

2017

Rapport de Projet

Mise en fonctionnement du Robot hexapode R.Hex

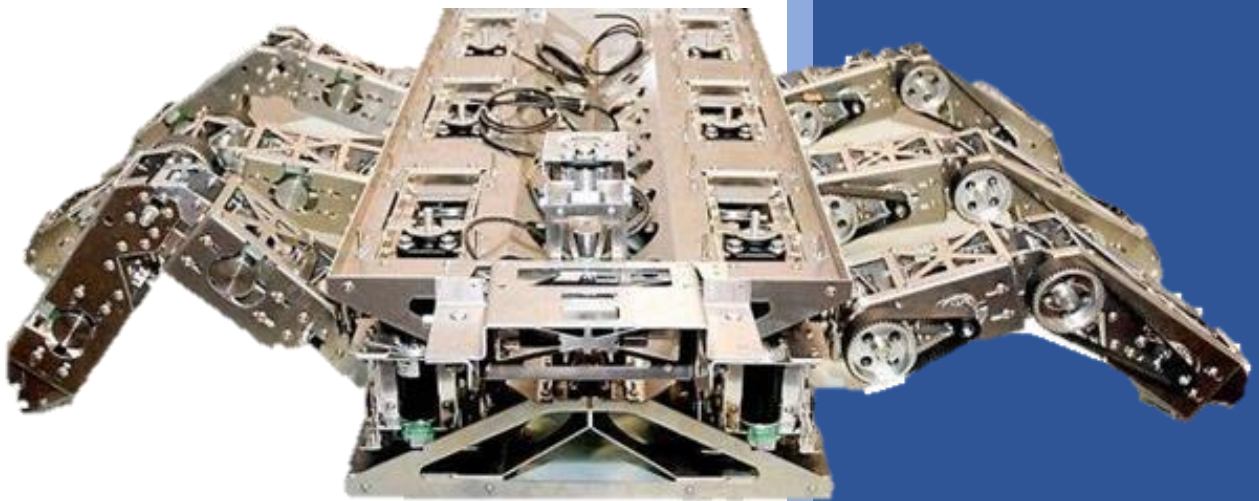


Table des matières

Introduction.....	2
Fonctionnement général.....	3
Les tours de commande	3
Les encodeurs.....	4
Communication	6
Protocole utilisé.....	6
Fonctions primaires du système	7
Simulation.....	9
Simulation du robot sur Matlab	9
Déplacement du robot sous Matlab	9
Conclusion	11

Introduction

Pour commencer, ce document n'est pas un rapport global du projet. Il rapporte seulement les nouveaux éléments qui ont été ajoutés et les problèmes rencontrés tout au long de cette troisième session de projet sur le robot hexapode RHEX. Nous sommes partis des rapports et travaux déjà existants. Vous trouverez donc joint à ce rapport les documents techniques, les schémas électriques ou encore les précédents rapports que nous avons utilisés.

Nous avons donc travaillé sur le robot hexapode RHEX. C'est l'un des projets de l'équipe EXPLORE du département robotique du LIRMM. L'objectif est de créer un robot hexapode de grande taille avec de grandes capacités motrices, sensorielles et cognitives. A l'heure actuelle, la tête du robot n'est ni montée, ni configurée. Nous avons donc seulement travaillé sur ses capacités motrices. Le Robot pèse 60kg et mesure 1m20 d'envergure. Ses 6 pattes possèdent chacune 4 moteurs qui leur donne 4 degrés de liberté. Ces caractéristiques imposent certaines contraintes que nous verrons plus tard dans ce document.

Nous avons divisé notre projet en 3 grandes parties. Chaque partie avait ses propres objectifs indépendamment des 2 autres.

- La partie mécanique : L'objectif principal était d'assurer la mise en place du robot afin qu'il soit prêt à l'emploi. Il fallait donc valider et tester chaque carte du robot puis monter les cartes de commande pour chaque patte du robot.
- La partie communication : L'objectif était de choisir et mettre en place un système de communication entre un maître et les 6 pattes du robot avec un système temps réel.
- La partie Simulation : Avec le logiciel MATLAB, nous avons pour objectif de simuler le robot hexapode afin de définir ses contraintes et de mettre en place un algorithme de marche.

Vous trouverez également joint à ce document le programme actuel du robot ainsi que le programme MATLAB commenté permettant la simulation du robot.

Fonctionnement général

Les tours de commande

Pour commencer, le premier objectif était de rendre la structure du robot opérationnelle. Le câblage reliant les moteurs à chaque extrémité des pattes a alors été réalisé par un étudiant extérieur à l'école. De notre côté, nous avons fabriqué et assemblé les 6 tours de commande du robot.

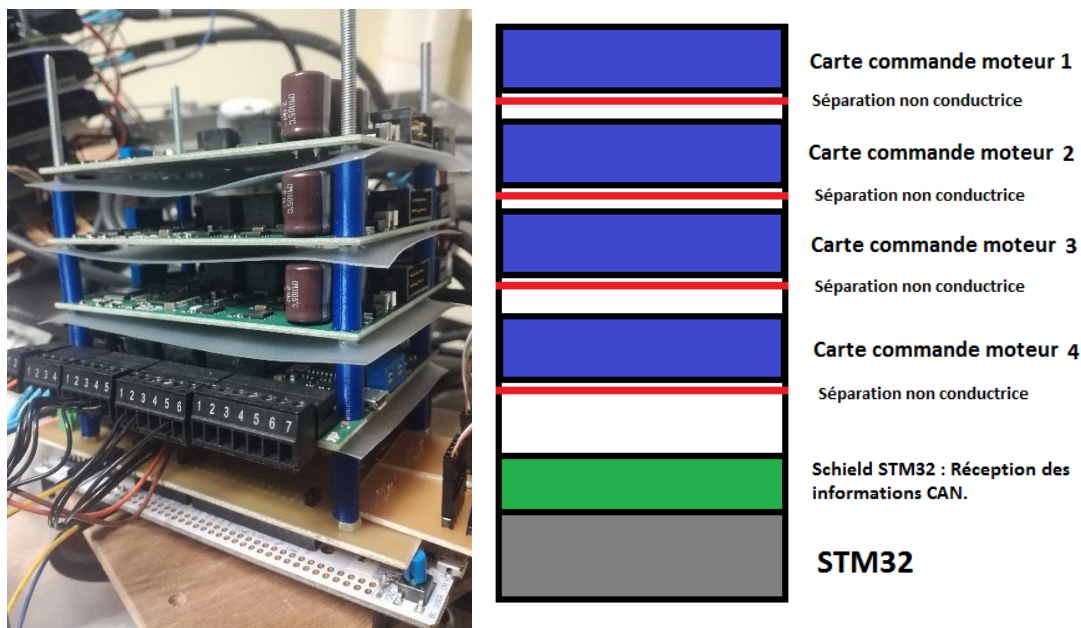


Figure 1 Photo et schéma d'une des 6 tours de commande

Alimentation des cartes :

- Les « tours » sont alimentées en 24V, il faut prévoir au moins 1.5A par pattes. Nous avons constaté ce maximum alors qu'une patte soulevait le robot.
- La STM est alimentée en 5V par la carte, au travers d'un traco-power.
- La STM envoie les ordres sur les cartes de commande, par le connecteur encadré en jaune. Les cartes n'envoient pas de consignes aux moteurs sans que le pin 2 et le pin 6 soient reliés. Les cartes de contrôle sont alimentées en 24V par le connecteur encadré en rouge. Les ordres de la STM sont envoyés dans une PWM à 10kHz (period_PWM). Les cartes de contrôle clignotent rouges si la PWM n'est pas bien envoyée et/ou que les pins 2 et 6 ne sont pas reliés.
- Les cartes de contrôles sont liées aux moteurs par le connecteur encadré en vert.

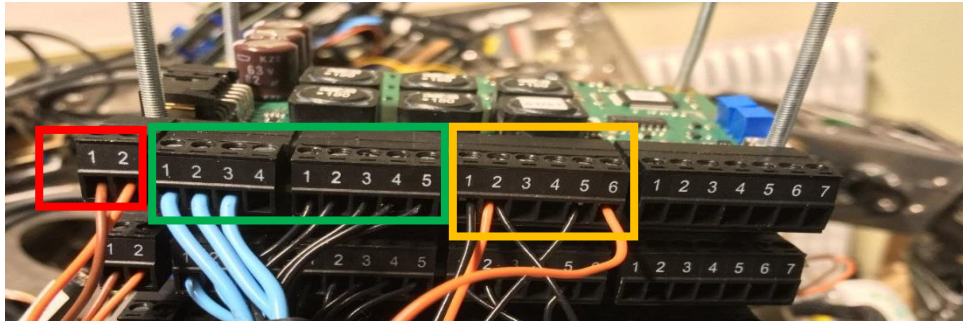


Figure 2 Photo, zoom sur les branchements de chaque carte de la tour

Les encodeurs

Le fonctionnement des encodeurs grâce à l'utilisation des bus CAN a déjà été très largement expliqué dans les 2 rapports précédents. Nous avons donc compris et utilisé ce qui avait été fait. A présent, tous les encodeurs du robot sont configurés, il est donc inutile de reprogrammer cette partie-là. Chaque composant lié au CAN a un nom unique facilitant la lecture des données.









Remarque : La configuration des encodeurs nécessite une fréquence différente à celle permettant leur utilisation sur le robot.

Les problèmes qu'il est possible de rencontrer sont listés dans le tableau suivant. Pendant notre projet, nous nous sommes heurtés à deux problèmes. Premièrement l'alimentation (tension insuffisante), et la fonction du mode pré-operationnel que l'on exécutait mal.

greenLED = BUS State
red LED = ERR display



Figure 3 LED/Témoin sur chaque composant affichant l'état de marche

Annunciator	LED	Description	Cause of error	Addendum
Bus OFF		No connection to the Master ²	Data transmission line break Incorrect baud rate Inverted data line	Observe combination with ERR LED If ERR LED is also OFF, please check power supply ³
Bus flashing ca. 250ms		Connection to Master Pre-operational state		SDO communication
Bus flashing ca. 1sec		Connection to Master Stopped state		SDO communication not possible Only NMT commands
Bus ON		Connection to Master Operational state		PDO Transfer is active
ERR OFF		Device working normally		Observe combination with BUS LED
ERR flashing		Connection to Master interrupted	Combination with BUS status	BUS LED green, flashing or ON - is dependent on Object 1029h Error Behaviour
ERR ON		BUS OFF State	Short circuit on the Bus or Incorrect baud rate	
ERR +Bus flashing		LSS-Mode	LSS Mode Global selected	Device is waiting for a LSS-command

The individual LED annunciators can of course also occur in combinations.

² The Master can be a PLC or a second communication partner.
³ Operating voltage

Figure 4 Tableau récapitulant l'ensemble des erreurs pouvant être détectées

Communication

Dans le but de pouvoir contrôler les 6 pattes de manière cohérente et donc de les synchroniser entre elles nous avons décidé de connecter l'ensemble de nos microcontrôleurs à un ordinateur en utilisant le port Ethernet disponible. Nous avons donc décidé du protocole de communication à utiliser ainsi que l'ensemble de fonctions de bases à définir pour pouvoir déplacer notre robot.

Protocole utilisé

Le fait d'utiliser le port Ethernet pour communiquer avec le maître permet de rester dans un protocole de communication assez répandu pour pouvoir changer de support dans le futur pour un module plus embarqué, vers une autre Nucleo-144 par exemple.

Afin de s'assurer de la bonne réception des messages envoyés, nous avons privilégié le protocole TCP par rapport à l'UDP. Sans cela, une erreur de commande sur l'un de nos moteurs pourrait être catastrophique. Ci-dessous le schéma de communication de notre robot. On note que les différentes STM ne peuvent communiquer entre elles que par l'intermédiaire de l'ordinateur.

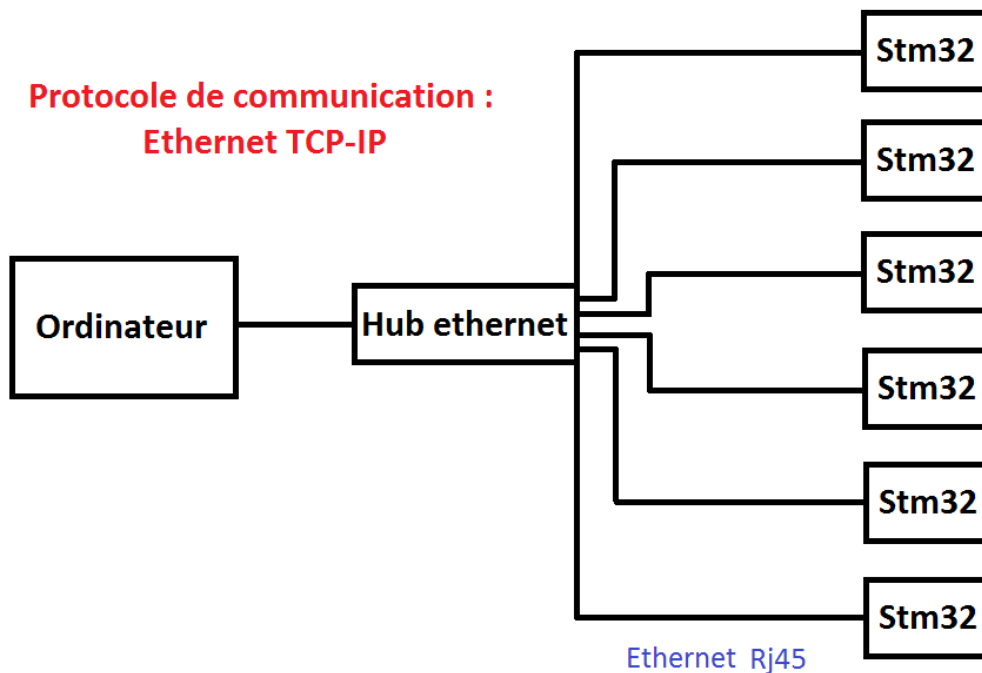




Figure 5 Schéma du protocole de communication imaginé pour le robot

Fonctions primaires du système

L'objectif était d'implanter un programme avec un Os temps réel dans l'ordinateur. Nous avons donc listé toutes les fonctions primaires que devait pouvoir réaliser le robot.

Entité	Fonctions	Explication
 Ordinateur	Init_Communication()	Initialisation du protocole de communication.
	Read_motors_all()	Demande la valeur actuelle des 24 angles de Rhex.
	Read_motors(i)	Demande la valeur actuelle des angles de la patte (i)
	Stop_motors_all()	Arrête le Robot dans sa position actuelle. (PWM = 50%)
	Reception(IP, message)	Reçoit un message provenant d'une patte (Adresse IP correspondante) et traduit l'information envoyée.
	Send(IP, message)	Envoie une commande à l'une des pattes. (Adresse IP correspondante).
	Command_position(i, x, y, z)	Commander en position la patte i.
	Command_angle(i, θ1, θ2, θ3, θ4)	Commander en angle la position de la patte i.
	Command_cycle(distance)	Commander le robot par une distance à parcourir. Il déterminera le nombre de cycle à effectuer.
	Command_cycle(time)	Commander le robot en ligne droite pendant un temps. Une fois ce temps atteint, le robot se fixera dans sa position actuelle (PWM = 50%).
	Processing(actual_angles)	Détermine les points suivant à atteindre pour les pattes à partir de sa position actuelle.
 STM32F429	Read_motors()	Lis les angles des 4 moteurs qu'il commande.
	Stop_motors()	Envoie une PWM de 50% sur ses 4 moteurs dans le but d'arrêter le robot.
	Send(message)	Envoie un message à l'ordinateur. Adresse IP fixe donc non spécifié en argument.
	Reception(message)	Reçoit une commande provenant de l'ordinateur.
	Processing(actual_angles)	Détermine les points suivant à atteindre pour les pattes à partir de sa position actuelle.
	Set_motors(m1, m2, m3, m4)	Envoie la nouvelle valeur de PWM dans chaque moteur de la patte.
	Cut(message)	Découpe le message reçu dans le but de l'interpréter.
	Reset()	Définit la position actuelle de la patte comme point d'origine.

En partant de ce tableau de fonctions, il est possible de créer toutes les tâches nécessaires au déplacement du robot. Par exemple, ci-dessous la fonction permettant de lire les angles d'un bras pour une STM.

```
READ_MOTORS()  
  
Reception(IP, message);  
Cut(message);  
Angles = Read_motors();  
Send(IP, Angles);
```

Changement d'un encodeur : On note que tous les encodeurs ont été préconfigurés. Dans le cas d'une remise à zéro de l'un d'eux ou d'un changement. Vous trouverez, joint à ce document, un programme permettant leur initialisation. Il faudra faire correspondre le nom écrit sur l'encodeur que vous changez grâce au bouton bleu. Vous pouvez vous servir du logiciel « putty » pour savoir quel nom lui sera attribué. Après cela, au lancement du programme de base vous pourrez définir le point d'origine de la patte et donc de l'encodeur par une simple pression sur le bouton bleu. Le bras doit être complètement à l'horizontale, parallèle au sol.

Problème rencontré : Seul un programme sans OS en temps réel est opérationnel. En effet, afin de mettre en place ce nouveau système de communication l'utilisation du logiciel CubeMX nous a semblé obligatoire. La création des bibliothèques permettant l'utilisation de free RTOS, de la communication Ethernet et des bus CAN nous était impossible au vu de nos compétences actuelles et du temps qu'il nous restait. Il y a donc un gros travail restant sur la transformation des bibliothèques originales pour qu'elles s'adaptent à cubeMX ou au moins à Free RTOS (l'OS temps réel utilisé).

Le programme actuel, sans OS temps réel, que vous pouvez trouver joint à ce document, permet une marche synchronisée entre toutes les pattes mais il n'y a aucune communication entre chaque patte. Il est donc indispensable de mettre en place l'OS en temps réel afin de remplir les objectifs initiaux de ce projet.

Simulation

Nous avons simulé le robot R.hex sur Matlab afin de pouvoir tester notre algorithme de marche sans prendre de risques avec le vrai robot et pouvoir utiliser la puissance de calcul du logiciel. Ci-joint à ce document le fichier Matlab permettant la simulation complète du robot, commenté. Afin de faciliter votre compréhension et de le réutiliser le plus efficacement possible sans perdre de temps. Une brève explication de la simulation a été réalisée en plus des commentaires sur le programme.

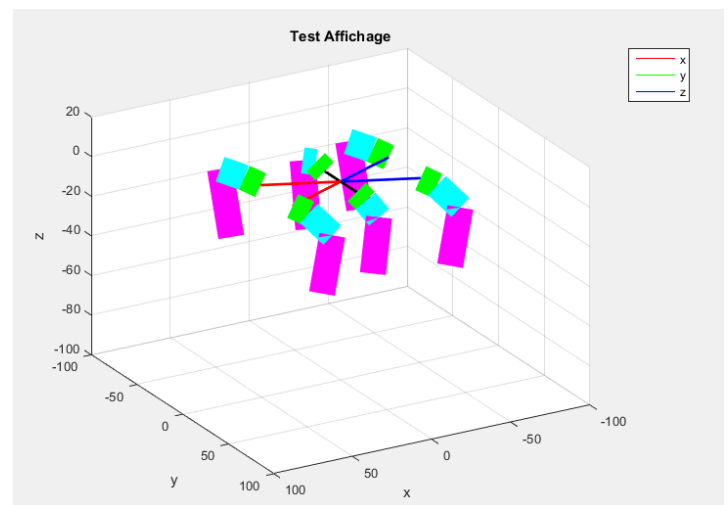


Figure 6 Le robot hexapode simulé sur Matlab

Simulation du robot sur Matlab

Pour commencer, Une première simulation Matlab avait déjà été effectuée. Nous avons commencé à travailler à partir de celui-ci. La première étape de la simulation est de créer le robot sur Matlab. Il a donc été séparé en 2 entités, le corps et les pattes. Il a été décidé de représenter le corps sous la forme d'une étoile comme le montre la figure 6. Le robot étant symétrique, on fait partir chaque patte d'un bout de l'étoile. Cela permet également de définir un point central du robot (centre de l'étoile), On pourra alors suivre l'évolution du déplacement du corps du robot. Les dimensions ainsi que toutes les informations concernant le corps sont inscrites dans la fonction « Create_Corp ». Pour un souci de simplicité, on ne fait pas bouger le corps dans l'affichage du robot quand celui-ci effectue l'algorithme de marche. De la même façon les informations sur les pattes sont stockées dans la fonction « create_patte ». Pour afficher le robot on utilise logiquement la vue 3D que propose Matlab et on réalise ses mouvements grâce à la fonction « Patte_dessine ». Cela dessine la patte donnée en argument.

Déplacement du robot sous Matlab

Pour réaliser le déplacement du robot, des fonctions de base ont été faites, les modèles géométriques directs et indirects, les changements de repère etc...).

- La fonction « MGD » prend 4 arguments en entrée : Patte_L, Patte_R, angle_D, angle_G, les deux premiers concernent les informations sur l'état actuel des pattes et les deux derniers sur les angles des 3 pattes de droite et de gauches. Une fois la fonction réalisée, elle renvoie les nouvelles valeurs de Patte_L, Patte_R en prenant compte des angles que l'on a donné. Pour cela la fonction « MGD3 utilise les fonctions « trans », « rot » et « jTj » permettant les calculs des nouveaux repères.

- Le MGI qui est en fait la fonction « déplacement_pas » prend un point de départ et d'arrivée en argument. Son objectif est de faire un pas jusqu'au point d'arrivée. Chaque appel de cette fonction réalisera un pas. Nous avons décidé de faire des pas de la forme d'une ellipse dans le but d'éviter aux pattes de frotter le sol. Un pas représente donc tout le tour d'une ellipse. Grâce à l'équation de cette forme, nous sommes capables de déterminer le prochain point à atteindre pour le robot. La fonction MGI pourra alors déterminer les angles de chaque moteur à avoir pour atteindre ce nouveau point.

- L'algorithme de marche que nous avons choisi est le triangle inversé. Il y a 2 groupes de 3 pattes qui tournent dans le même sens et à la même vitesse avec un déphasage de π . Chaque groupe est composé des pattes aux extrémités d'un côté et la patte du milieu de l'autre. On voit bien sur le schéma ci-dessous les 2 groupes de pattes synchronisés. (Groupe 1 : bleu, groupe 2 ; rouge).

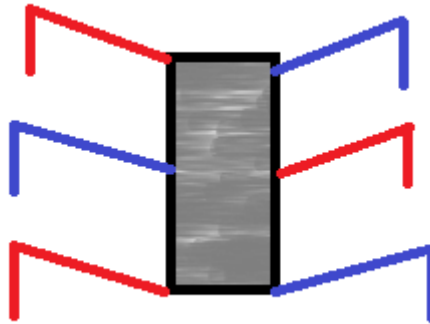


Figure 7 Schéma du robot montrant les 2 groupes de pattes synchronisées

Le décalage de π entre les deux groupes donne donc une différence d'une demi ellipse. Enfin pour vérifier que le déplacement s'effectue sans problèmes de collision ou dus aux contraintes physiques du robot, on affiche un tableau comprenant l'évolution de chaque angle du robot.

- Dans ce modèle, simulé et réel, les contraintes articulaires ont été définies telles que chaque patte a son espace de mouvement. Cela assure qu'aucune des pattes n'entrera en collision tant qu'elle reste dans son espace. Une amélioration possible est d'adapter les contraintes par rapport aux autres pattes qu'il l'entoure afin de permettre un mouvement plus ample et donc un déplacement plus rapide.

- Pour finir, il est important de signaler que dans un souci de simplicité, Seul 3 des 4 moteurs sont utilisés pour faire un mouvement. En effet le moteur permettant la rotation complète de la patte (moteur 2) a été jugé inutile dans une situation simple par les étudiants ayant travaillé sur Rhex avant nous. On fixera donc la valeur de ce moteur.

Conclusion

L'objectif initial du projet était de simuler puis de rendre le robot hexapode R.hex opérationnel. Pour cela plusieurs objectifs ont été définis. La simulation du robot sur Matlab est terminée, le robot suit un algorithme de marche avec ses 6 pattes synchronisées. De son côté le vrai robot a été complètement câblé et pourvu de tours permettant la commande des moteurs via des STM32f429. Tous les composants du robot ont été configurés, encodeurs et cartes de commande. Chaque STM contient un programme permettant la réalisation d'un mouvement suivant une ellipse par la méthode des triangles inversés. Pourtant le projet est encore loin d'être terminé. Il reste quelques détails mécaniques comme la création de boîtier permettant d'accueillir les tours de commande afin de gagner en propreté. Mais surtout une grosse partie programmation pour mettre en place l'OS temps réel et le protocole de communication. Pour finir, chaque patte utilise deux bus CAN. Aussi, il serait intéressant de limiter le nombre de bus dans la mesure où la technologie utilisée permet en théorie de joindre l'ensemble des capteurs du robot sur un seul bus CAN.

Ce projet a été très enrichissant pour tout le groupe. Ayant un groupe composé de 5 personnes qui ont choisi l'option robotique, cela nous a permis d'appréhender les problèmes dans un projet de robotique auxquels nous pourrions être confrontés. Ce projet à l'avantage de toucher à plusieurs domaines, la simulation d'un robot sur Matlab, la programmation sur microcontrôleur, la mise en place de circuits électroniques, l'organisation ou encore le travail d'équipe... Les axes de travail étaient nombreux, le partage des tâches a donc été la clé de ce projet.

Nous restons à disposition pour les futurs groupes qui travailleront sur ce robot. Nous serons ravis de pouvoir répondre à vos questions dans l'espoir de voir le robot fonctionner à son plein potentiel. Nous tenons à remercier Mr Dubreuil pour tout le temps passé avec nous sur ce projet, ainsi que Mr Zapata et Mr Latorre qui ont accepté de répondre à nos nombreuses questions.

