaces différentes et ont abîmés les roues et les parties du bolide ce qui a modifié les paramètres du système. Cela souligne l'importance des tests réels pour implémenter un contrôleur.

## 8.4 Contrôleur pour un Système plus Rapide

Pour un système plus rapide, les constantes trouvées sont :  $K_{pr1} = 5,5$ ,  $K_{ir1} = 0,008$ ,  $K_{dr1} = 21,5$ . Pour atteindre une vitesse de réponse plus élevée, la valeur de  $K_{pr}$  a été augmentée pour que le robot réagisse plus rapidement aux erreurs de position. Un petit terme intégrateur a été ajouté pour corriger les erreurs résiduelles sans induire d'instabilité. Le terme dérivatif a été légèrement ajusté vers une valeur plus élevée afin de contrôler l'inertie du système à des vitesses accrues.

## 8.5 Résultat Final de la Trajectoire

En utilisant MATLAB, la trajectoire finale du robot a été obtenue. À partir des vitesses de translation et de rotation calculées, les coordonnées polaires ont été transformées en coordonnées cartésiennes grâce au logiciel de MATLAB pour visualiser la trajectoire suivie par le robot dans un espace bidimensionnel 31.

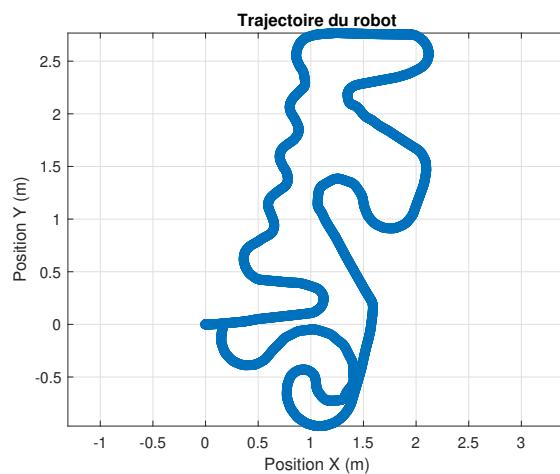


FIGURE 31 – Trajectoire tracé par le robot

Ensuite, les résultats des profils de la position angulaire et des vitesses angulaires de chaque moteur séparément, ainsi que de la vitesse de translation et de la vitesse angulaire du véhicule tout au long du parcours sont présentés.

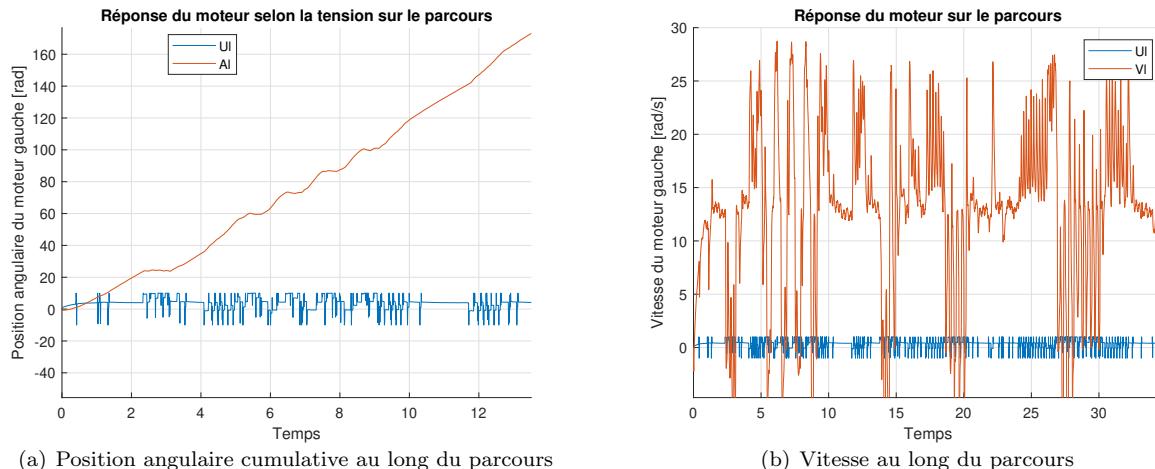


FIGURE 32 – Résultats de la roue gauche par rapport à son signal de commande respectif

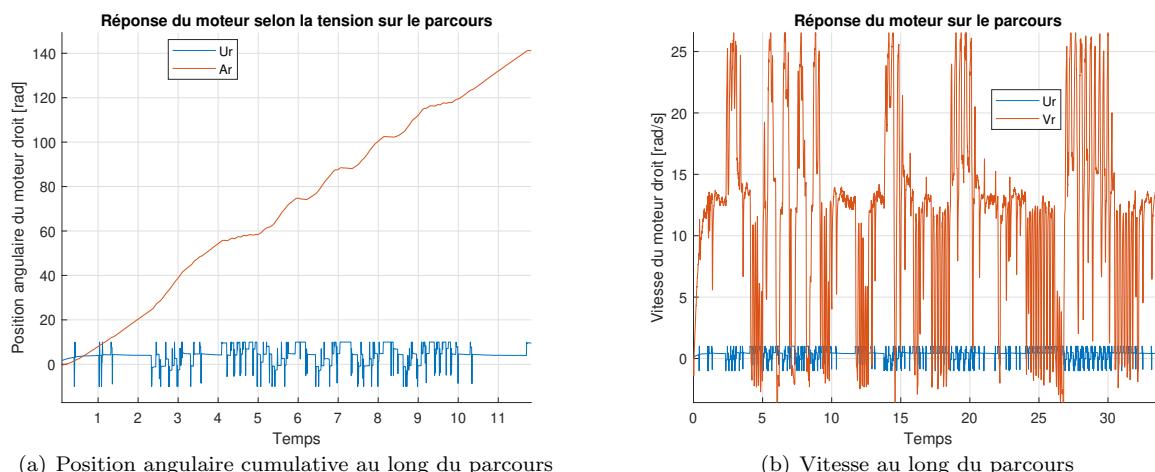


FIGURE 33 – Résultats de la roue droite par rapport à son signal de commande respectif

Bien que les signaux ne soient pas très clairs sur les graphiques en raison des pics présents, on peut observer

que, tout au long du trajet, la vitesse du robot cherchait à se stabiliser autour de la valeur de 0,5, comme le montre le graphique 34.

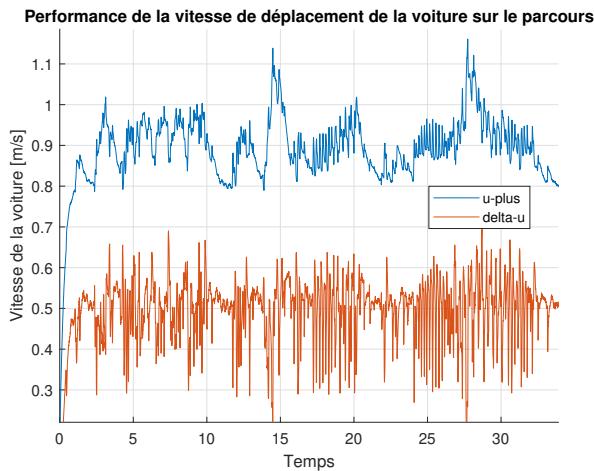


FIGURE 34 – Vitesse de système en translation , résultat de profil de vitesse au long du parcours

Dans le système de rotation, nous observons qu'il y a par moments une tendance à atteindre zéro, en raison des sections du parcours où le robot devait se maintenir en ligne droite 35.

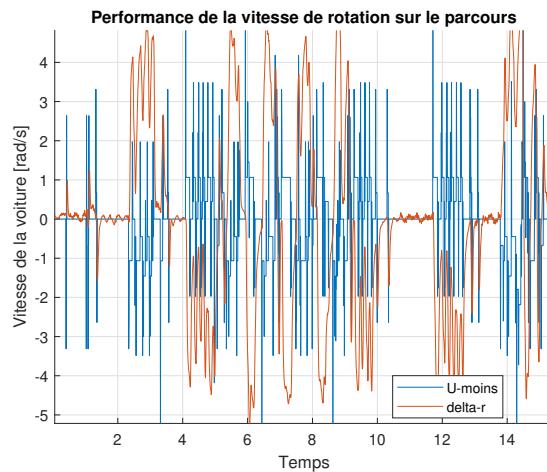


FIGURE 35 – Vitesse de système en rotation, résultat de profil de vitesse au long du parcours

## 9 Résultats expérimentaux des performances individuelles des contrôleurs

Dans cette section, nous observons en temps réel la performance de chaque contrôleur implémenté sur le véhicule, testé séparément.

### 9.1 Performance du contrôleur PI pour le système de translation

D'un part, pour évaluer la performance du contrôleur PI implanté dans notre système de translation, nous avons désactivé le contrôleur de rotation en mettant toutes ses constantes à zéro, puis nous avons réalisé un test. Le contrôleur PI établi pour le système de translation a montré des caractéristiques robustes. Cela a été vérifié en testant le système avec différents poids du véhicule, et le contrôleur est resté stable dans tous les cas. Lors de la phase de conception, ce contrôleur a été réglé pour obtenir un temps de réponse d'environ 1 seconde, sans dépassement théorique. Cependant, comme le montrent les résultats, un dépassement et une déviation ont été observés. Ce phénomène peut être expliqué par une lecture incorrecte de la valeur de l'encodeur lors d'une phase de réinitialisation, ce qui a entraîné une diminution abrupte de la vitesse. L'actionneur a alors rapidement augmenté sa valeur pour compenser, créant ainsi une forme de signal sinusoïdal entre 0,4 et 0,5 secondes (voir images 37). Malgré ce phénomène, nous avons observé que le système a suivi la référence jusqu'à se stabiliser et

maintenir la référence à 0,5 m/s, contrôlant ainsi les vitesses de chaque moteur séparément pour garantir que la vitesse totale du véhicule reste à 0,5 m/s (voir 36).

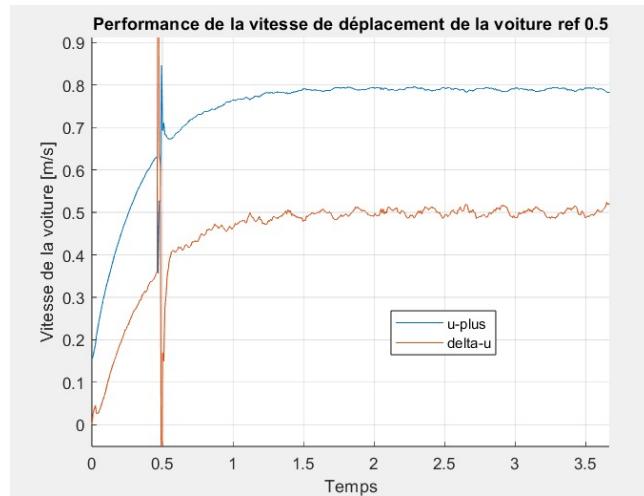


FIGURE 36 – Performance de la vitesse de déplacement

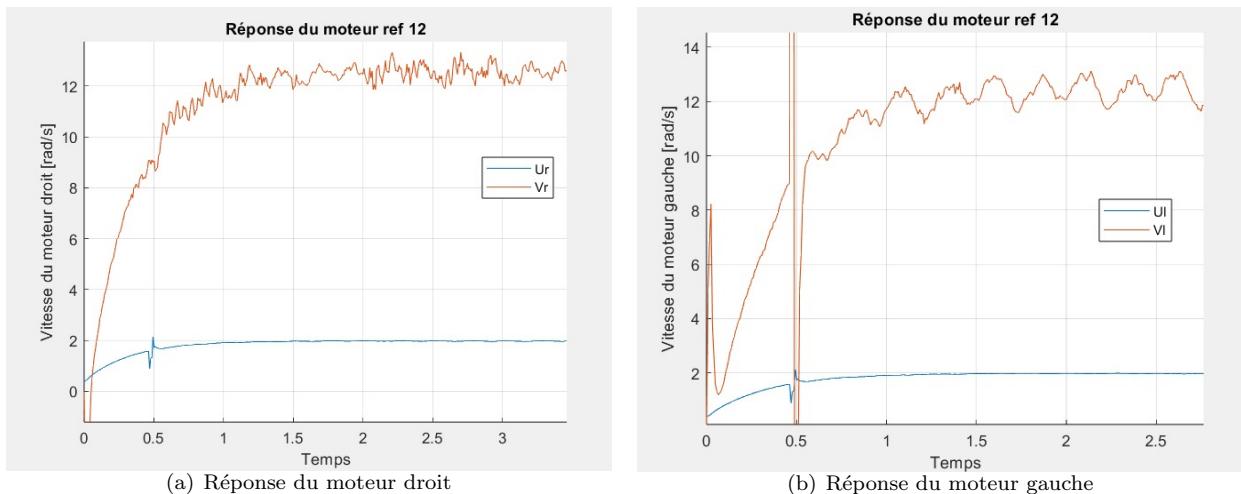


FIGURE 37 – Performance de chaque moteur séparément

## 9.2 Performance du contrôleur PID pour le système de rotation

D'autre part, pour évaluer la performance du contrôleur PID implémenté dans notre système de rotation, nous avons désactivé le contrôleur de translation en mettant toutes ses constantes à zéro, puis nous avons réalisé un test. L'expérience consistait à déplacer la ligne noire en dehors du centre du véhicule pendant quelques secondes, d'abord à gauche, puis à droite du véhicule, ce qui a généré une erreur dans le système de rotation. Le contrôleur a alors dû intervenir pour corriger cette erreur, permettant ainsi d'observer ses performances globales (voir 38).

Lors de la phase de conception, nous avons constaté que ce contrôleur devait être plus rapide que celui de translation, car il devait être capable de suivre la ligne et de continuer sur la trajectoire, plutôt que de simplement maintenir une vitesse constante. De plus, plus l'erreur était grande, la rotation de l'un des moteurs changeait de direction pour éviter de perdre immédiatement la ligne. En revanche, si l'erreur était petite, la vitesse d'un moteur était simplement réduite. Ce phénomène a été observé en pratique après l'implémentation du contrôleur, ce qui explique également pourquoi le véhicule était fluide lors des virages. D'ailleurs, Il peut être expliqué par le fait qu'il s'agit d'un contrôleur plus rapide, avec l'ajout du terme intégral uniquement lorsque l'erreur est plus grande.

Par ailleurs, étant donné que ce système ressemble davantage à un système du second ordre, un dépassement a été observé, avec une valeur d'environ 0,10 rad/s, comme montré dans 39. Le temps de réponse était de 0,03 secondes, et la performance du contrôleur était similaire, qu'il faille tourner à gauche ou à droite, indépendamment de la direction.

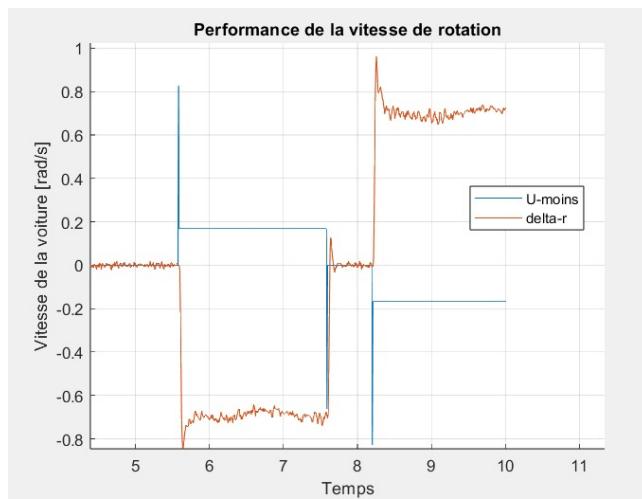


FIGURE 38 – Performance de la vitesse de rotation

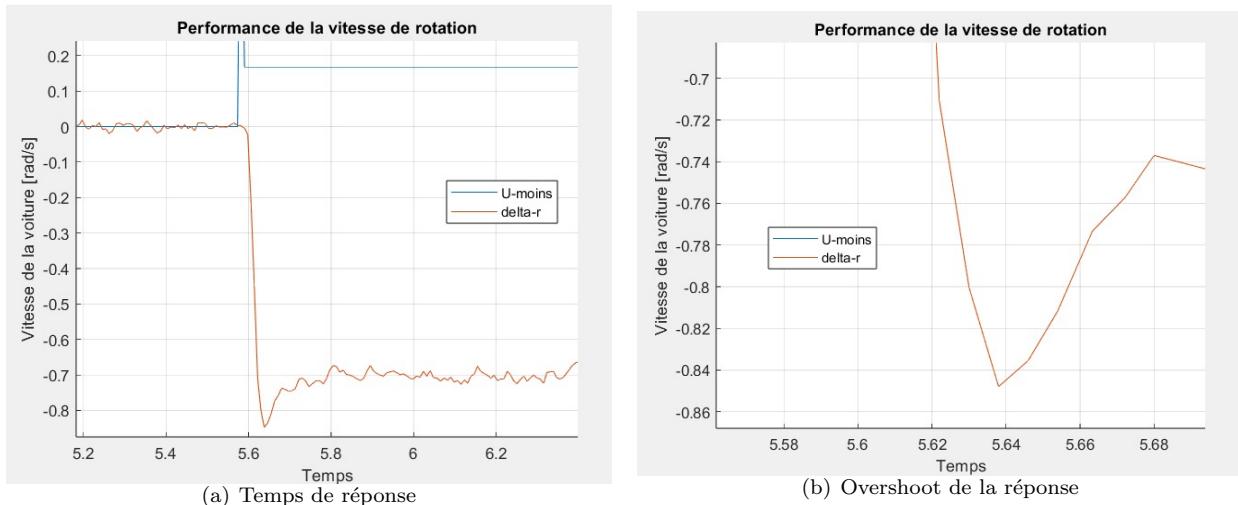


FIGURE 39 – Vues détaillées du performance du contrôleur

## 10 Code du parcours le plus rapide

```

1 // === Bibliothèques nécessaires ===
2 #include "Mecatrolib.h"
3 #include "Wire.h"
4 #include "sensorbar.h"
5 #include "AS5600.h"           // Pour les encodeurs
6 #include "SparkFun_I2C_Mux_Arduino_Library.h" // Pour le multiplexeur I2C
7
8 // === Définition des constantes ===
9 #define CONTROL_LOOP_PERIOD 5          // Période de la boucle de contrôle en ms
10 #define WIFI_SSID "ArduinoMecatrolib" // SSID du WiFi
11 #define WIFI_PASSWORD "password1234"   // Mot de passe WiFi
12
13 // === Déclaration des variables globales ===
14 // Variables pour le fonctionnement général
15 float Ur, Ul, rAngle, rCumulative, rSpeed, lAngle, lCumulative, lSpeed;
16 float Vr, Vl;
17 bool serial = true;
18 long previousMillis = 0;
19
20 // Constantes PID
21 float kpt = 0.3077, kit = 6.807, kdt = 0.0; // Translation
22 float kpr1 = 5.5, kir = 0.008, kdr = 21.5; // Rotation
23
24

```

```

25 // Variables du contr leur pour le moteur droit
26 float Us_eq = 0.0, offset_rAng = 0.0;
27 float Us_p = 0, Us_i = 0, Us_ip = 0, Us_d = 0, Us_dp = 0, Us_pid = 0, Us_pid_reel = 0;
28 float N = 50, Es = 0, Esp = 0, rdutycycle = 0, rPWM = 0;
29
30 // Variables du contr leur pour le moteur gauche
31 float Ur_eq = 0.0, Ur_p1 = 0, Ur_i = 0, Ur_ip = 0, Ur_d = 0, Ur_dp = 0, Ur_pid = 0,
      Ur_pid_reel = 0;
32 float ldutycycle = 0, lPWM = 0;
33 float Ts=5.0;//5ms
34
35 // Propri t s physiques
36 float r_roue = 0.04, l_dis = 0.257;
37 float delta_u = 0, delta_r = 0, delta_phi = 0;
38 float delta_rp=0,delta_phip=0;
39 float pos = 0, posp = 0;
40
41 // Variables pour le filtre
42 float Bk = 0;
43 int k = 0;
44
45 // Variables pour les vitesses angulaires
46 float angleActuel1, anglePrecedent1, vitesseAngulaire1 = 0.0;
47 unsigned long temp_prec1 = 0;
48 float angleActuel2, anglePrecedent2, vitesseAngulaire2 = 0.0;
49 unsigned long temp_prec2 = 0;
50
51 // === Capteurs et p riph riques ===
52 const uint8_t SX1509_ADDRESS = 0x3E;           // Adresse I2C du capteur de ligne
53 SensorBar mySensorBar(SX1509_ADDRESS);        // Capteur de ligne
54 QWIICMUX multiplexer;                         // Multiplexeur I2C
55 AS5600 rightEncoder(&Wire1);                 // Encodeur droit
56 AS5600 leftEncoder(&Wire1);                  // Encodeur gauche
57
58 // === Setup ===
59 void setup()
60 {
61     Serial.begin(230400);                      // Initialiser la communication s rie
62     Wire1.begin();                            // Initialiser le bus I2C secondaire
63     if (serial && !multiplexer.begin(0x70, Wire1)) {
64         Serial.println("Erreur : Multiplexeur I2C introuvable.");
65         return;
66     }
67
68     bool isInit = true;
69
70     // Configuration des encodeurs
71     multiplexer.setPort(6);
72     rightEncoder.begin();
73     if (!rightEncoder.isConnected()) {
74         Serial.println("Erreur : Encodeur droit non connect .");
75         isInit = false;
76     }
77
78     multiplexer.setPort(7);
79     leftEncoder.begin();
80     if (!leftEncoder.isConnected()) {
81         Serial.println("Erreur : Encodeur gauche non connect .");
82         isInit = false;
83     }
84
85     if (isInit) {
86         Wire1.setClock(400000);                // Configurer la vitesse I2C      400 kHz
87         mecatro::configureArduino(CONTROL_LOOP_PERIOD);
88     }
89
90     // Configuration du capteur de ligne
91     mySensorBar.setBarStrobe();
92     mySensorBar.clearInvertBits();
93     if (!mySensorBar.begin()) {
94         Serial.println(" chec de communication avec le capteur de ligne.");
95         while (1);
96     }
97 }
98
99

```

```

100 void loop()
101 {
102     mecatro::run();
103 }
104
105 // Fonction de filtre moyenne glissante
106 float filtreMoyenneGlissante(float encodeur) {
107     Bk = (float(k) / (k + 1)) * Bk + (1.0 / (k + 1)) * encodeur;
108     k++;
109     return Bk;
110 }
111
112 void reconstruireX() {
113     delta_u = r_roue / 2 * (vitesseAngulaire1 + vitesseAngulaire2);
114     delta_r = r_roue / l_dis * (vitesseAngulaire1 - vitesseAngulaire2);
115     delta_phi = delta_phi + delta_r * Ts / 1000;
116     position();
117 }
118
119 void controlePID() {
120     reconstruireX();
121     Es = 0.56 - delta_u; // 0.5
122
123     Us_p = kpt * Es;
124     Us_i = Us_ip + kit * (Ts / 1000) * Esp;
125     Us_d = kdt * (Es - Esp);
126     Us_pid = Us_p + Us_i + Us_d;
127     Esp = Es;
128     Us_ip = Us_i;
129     Us_dp = Us_d;
130     Us_pid_reel = Us_eq + Us_pid;
131
132     Ur_p1 = kpr1 * (-pos / 100);
133     if (-30 > pos || pos > 30) {
134         Ur_i = Ur_ip + kir * (Ts / 1000) * posp;
135     } else {
136         Ur_i = 0;
137     }
138     Ur_d = kdr * (-pos / 100 - posp);
139     Ur_pid = Ur_p1 + Ur_i + Ur_d;
140     Ur_pid_reel = Ur_eq + Ur_pid;
141
142     delta_rp = delta_r;
143     delta_phip = delta_phi;
144     posp = -pos / 100;
145     Ur_ip = Ur_i;
146     Ur_dp = Ur_d;
147
148     rPWM = (Us_pid_reel + Ur_pid_reel) / 2;
149     lPWM = (Us_pid_reel - Ur_pid_reel) / 2;
150
151     rdutycycle = min(max(rPWM, -1), 1); // -1      1
152     ldutycycle = min(max(lPWM, -1), 1); // -1      1
153
154     mecatro::setMotorDutyCycle(rdutycycle, ldutycycle);
155 }
156
157 float fenetre1[5];
158 int indiceFenetre1 = 0;
159 float sommeFenetre1 = 0;
160
161 float filtreMoyenneMobile1(float nouvelleValeur) {
162     sommeFenetre1 -= fenetre1[indiceFenetre1];
163     fenetre1[indiceFenetre1] = nouvelleValeur;
164     sommeFenetre1 += nouvelleValeur;
165     indiceFenetre1 = (indiceFenetre1 + 1) % 5;
166     return sommeFenetre1 / 5;
167 }
168
169 const int tailleFenetre = 5;
170 float fenetre2[tailleFenetre];
171 int indiceFenetre2 = 0;
172 float sommeFenetre2 = 0;
173
174 float filtreMoyenneMobile2(float nouvelleValeur) {
175     sommeFenetre2 -= fenetre2[indiceFenetre2];

```

```

176     fenetre2[indiceFenetre2] = nouvelleValeur;
177     sommeFenetre2 += nouvelleValeur;
178     indiceFenetre2 = (indiceFenetre2 + 1) % tailleFenetre;
179     return sommeFenetre2 / tailleFenetre;
180 }
181
182 int i = 0;
183 float alpha = 0.9;
184
185 float filtreMoyenneExponentielle(float nouvelleValeur, float Vi) {
186     Vi = alpha * nouvelleValeur + (1 - alpha) * Vi;
187     return Vi;
188 }
189
190 float trouverVitesse1(float angle1) {
191     unsigned long temp_1 = millis();
192     if (temp_1 - temp_prec1 >= CONTROL_LOOP_PERIOD) {
193         angleActuel1 = filtreMoyenneExponentielle(angle1, angleActuel1);
194         float deltaTemps = (temp_1 - temp_prec1) / 1000.0;
195         vitesseAngulaire1 = filtreMoyenneMobile1((angleActuel1 - anglePrecedent1) / deltaTemps
196             );
197         anglePrecedent1 = angleActuel1;
198         temp_prec1 = temp_1;
199     }
200     return vitesseAngulaire1;
201 }
202
203 float trouverVitesse2(float angle2) {
204     unsigned long temp_2 = millis();
205     if (temp_2 - temp_prec2 >= CONTROL_LOOP_PERIOD) {
206         angleActuel2 = filtreMoyenneExponentielle(angle2, angleActuel2);
207         float deltaTemps = (temp_2 - temp_prec2) / 1000.0;
208         vitesseAngulaire2 = filtreMoyenneMobile2((angleActuel2 - anglePrecedent2) / deltaTemps
209             );
210         anglePrecedent2 = angleActuel2;
211         temp_prec2 = temp_2;
212     }
213     return vitesseAngulaire2;
214 }
215
216 void position() {
217     pos = (mySensorBar.getPosition() * 105 / 254); // en m tres
218 }
219
220 void mecatro::controlLoop() {
221     float angle1;
222     float angle2;
223     multiplexer.setPort(6);
224     angle1 = rightEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS;
225     vitesseAngulaire1 = trouverVitesse1(angle1);
226     multiplexer.setPort(7);
227     angle2 = leftEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS;
228     vitesseAngulaire2 = trouverVitesse2(angle2);
229
230     unsigned long tempsActuel = millis(); // Temps actuel en millisecondes
231     uint8_t valeurBrute = mySensorBar.getRaw();
232
233     controlePID();
234 }
```

## 11 Code identification paramètres

```

1 /* Code de Identification parametres du robot.
2
3     Remarque : ce code n'cessite les biblioth ques suivantes (installez-les via le
4         gestionnaire de biblioth ques) :
5         - SparkFun I2C Mux Arduino Library
6         - AS5600 library
7 */
8
9 // Inclusion de la biblioth que actuelle
#include "MecatroUtils.h"
```

```

11 // Inclusion des bibliothèques AS5600 (pour les encodeurs) et SparkFun I2C Mux (pour le
12     multiplexeur)
13 #include "AS5600.h"
14 #include "SparkFun_I2C_Mux_Arduino_Library.h"
15
16 // Déclaration des variables globales
17 float Ur, Ul, Ar, Al, Vr, Vl, Bk;
18 int k = 0;
19
20 // Propriétés physiques
21 float r_roue = 0.04; // Rayon des roues en mètres
22 float l_dis = 0.257; // Distance entre les roues en mètres
23
24 float delta_u, delta_r, U_plus, U_moins;
25
26 float offset_lAng = 0.0;
27 float offset_rAng = 0.0;
28
29 // Entête pour la communication I2C
30 #include "Wire.h"
31
32 // Définition de la période de la boucle de contrôle en millisecondes
33 #define CONTROL_LOOP_PERIOD 5
34
35 #define WIFI_SSID "ArduinoMecatroGr5"
36 #define WIFI_PASSWRD "password123"
37
38 long periodo = 2000.0; // Période du signal en millisecondes
39 float mini = 10.0;
40 float signalValue = 0.0;
41
42 unsigned long tiempoAnterior1 = 0; // Variable pour stocker le temps précédent
43 float aumenta = 0;
44
45 // Déclaration du multiplexeur I2C
46 QWIICMUX multiplexer;
47
48 // Wire1 est utilisé pour le port Qwiic sur cette carte Arduino.
49 // Tous les appareils doivent être assignés à ce Wire (soit ici (encodeurs) soit lors de l'
50     initialisation (multiplexeur), selon les appareils).
51 AS5600 rightEncoder(&Wire1);
52 AS5600 leftEncoder(&Wire1);
53
54 void setup() {
55     // Configuration de la communication série avec le PC (pour le débogage et l'
56         enregistrement)
57     Serial.begin(230400);
58     // Démarrage de la communication I2C sur le port Qwiic
59     Wire1.begin();
60
61     // Initialisation du multiplexeur. Son adresse I2C est 0x70, et nous communiquons via le
62         port Qwiic (Wire1).
63     if (!multiplexer.begin(0x70, Wire1)) {
64         Serial.println("Erreur : multiplexeur I2C non trouvé. Vérifiez le câblage.");
65     } else {
66         bool isInit = true;
67         // Configuration du multiplexeur pour utiliser le port 0 pour communiquer avec l'encodeur
68             droit
69         multiplexer.setPort(6);
70         rightEncoder.begin();
71         offset_rAng = rightEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS;
72         if (!rightEncoder.isConnected()) {
73             Serial.println("Erreur : impossible de se connecter à l'encodeur droit. Vérifiez le
74                 câblage.");
75             isInit = false;
76         }
77         // Configuration du multiplexeur pour utiliser le port 3 pour communiquer avec l'encodeur
78             gauche
79         multiplexer.setPort(7);
80         leftEncoder.begin();
81         offset_lAng = leftEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS;
82         if (!leftEncoder.isConnected()) {
83             Serial.println("Erreur : impossible de se connecter à l'encodeur gauche. Vérifiez le
84                 câblage.");
85             isInit = false;
86     }

```

```

79
80     if (isInit) {
81         // Configuration de la vitesse d'horloge I2C      400 kHz (mode rapide).    faire apr s l
82         //initialisation pour viter les r initialisations.
83         Wire1.setClock(400000);
84         // Configuration de la commande des moteurs et de l'appel de la boucle de r troaction.
85         mecatro::configureArduino(CONTROL_LOOP_PERIOD);
86     }
87
88     // Initialisation de la t l m trie
89     unsigned int const nVariables = 12;
90     String variableNames[nVariables] = {"Ur", "Ul", "U_plus", "U_moins", "delta_u", "delta_r", " "
91         "Vr_f1", "VL_f1", "Vr_f2", "VL_f2", "Ar", "Al"};
92     mecatro::initTelemetry(WIFI_SSID, WIFI_PASSWRD, nVariables, variableNames,
93     CONTROL_LOOP_PERIOD);
94 }
95
96 void loop() {
97     // Cette fonction est indispensable : sans elle, rien ne se passe !
98     // Elle ne retourne jamais, placez tout votre code dans mecatro::controlLoop.
99     mecatro::run();
100 }
101
102 float ventanai[10]; // Tableau pour stocker les derni res lectures
103 int indiceVentana1 = 0; // Indice actuel dans la fen tre
104 float sumaVentana1 = 0; // Somme des valeurs dans la fen tre
105
106 float filtroMediaMovil1(float valorNuevo) {
107     // Soustrait la valeur la plus ancienne de la somme
108     sumaVentana1 -= ventanai[indiceVentana1];
109
110     // Remplace la valeur la plus ancienne par la nouvelle et l'ajoute la somme
111     ventanai[indiceVentana1] = valorNuevo;
112     sumaVentana1 += valorNuevo;
113
114     // Incr mente l'indice et le r initialise s'il atteint la fin de la fen tre
115     indiceVentana1 = (indiceVentana1 + 1) % 10;
116
117     // Calcule la moyenne
118     return sumaVentana1 / 10;
119 }
120
121 const int ventanaTamano = 10; // Taille de la fen tre de la moyenne mobile
122 float ventana2[ventanaTamano]; // Tableau pour stocker les derni res lectures
123 int indiceVentana2 = 0; // Indice actuel dans la fen tre
124 float sumaVentana2 = 0; // Somme des valeurs dans la fen tre
125
126 float filtroMediaMovil2(float valorNuevo) {
127     // Soustrait la valeur la plus ancienne de la somme
128     sumaVentana2 -= ventana2[indiceVentana2];
129
130     // Remplace la valeur la plus ancienne par la nouvelle et l'ajoute la somme
131     ventana2[indiceVentana2] = valorNuevo;
132     sumaVentana2 += valorNuevo;
133
134     // Incr mente l'indice et le r initialise s'il atteint la fin de la fen tre
135     indiceVentana2 = (indiceVentana2 + 1) % ventanaTamano;
136
137     // Calcule la moyenne
138     return sumaVentana2 / ventanaTamano;
139 }
140
141 // float alpha = 0.1; // Ajuste cette valeur entre 0 et 1 pour plus ou moins de lissage 0.0015
142 // 0.08-
143 int i=0;
144 float alpha=0.9;
145
146 float filtroMediaExponencial(float valorNuevo, float Vi) {
147     // Applique un filtre de moyenne exponentielle
148     Vi = alpha * valorNuevo + (1 - alpha) * Vi;
149     return Vi;
150 }
151
152 // Variables pour le calcul de la vitesse
153 float anguloActual1; // Angle actuel (capteur 1)

```

```

151 float anguloAnterior1; // Angle précédent (capteur 1)
152 float velocidadAngular1 = 0.0; // Vitesse angulaire (capteur 1)
153 unsigned long temp_prev1 = 0; // Temps précédent (capteur 1)
154
155 // Variables pour le calcul de la vitesse
156 float anguloActual2; // Angle actuel (capteur 2)
157 float anguloAnterior2; // Angle précédent (capteur 2)
158 float velocidadAngular2 = 0.0; // Vitesse angulaire (capteur 2)
159 unsigned long temp_prev2 = 0; // Temps précédent (capteur 2)
160
161 float encontrarVelocidad1(float angulo1) {
162     unsigned long temp_1 = millis();
163     if (temp_1 - temp_prev1 >= CONTROL_LOOP_PERIOD) {
164         // Applique un filtre exponentiel sur l'angle
165         anguloActual1 = filtroMediaExponencial(angulo1, anguloActual1);
166         // Calcule l'cart de temps (en secondes)
167         float deltaTiempo = (temp_1 - temp_prev1) / 1000.0;
168         // Calcule la vitesse angulaire
169         velocidadAngular1 = (anguloActual1 - anguloAnterior1) / deltaTiempo;
170         // Met jour les valeurs précédentes
171         anguloAnterior1 = anguloActual1;
172         temp_prev1 = temp_1;
173     }
174     return velocidadAngular1;
175 }
176
177 float encontrarVelocidad2(float angulo2) {
178     unsigned long temp_2 = millis();
179     if (temp_2 - temp_prev2 >= CONTROL_LOOP_PERIOD) {
180         // Applique un filtre exponentiel sur l'angle
181         anguloActual2 = filtroMediaExponencial(angulo2, anguloActual2);
182         // Calcule l'cart de temps (en secondes)
183         float deltaTiempo = (temp_2 - temp_prev2) / 1000.0;
184         // Calcule la vitesse angulaire
185         velocidadAngular2 = (anguloActual2 - anguloAnterior2) / deltaTiempo;
186         // Met jour les valeurs précédentes
187         anguloAnterior2 = anguloActual2;
188         temp_prev2 = temp_2;
189     }
190     return velocidadAngular2;
191 }
192
193 // Cette fonction est appellée périodiquement, toutes les CONTROL_LOOP_PERIOD ms.
194 // Mettez tout votre code ici.
195 void mecatro::controlLoop() {
196     float angulo1;
197     float angulo2;
198
199     multiplexer.setPort(6);
200
201     // Recupère l'angle cumulatif de l'encodeur droit
202     angulo1 = rightEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS;
203
204     velocidadAngular1 = encontrarVelocidad1(angulo1) - offset_rAng;
205
206     multiplexer.setPort(7);
207
208     // Recupère l'angle cumulatif de l'encodeur gauche
209     angulo2 = leftEncoder.getCumulativePosition() * AS5600_RAW_TO_RADIANS - offset_lAng;
210
211     velocidadAngular2 = encontrarVelocidad2(angulo2);
212
213     // Calculs pour les tensions des moteurs
214     Ur = 1 * function();
215     Ul = 1 * function();
216
217     U_plus = Ur + Ul;
218     U_moins = Ur - Ul;
219     delta_u = r_roue / 2 * (velocidadAngular1 + velocidadAngular2);
220     delta_r = r_roue / l_dis * (velocidadAngular1 - velocidadAngular2);
221
222     // Enregistre les données pour le suivi
223     mecatro::log(0, Ur);
224     mecatro::log(1, Ul);
225     mecatro::log(2, U_plus);
226     mecatro::log(3, U_moins);

```

```

227     mecatro::log(4, delta_u);
228     mecatro::log(5, delta_r);
229     mecatro::log(6, velocidadAngular1);
230     mecatro::log(7, velocidadAngular2);
231     mecatro::log(8, filtroMediaMovil1(velocidadAngular1));
232     mecatro::log(9, filtroMediaMovil2(velocidadAngular2));
233     mecatro::log(10, anguloActual1);
234     mecatro::log(11, anguloActual2);

235
236     // Désactive les moteurs en fixant un cycle de service à 0
237     mecatro::setMotorDutyCycle(Ul, Ur);
238 }
239 float function() {
240     unsigned long tiempoActual = millis(); // Obtient le temps actuel
241     unsigned long tiempoTranscurrido = tiempoActual - tiempoAnterior1;

242     // Change le signal toutes les 2 * période millisecondes
243     if (tiempoTranscurrido <= 2 * periodo) {
244         signalValue = creneau1(tiempoTranscurrido);
245     } else if (tiempoTranscurrido <= 4 * periodo) {
246         signalValue = creneau0(tiempoTranscurrido);
247     } else if (tiempoTranscurrido <= 5 * periodo) {
248         signalValue = 0;
249     } else if (tiempoTranscurrido <= 6 * periodo) {
250         signalValue = sawtooth(tiempoTranscurrido);
251     } else if (tiempoTranscurrido <= 7 * periodo) {
252         signalValue = 0;
253     } else if (tiempoTranscurrido <= 8 * periodo) {
254         signalValue = senoNegativo(tiempoTranscurrido);
255     } else if (tiempoTranscurrido <= 10 * periodo) {
256         signalValue = 0;
257     } else {
258         // R initialise le timer après avoir complété un cycle complet de signaux
259         tiempoAnterior1 = tiempoActual;
260     }

261     return signalValue;
262 }
263
264 // Fonction de signal carré (niveau haut et bas)
265 float creneau0(unsigned long tiempoTranscurrido) {
266     if ((tiempoTranscurrido / (periodo / 2)) % 2 == 0) {
267         return 0.0; // Niveau haut
268     } else {
269         return 1.0; // Niveau bas
270     }
271 }
272
273 // Fonction de signal carré (niveau positif et négatif)
274 float creneau1(unsigned long tiempoTranscurrido) {
275     if ((tiempoTranscurrido / (periodo / 2)) % 2 == 0) {
276         return -1.0; // Niveau positif
277     } else {
278         return 1.0; // Niveau négatif
279     }
280 }
281
282 // Fonction de signal en dent de scie
283 float sawtooth(unsigned long tiempoTranscurrido) {
284     return float(tiempoTranscurrido % periodo) / (periodo); // Augmente linéairement
285 }
286
287 // Signal sinusoïdal variant de 1 à -1
288 float senoNegativo(unsigned long tiempoTranscurrido) {
289     float t = float(tiempoTranscurrido % periodo) / periodo;
290     return sin(2 * PI * t); // Variation de 1 à -1
291 }
292
293 // Fonction de signal sinusoïdal
294 float sinus(unsigned long tiempoTranscurrido) {
295     unsigned long tiempoActual = millis(); // Obtient le temps actuel
296     return sin(tiempoActual / 1000.0); // Signal sinusoïdal
297 }
298
299 float creneaux() {
300     unsigned long tiempoActual = millis(); // Obtient le temps actuel
301 }
```

```

303
304     // V rifie si la moiti de la p riode s'est coule
305     if (tiempoActual - tiempoAnterior1 >= periodo / 2) {
306         // Change l' tat du signal
307         signalValue = !signalValue; // Inverse le signe -signalValue
308         tiempoAnterior1 = tiempoActual; // Met jour le temps pr c dent
309     }
310
311     // Retourne la valeur actuelle du signal
312     return signalValue;
313 }
```

Listing 3 – Code de Identification parametres du robot.

## 12 Code MatLab

```

1 close all; clear all; clc;
2
3
4 %WiFi settings
5 IP = '192.168.4.1';
6 PORT_NUMBER = 80;
7
8 %USB settings
9 port_name = '/dev/cu.usbmodemF412FA70B6442';
10 baudrate = 230400;
11
12 %Recording time
13 Tmax = 50;
14
15 %Communication method
16 method = 'WiFi';
17
18 %Gains
19 K = [];
20
21 if strcmp(method, 'WiFi')
22     [log_time, data_values, line_idx] = get_data_WiFi(IP, PORT_NUMBER, Tmax, K);
23 elseif strcmp(method, 'USB')
24     [log_time, data_values, line_idx] = get_data_USB(port_name, Tmax, baudrate, K);
25 else
26     print('Use a valid connection method')
27 end
28
29 %Display logged variable names
30 disp(data_values.keys)
31
32 % Plot logged variables
33 figure;
34 hold on;
35 legend_labels = data_values.keys;
36 for i = 1:numel(legend_labels)
37     d = legend_labels{i};
38     v = data_values(d);
39     plot(log_time, v, 'DisplayName', d);
40     title(d); % Opcional: Agrega un t tulo con el nombre de la serie de datos
41     xlabel('Log Time'); % Opcional: Etiqueta del eje x
42     ylabel('Value'); % Opcional: Etiqueta del eje y
43     legend show; % Muestra la leyenda
44     %plot(log_time, v, 'DisplayName', d);
45 end
46 hold off;
47 legend('Location', 'best');
48 grid on;
49 % Factor de conversi n de kgf.mm a Nm
50 factor_conversion = 9.80665e-3;
51 k_couple=144*factor_conversion/(5.6-0.15);
52
53 %% Obtencion de las funciones de transferencia angulos y velocidades motores
54
55 % Datos de entrada y salida (aseg rate de que est n alineados en tama o)
56 y1 = data_values('Vr_f1'); % Datos de salida (posici n del motor)
57 u1 = data_values('Ur'); % Datos de entrada (voltaje aplicado al motor)
58 Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
```

```

59 % Crear objeto iddata
60 data = iddata(y1', u1', Ts);
61
62 % Número de ceros y polos para el modelo de motor
63 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
64 num_poles = 1; % Número de polos
65
66 % Estimar función de transferencia
67 sys1 = tfest(data, num_poles, num_zeros);
68
69 % Mostrar la función de transferencia estimada
70 disp(sys1);
71
72 % Datos de entrada y salida (asegurarse de que estén alineados en tamaño)
73 y2 = data_values('Vl_f1'); % Datos de salida (posición del motor)
74 u2 = data_values('U1'); % Datos de entrada (voltaje aplicado al motor)
75 %Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
76
77 % Crear objeto iddata
78 data = iddata(y2', u2', Ts);
79
80 % Número de ceros y polos para el modelo de motor
81 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
82 num_poles = 1; % Número de polos (un sistema de segundo orden)
83
84 % Estimar función de transferencia
85 sys2 = tfest(data, num_poles, num_zeros);
86
87 % Mostrar la función de transferencia estimada
88 disp(sys2);
89
90
91 % Datos de entrada y salida (asegurarse de que estén alineados en tamaño)
92 y3 = data_values('Ar'); % Datos de salida (posición del motor)
93 u3 = data_values('Ur'); % Datos de entrada (voltaje aplicado al motor)
94 %Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
95
96 % Crear objeto iddata
97 data = iddata(y3', u3', Ts);
98
99 % Número de ceros y polos para el modelo de motor
100 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
101 num_poles = 2; % Número de polos (un sistema de segundo orden)
102
103 % Estimar función de transferencia
104 sys3 = tfest(data, num_poles, num_zeros);
105
106 % Mostrar la función de transferencia estimada
107 disp(sys3);
108
109 % Datos de entrada y salida (asegurarse de que estén alineados en tamaño)
110 y4 = data_values('Al'); % Datos de salida (posición del motor)
111 u4 = data_values('U1'); % Datos de entrada (voltaje aplicado al motor)
112 %Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
113
114 % Crear objeto iddata
115 data = iddata(y4', u4', Ts);
116
117 % Número de ceros y polos para el modelo de motor
118 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
119 num_poles = 2; % Número de polos (un sistema de segundo orden)
120
121 % Estimar función de transferencia
122 sys4 = tfest(data, num_poles, num_zeros);
123
124 % Mostrar la función de transferencia estimada
125 disp(sys4);
126
127
128
129 %% Obtención de las funciones de transferencia sistema global
130 % Datos de entrada y salida
131 y5 = data_values('delta_u'); % Datos de salida (posición del motor)
132 u5 = data_values('U_plus'); % Datos de entrada (voltaje aplicado al motor)
133 %Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
134

```

```

135
136 % Crear objeto iddata
137 data = iddata(y5', u5', Ts);
138
139 % Número de ceros y polos para el modelo de motor
140 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
141 num_poles = 1; % Número de polos (un sistema de segundo orden)
142
143 % Estimar función de transferencia
144 sys5 = tfest(data, num_poles, num_zeros);
145
146 disp(sys5);
147
148 % Datos de entrada y salida (asegúrate de que estén alineados en tamaño)
149 y6 = data_values('delta_r'); % Datos de salida (posición del motor)
150 u6 = data_values('U_moins'); % Datos de entrada (voltaje aplicado al motor)
151
152 % Crear objeto iddata
153 data = iddata(y6', u6', Ts);
154
155 % Número de ceros y polos para el modelo de motor
156 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
157 num_poles = 1; % Número de polos
158
159 % Estimar función de transferencia
160 sys6 = tfest(data, num_poles, num_zeros);
161
162 disp(sys6);
163 %% Con 2 filtros
164
165 % Datos de entrada y salida
166 y7 = data_values('Vr_f2'); % Datos de salida (posición del motor)
167 u7 = data_values('Ur'); % Datos de entrada (voltaje aplicado al motor)
168 Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
169
170 % Crear objeto iddata
171 data = iddata(y7', u7', Ts);
172
173 % Número de ceros y polos para el modelo de motor
174 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
175 num_poles = 1; % Número de polos (un sistema de segundo orden)
176
177 % Estimar función de transferencia
178 sys7 = tfest(data, num_poles, num_zeros);
179
180 % Mostrar la función de transferencia estimada
181 disp(sys7);
182
183 % Datos de entrada y salida (asegúrate de que estén alineados en tamaño)
184 y8 = data_values('Vl_f2'); % Datos de salida (posición del motor)
185 u8 = data_values('Ul'); % Datos de entrada (voltaje aplicado al motor)
186 %Ts = mean(diff(log_time)); % Calcula el tiempo de muestreo promedio entre puntos
187
188 % Crear objeto iddata
189 data = iddata(y8', u8', Ts);
190
191 % Número de ceros y polos para el modelo de motor
192 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)
193 num_poles = 1; % Número de polos (un sistema de segundo orden)
194
195 % Estimar función de transferencia
196 sys8 = tfest(data, num_poles, num_zeros);
197
198 % Mostrar la función de transferencia estimada
199 disp(sys8);
200 %% si fuera de 2 orden
201
202 % Datos de entrada y salida
203 y10 = data_values('delta_r'); % Datos de salida (posición del motor)
204 u10 = data_values('U_moins'); % Datos de entrada (voltaje aplicado al motor)
205
206 % Crear objeto iddata
207 data = iddata(y10', u10', Ts);
208
209 % Número de ceros y polos para el modelo de motor
210 num_zeros = 0; % Número de ceros (un sistema simple de motor puede no tener ceros)

```

```

211 num_poles = 2; % Número de polos (un sistema de segundo orden)
212
213 % Estimar función de transferencia
214 sys10 = tfest(data, num_poles, num_zeros);
215
216 disp(sys10);
217
218 %% Comparativa función velocidad con dos filtros
219
220 % Plot logged variables
221 figure(10);
222 hold on;
223 plot(log_time, data_values('Vr_f1'), 'DisplayName', 'Vr');
224 plot(log_time, data_values('Vr_f2'), 'DisplayName', 'Vr-deux-filtre');
225 xlabel('Log Time'); % Opcional: Etiqueta del eje x
226 ylabel('Value'); % Opcional: Etiqueta del eje y
227 legend show; % Muestra la leyenda
228 hold off;
229 legend('Location', 'best');
230 grid on;
231
232 figure(11);
233 hold on;
234 plot(log_time, data_values('Vl_f1'), 'DisplayName', 'f1');
235 plot(log_time, data_values('Vl_f2'), 'DisplayName', 'Vl-deux-filtre');
236 xlabel('Log Time'); % Opcional: Etiqueta del eje x
237 ylabel('Value'); % Opcional: Etiqueta del eje y
238 legend show; % Muestra la leyenda
239 hold off;
240 legend('Location', 'best');
241 grid on;
242
243
244 %% resultados
245 % Plot logged variables
246 figure(4);
247 hold on;
248 plot(log_time, data_values('U_plus'), 'DisplayName', 'u-plus');
249 plot(log_time, data_values('delta_u'), 'DisplayName', 'delta-u');
250 title('Performance de la vitesse de translation de la voiture'); % Opcional: Agrega un título
251 con el nombre de la serie de datos
252 xlabel('Temps'); % Opcional: Etiqueta del eje x
253 ylabel('Vitesse de la voiture [m/s]'); % Opcional: Etiqueta del eje y
254 legend show; % Muestra la leyenda
255 hold off;
256 legend('Location', 'best');
257 grid on;
258
259 figure(5);
260 hold on;
261 plot(log_time, data_values('U_moins'), 'DisplayName', 'U-moins');
262 plot(log_time, data_values('delta_r'), 'DisplayName', 'delta-r');
263 title('Performance de la vitesse de rotation de la voiture'); % Opcional: Agrega un título
264 con el nombre de la serie de datos
265 xlabel('Temps'); % Opcional: Etiqueta del eje x
266 ylabel('Vitesse du moteur droit [rad/s]'); % Opcional: Etiqueta del eje y
267 legend show; % Muestra la leyenda
268 hold off;
269 legend('Location', 'best');
270 grid on;
271
272 figure(6);
273 hold on;
274 plot(log_time, data_values('Ur'), 'DisplayName', 'Ur');
275 plot(log_time, data_values('Vr_f2'), 'DisplayName', 'Vr-deux-filtre');
276 title('Rponse du moteur divers signaux de tension');
277 xlabel('Temps'); % Opcional: Etiqueta del eje x
278 ylabel('Vitesse du moteur droit [rad/s]'); % Opcional: Etiqueta del eje y
279 legend show; % Muestra la leyenda
280 hold off;
281 legend('Location', 'best');
282 grid on;
283
284 figure(7);
285 hold on;

```

```

285 plot(log_time, data_values('Ul'), 'DisplayName', 'Ul');
286 plot(log_time, data_values('Vl_f2'), 'DisplayName', 'Vl-deux-filtre');
287 title('R ponse du moteur divers signaux de tension');
288 xlabel('Temps'); % Opcional: Etiqueta del eje x
289 ylabel('Vitesse du moteur gauche [rad/s]'); % Opcional: Etiqueta del eje y
290 legend show; % Muestra la leyenda
291 hold off;
292 legend('Location', 'best');
293 grid on;
294
295
296 figure(8);
297 hold on;
298 plot(log_time, data_values('Ul'), 'DisplayName', 'Ul');
299 plot(log_time, data_values('Al'), 'DisplayName', 'Al');
300 title('R ponse du moteur divers signaux de tension');
301 xlabel('Temps'); % Opcional: Etiqueta del eje x
302 ylabel('Position angulaire du moteur gauche [rad]'); % Opcional: Etiqueta del eje y
303 legend show; % Muestra la leyenda
304 hold off;
305 legend('Location', 'best');
306 grid on;
307
308 figure(9);
309 hold on;
310 plot(log_time, data_values('Ur'), 'DisplayName', 'Ur');
311 plot(log_time, data_values('Ar'), 'DisplayName', 'Ar');
312 title('R ponse du moteur divers signaux de tension');
313 xlabel('Temps'); % Opcional: Etiqueta del eje x
314 ylabel('Position angulaire du moteur droit [rad]'); % Opcional: Etiqueta del eje y
315 legend show; % Muestra la leyenda
316 hold off;
317 legend('Location', 'best');
318 grid on;
319
320
321 %% grafica
322
323 % Par metros iniciales
324 x0 = 0; % Posicion inicial en x
325 y0 = 0; % Posicion inicial en y
326 theta0 = 0; % ngulo inicial (en radianes)
327
328 % Suponiendo que tienes w, v y t definidos:
329 w= data_values('delta_r');
330 v=data_values('delta_u');
331 w=w(1:3951);
332 v=v(1:3951);
333 t = log_time(1:3951);
334 % t - vector de tiempos (s)
335
336 % Inicializa los vectores para almacenar la posicion
337 x = zeros(length(t), 1);
338 y = zeros(length(t), 1);
339 theta = zeros(length(t), 1);
340
341 % Condiciones iniciales
342 x(1) = x0;
343 y(1) = y0;
344 theta(1) = theta0;
345
346 % M todo de integraci n (Euler) para obtener la trayectoria
347 for i = 2:length(t)
348     dt = t(i) - t(i-1); % Intervalo de tiempo
349
350     % Actualiza el ngulo con la velocidad angular
351     theta(i) = theta(i-1) + w(i-1) * dt;
352
353     % Actualiza la posicion con la velocidad de traslaci n y el ngulo
354     x(i) = x(i-1) + v(i-1) * cos(theta(i-1)) * dt;
355     y(i) = y(i-1) + v(i-1) * sin(theta(i-1)) * dt;
356 end
357
358 % Graficar la trayectoria
359 figure;
360 plot(x, y, '-o');

```

```

361 xlabel('Position X (m)');
362 ylabel('Position Y (m)');
363 title('trajectoire du robot avec les poids');
364 grid on;
365 axis equal;
366 %% resultados pista
367 % Plot logged variables
368 figure(4);
369 hold on;
370 plot(log_time, data_values('U_plus'), 'DisplayName', 'u-plus');
371 plot(log_time, data_values('delta_u'), 'DisplayName', 'delta-u');
372 title('Performance de la vitesse de placement de la voiture sur le parcours'); % Opcional:
    Agrega un t tulo con el nombre de la serie de datos
373 xlabel('Temps'); % Opcional: Etiqueta del eje x
374 ylabel('Vitesse de la voiture [m/s]'); % Opcional: Etiqueta del eje y
375 legend show; % Muestra la leyenda
376 hold off;
377 legend('Location', 'best');
378 grid on;
379
380 figure(5);
381 hold on;
382 plot(log_time, data_values('U_moins'), 'DisplayName', 'U-moins');
383 plot(log_time, data_values('delta_r'), 'DisplayName', 'delta-r');
384 title('Performance de la vitesse de rotation sur le parcours'); % Opcional: Agrega un t tulo
    con el nombre de la serie de datos
385 xlabel('Temps'); % Opcional: Etiqueta del eje x
386 ylabel('Vitesse de la voiture [rad/s]'); % Opcional: Etiqueta del eje y
387 legend show; % Muestra la leyenda
388 hold off;
389 legend('Location', 'best');
390 grid on;
391
392 figure(6);
393 hold on;
394 plot(log_time, data_values('Ur')*5, 'DisplayName', 'Ur');
395 plot(log_time, data_values('Vr_f1'), 'DisplayName', 'Vr');
396 title('R ponse du moteur sur le parcours');
397 xlabel('Temps'); % Opcional: Etiqueta del eje x
398 ylabel('Vitesse du moteur droit [rad/s]'); % Opcional: Etiqueta del eje y
399 legend show; % Muestra la leyenda
400 hold off;
401 legend('Location', 'best');
402 grid on;
403
404
405 figure(7);
406 hold on;
407 plot(log_time, data_values('Ul')*5, 'DisplayName', 'Ul');
408 plot(log_time, data_values('Vl_f1'), 'DisplayName', 'Vl');
409 title('R ponse du moteur sur le parcours');
410 xlabel('Temps'); % Opcional: Etiqueta del eje x
411 ylabel('Vitesse du moteur gauche [rad/s]'); % Opcional: Etiqueta del eje y
412 legend show; % Muestra la leyenda
413 hold off;
414 legend('Location', 'best');
415 grid on;
416
417
418 figure(8);
419 hold on;
420 plot(log_time, data_values('Ul')*10, 'DisplayName', 'Ul');
421 plot(log_time, data_values('Al'), 'DisplayName', 'Al');
422 title('R ponse du moteur selon la tension sur le parcours');
423 xlabel('Temps'); % Opcional: Etiqueta del eje x
424 ylabel('Position angulaire du moteur gauche [rad]'); % Opcional: Etiqueta del eje y
425 legend show; % Muestra la leyenda
426 hold off;
427 legend('Location', 'best');
428 grid on;
429
430
431 figure(9);
432 hold on;
433 plot(log_time, data_values('Ur')*10, 'DisplayName', 'Ur');
434 plot(log_time, data_values('Ar'), 'DisplayName', 'Ar');
435 title('R ponse du moteur selon la tension sur le parcours');

```

```

435 xlabel('Temps'); % Opcional: Etiqueta del eje x
436 ylabel('Position angulaire du moteur droit [rad]'); % Opcional: Etiqueta del eje y
437 legend show; % Muestra la leyenda
438 hold off;
439 legend('Location', 'best');
440 grid on;

```

Listing 4 – Code MatLab télémetrie

## 13 Amélioration

### Références

- [1] Pololu. (s.d.). Ball caster with 1.
- [2] Seeed Studio. Grove - 12-bit magnetic rotary position sensor (as5600), 2023. Accessed : 2024-11-25.