Projet de fin d'études

Intégration d'une carte d'acquisition et de commande dans un véhicule autonome

Réalisé par : Benoit CONFLAND
Ismaïl EL HASNAOUI

Tuteur : Mr Rochdi MERZOUKI
Année : 2012/2013

Remerciements

Nous tenons à remercier particulièrement Mr MERZOUKI, Professeur des universités à Polytech'Lille pour le soutien, les idées fournies, mais aussi pour nous avoir donné l'opportunité de travailler sur un projet de recherche orienté systèmes embarqués.

Nous remercions aussi Mr Vincent COELEN, ingénieur IMA et thésard en robotique médicale pour son soutien technique tout au long du projet.

Nous remercions également Mr Michel POLLART, technicien au laboratoire LAGIS, pour son aide précieuse et pour le travail qu'il a réalisé concernant la partie mécanique et électrique du Robucar.

Sommaire

Introduction

Dans le cadre de notre projet de fin d'études, nous avons choisi le sujet : **«** Intégration d'une carte d'acquisition et de commande dans un véhicule autonome ». Ce projet de fin d'études entre dans le cadre du projet européen InTraDE (Intelligent Transportation for Dynamic Environment, www.intrade-nwe.eu), dont le but est de concevoir un système de transport intelligent permettant d'acheminer du fret à l'intérieur d'espaces confinés portuaires. L'objectif est d'améliorer la gestion du trafic, et d'optimiser l'espace de travail. Le développement d'un système de transport intelligent a été envisagé pour répondre à ces besoins : un IAV (Intelligent Autonomous Vehicule).

Polytech'Lille a récupéré un châssis de RobuCar, provenant de l'université de Nancy. Ce châssis est cependant incomplet : les blocs de puissance avant et arrière sont absents, il n'y a aucune carte d'acquisition et de commande.

Notre projet cible l'action WP4A11, autour de la transférabilité du mode de transport. L'objectif est de développer des IAV dans un cadre extra maritime. Le projet consiste à remettre en état ce châssis pour qu'à terme, il puisse de nouveau rouler avec la flotte de l'InTraDE. En effet, il servira à la mise en place d'un train de RobuCars. L'objectif étant dans le futur d'utiliser ces véhicules pour le transport de personnes sur le campus de l'université Lille1, de façon autonome et sûre.

Présentation du projet

1. Problématique

Notre objectif est d'automatiser un RobuCar avec une seule intelligence embarquée de contrôle et de supervision. La nouveauté par rapport aux autres RobuCars réside dans la carte d'acquisition et de commande unique. Le RobuCar est un véhicule qui possède à l'origine deux powerPC : ces derniers commandent chacun un train. La communication entre les trains avant et arrière se fait par bus CAN. Pendant leurs travaux les chercheurs du laboratoire LAGIS se sont rendu compte que cette communication était source de problèmes et que la robustesse des powerPC était limitée.

2. Cahier des charges

Le système doit être embarqué, doit permettre de contrôler les entrés et sorties de manière sûre. C'est à dire que le système doit être fiable et sécurisé pour les utilisateurs.
Il ne doit y avoir qu'un seul système pour commander la traction (4x4) et la direction (2x2). Le système doit être capable d'acquérir les données, de les traiter, pour pouvoir commander le RobuCar. La supervision du véhicule est également attendue.

3. Présentation du Robucar

Le Robucar est un véhicule électrique autonome destiné à évoluer dans des milieux sains ou hostiles afin d'accomplir des missions spécifiques (Transports, explorations,…). Il est commercialisé par la société française ROBOSOFT qui est spécialisée dans les solutions de robotique. Le laboratoire LAGIS dispose de plusieurs véhicules de ce type. La principale caractéristique de cette gamme est les deux trains identiques et indépendants. Chaque train comporte :

- Deux roues entraînées chacune par un motoréducteur électrique 48V équipé d'un frein électromagnétique à manque de courant.
- Deux codeurs incrémentaux.
- Une suspension à deux triangles superposés.
- Une commande de braquage avec un vérin électrique 48V.
- Un codeur absolu communicant à travers une liaison série de type SSI.
- Un bloc de puissance incluant deux variateurs de tension programmables.

Une description plus détaillée des composantes du véhicule est donnée en annexe.

4. Solution proposée

Nous avons choisi de mettre en place une carte Dspace 1103 afin d'offrir une meilleur robustesse. Ainsi la mise en place d'une telle carte d'acquisition et de commande permettrait de résoudre ces soucis.
De plus, le contrôleur est entièrement programmable à partir de l'environnement Matlab/Simulink en utilisant des libraires spécifiques.
L'entreprise fournit également un logiciel d'interface graphique -ControlDesk- pour commander et superviser le système en temps réel.

4.1. Présentation de la dSPACE

La dSPACE est un contrôleur temps réel, puissant et robuste pour le prototypage de lois de commande et plus particulièrement pour des applications telles que les contrôleurs automobiles, les contrôleurs de suspension active ou encore dans la robotique.

Elle est équipée d'un PowerPC d'une fréquence de 400Mhz et d'une mémoire de 128 MB.
Le module d'E/S quant à lui est constitué de :

- 36 canaux A/D
- 8 canaux D/A
- 50 I/O
- 6 interfaces incrémentales
- 1 port CAN
- 4 ports série RS232/RS422
- 12 sorties PWM

Figure 1 : dSPACE 1103

Figure 2 : Carte d'E/S

L'architecture interne d'une dSPACE 1103 est détaillée en annexe.

4.2. Présentation de dSPACE ControlDesk

ControlDesk est un logiciel d'instrumentation et d'expérimentation qui nous permet lors de ce projet de développer des interfaces graphiques pour la commande en temps réel du système géré par le contrôleur dSPACE. En effet l'interface développée nous permet à la fois d'envoyer des consignes directement au système et de visualiser les différentes grandeurs mesurées et tout cela en temps réel.

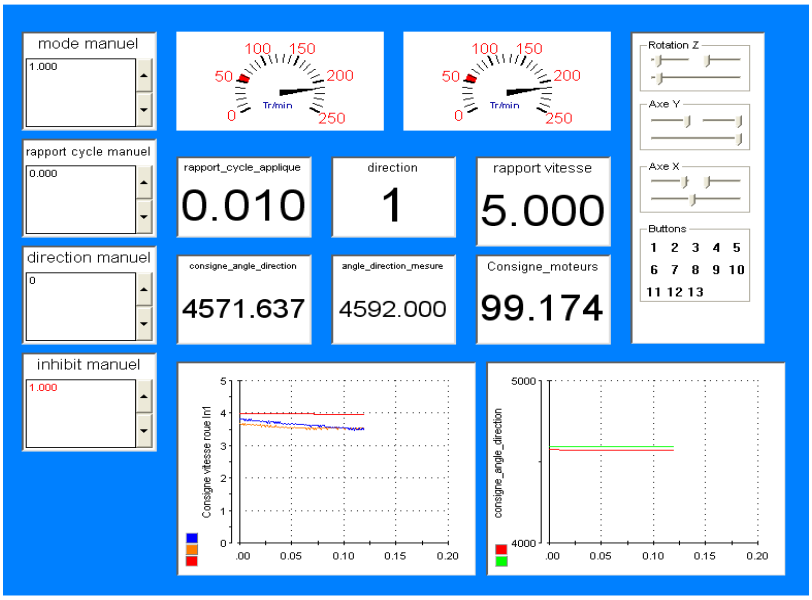


Figure 3 : Exemple d'interface de commande

Travaux effectués

Etapes d'analyse et de développement

En Septembre, nous avons pris connaissance des membres de l'équipe avec laquelle nous allions travailler. Nous avons également pu voir à quoi devrait ressembler notre projet une fois fini grâce à eux car ils travaillaient également sur un RobuCar. Notre véhicule avait une particularité, il n'avait pas de blocs de puissance. Ainsi, il fallait les reconstruire ! Nous nous sommes donc intéressés dans un premier temps à la dSPACE 1103. Cette carte d'acquisition et de commande nous était totalement inconnue, en attendant de pouvoir faire des tests sur le véhicule réel, nous nous sommes attardés sur les documentations de la dSPACE et des variateurs, et différents codeurs pour prévoir la commande. Nous avons notamment suivi des tutoriels pour comprendre l'utilisation du logiciel ControlDesk (interface graphique en temps réel). Ainsi, nous avons cerné les capacités de la dSPACE, notamment en terme d'entrés/sorties, les différents types de communications possibles... Nous avons créé un fichier Simulink pour retrouver les différents blocs particuliers à la dSPACE.

Dans un second temps, nous le verrons plus tard, nous avons rencontré des soucis au niveau du câblage. Nous avons donc de nouveau du replonger dans les documentations des variateurs pour pouvoir identifier chaque fil et corriger les erreurs. Lorsque le boîtier a été opérationnel, nous avons également du récupérer les données du codeur absolu à travers un Arduino ATmega 2560. Nous nous sommes également documentés sur internet pour trouver des solutions (l'Arduino étant open source).

1. Commande des moteurs

L'objectif de cette partie est de réaliser la commande des moteurs du train arrière dans un premier temps, puis dans un second temps étendre cette commande au train avant. Pour ce faire, nous procéderons en deux parties. Dans un premier temps, nous nous concentrerons sur l'acquisition et la commande du moteur. Puis dans un deuxième temps, nous nous intéresserons à la régulation en vitesse des moteurs.

1.1 Commande et acquisition

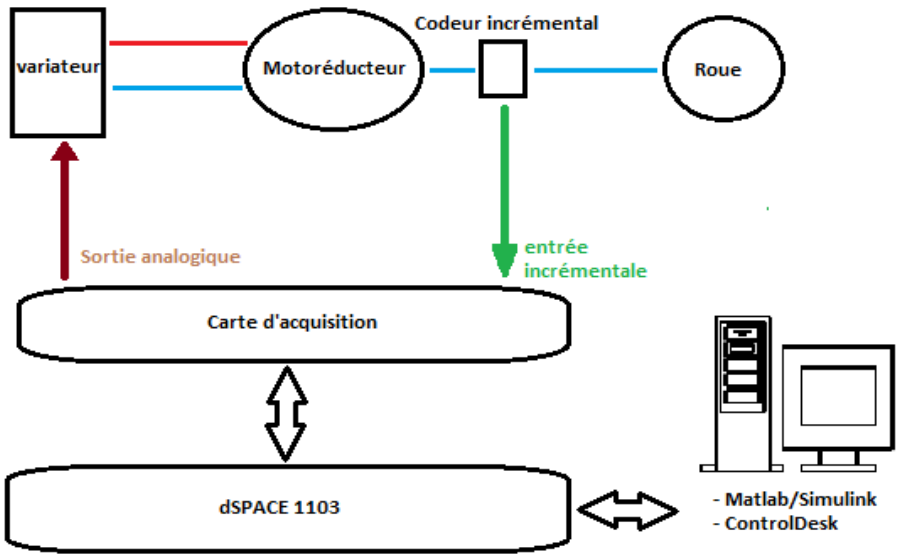


Figure 4 : Schéma de principe

L'architecture de commande d'un moteur est représentée dans la figure ci dessus. Nous programmons sous Matlab/Simulink, nous chargeons le programme dans la Dspace 1103 qui envoie un signal analogique au variateur via son interface E/S. Le variateur envoie la consigne au moteur. Nous pouvons ensuite récupérer la position du moteur grâce à un capteur incrémental GI338 relié directement à la carte d'acquisition Dspace.

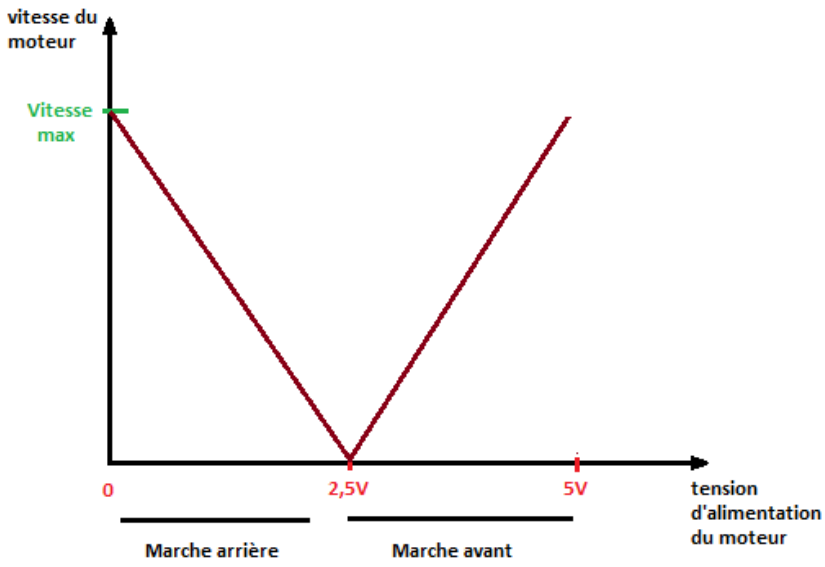


Figure 5 : Caractéristique moteur

Le moteur a une caractéristique particulière au niveau de sa tension d'alimentation (cf figure 5). En effet, la tension envoyée au moteur est toujours positive. En fonction de sa valeur, le moteur tourne dans un certain sens.

Pour ne pas mettre en erreur les variateurs, il faut au démarrage envoyer une consigne de 2,5V. Cela correspond à une vitesse nulle. Il faut savoir que le moteur est prévu pour recevoir une tension maximum de 5V. Or la dSPACE est capable de fournir des signaux analogiques supérieurs à 10V. Ainsi pour protéger le moteur, nous avons limité l'amplitude du signal analogique à 5V.

Nous avons choisi de commander notre moteur en pourcentage. L'utilisateur fait varier la pédale entre 0 et 100%, ensuite le signal est converti en consigne de vitesse puis finalement, cette consigne est ramenée entre 0 et 2,5V (marche avant) ou 2,5V et 5V (marche arrière) en fonction du choix de l'utilisateur (présence d'un bouton « marche arrière »).

La valeur retournée par le codeur incrémental est en ppm (pulse per minute). Nous savons qu'en un tour de moteur, il y a 500 pulses. Ainsi nous pouvons obtenir le nombre de tours du moteur. Nous connaissons également le rapport du réducteur qui vaut 1/13. Le nombre de tours de roue est donc connu, en dérivant cette valeur on obtient la vitesse en tr/s. Nous pouvons donc avoir un retour pour savoir si la consigne est bien suivie.

Lors des premiers tests, nous nous sommes aperçu que la consigne de vitesse n'était pas atteinte. De plus, les deux roues ne tournaient pas à la même vitesse. Nous avons donc mis en place la régulation des roues en vitesse.

1.2 Régulation en vitesse

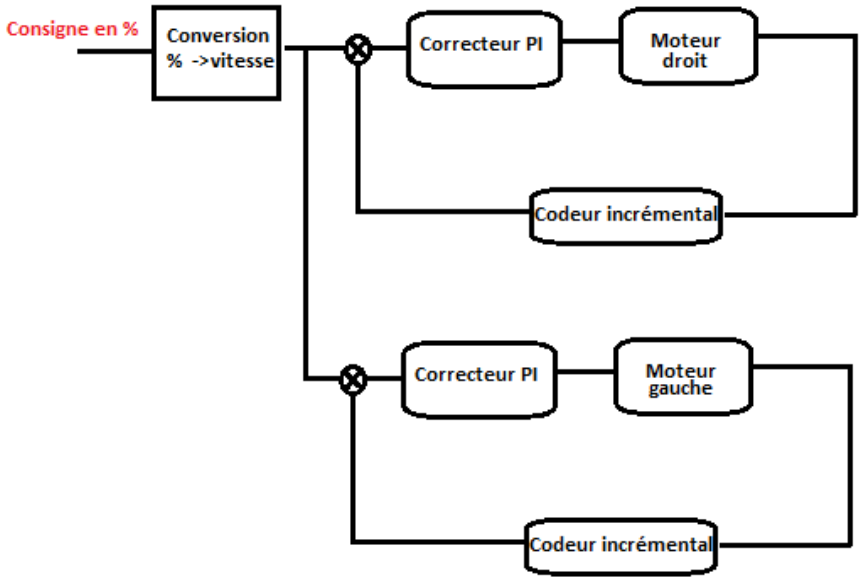


Figure 6 : Schéma de régulation des moteurs en vitesse

Nous pouvons voir ci dessous le schéma de régulation. Nous avons choisi de réguler en vitesse, car nous avons un capteur incrémental qui nous permet de l'obtenir. Nous avons donc placé un bloc PID pour corriger l'erreur statique. Il s'agit en fait d'un correcteur proportionnel intégral (PI). Nous avons réglé les paramètres du PI expérimentalement, afin d'obtenir un comportement adéquat pour une roue.

Nous envoyons la même consigne aux deux moteurs du train arrière, et grâce aux capteurs incrémentaux de chaque moteur, on peut réguler chaque roue (cf figure ci dessus).

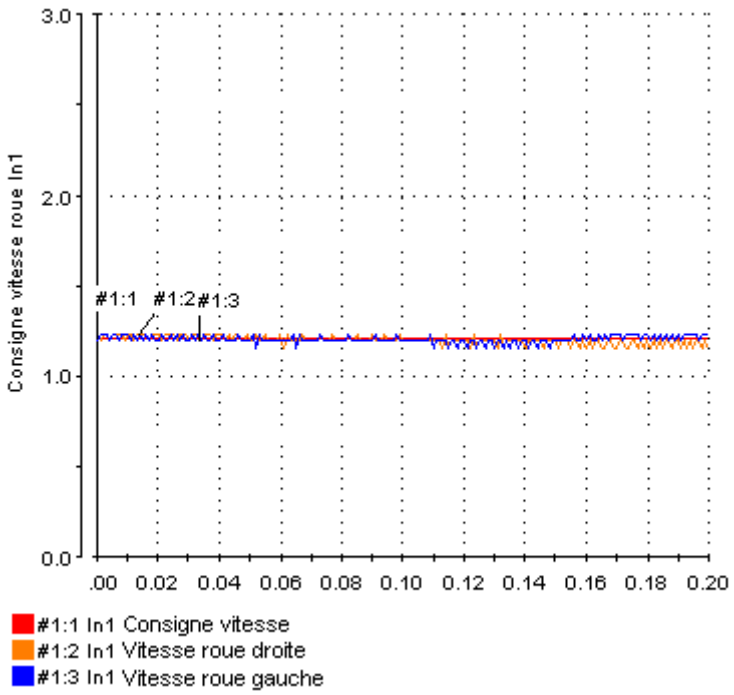


Figure7: Réponse de la vitesse

On peut voir sur la courbe que la consigne est bien atteinte et qu'il n'y a plus d'erreur statique. Les oscillations observées sur la figure ne se voient pas sur le comportement des roues. Expérimentalement, nous voyons que les deux roues tournent à la même vitesse.

Après avoir validé la commande et la régulation sur le train arrière, nous avons réalisé la commande du train avant en se basant sur celle que nous avons réalisé pour le train arrière.

2. Commande de la direction

2.1 Commande et acquisition

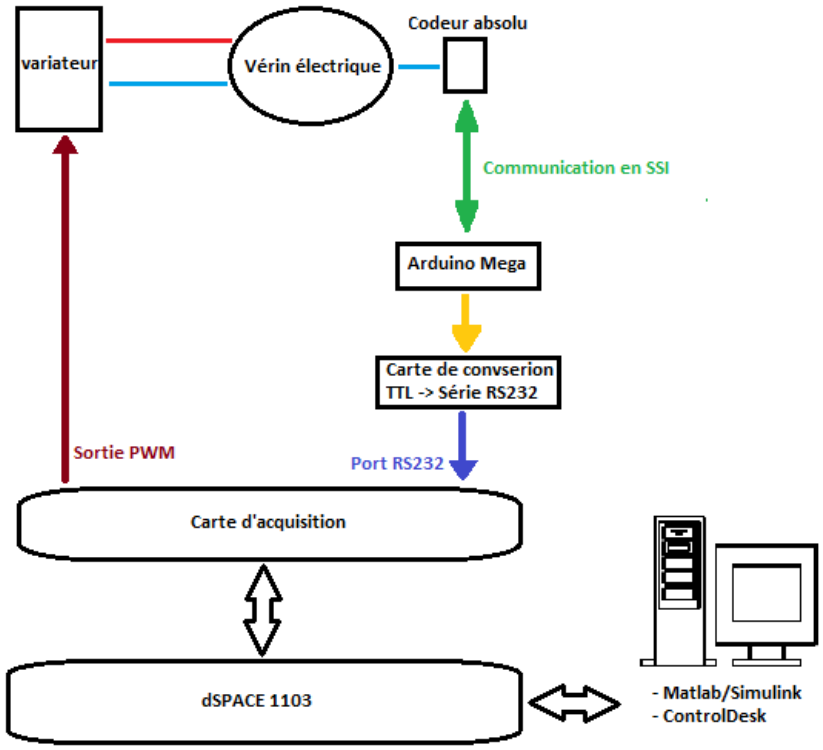


Figure 8 : Schéma de principe de la commande d'un vérin de direction

La commande pour la direction se fait comme pour celle du moteur à quelques petits détails prêts. En effet, la vitesse de déplacement se fait avec un signal PWM envoyé au variateur. De plus le capteur absolu GA240 à la particularité de communiquer en SSI (*Synchronous Serial Interface*). Or la carte d'E/S ne possède pas de port SSI. Ainsi, nous devons récupérer le signal retourné par le capteur absolu, en extraire la position et l'envoyer à la dSPACE à l'aide d'un circuit intégré adéquat (MAX232N), on réalise la conversion des signaux TTL de l'Arduino en série RS232.

2.2 Programme Arduino

Nous allons ici vous décrire l'algorithme de notre code (cf ANNEXE). Dans un premier temps, il faut initialiser les registres correspondant à la communication série. En effet, la dSPACE communique d'une certaine façon (communication réglable). Il faut donc accorder les deux communications pour qu'elles puissent communiquer normalement. Nous avons choisi de communiquer selon une liaison série en 8bits, avec un bit de stop et sans parité.

L'algorithme se compose en trois phases : acquisition des données brutes du codeur

(fonction *getPosition()*), transformation de ces données en valeurs décimales (fonction *transformation()*) et enfin envoi des données sur le port série (fonction *sendPosition()*).

getPosition() :

On initialise deux tableaux d'entiers, un pour chaque train. Ensuite on fait la lecture des codeurs en utilisant la même horloge : à chaque état bas, on range la valeur des pins correspondant aux datas des capteurs dans les tableaux. On récupère des tableaux de 14bits.

Transformation() :

Les données dans les tableaux sont des entiers, pas des booléens. Ainsi, on transforme les tableaux d'entiers en tableaux de booléens. Le codeur absolu est basé sur 13bits or nous avons des tableaux de 14bits, le bit en plus représente un bit de start. Il faut décaler d'un bit pour avoir les bonnes valeurs du codeur. La fonction retourne les bonnes valeurs de position.

SendPosition() :

Les valeurs obtenues en décimal sont codées sur 13bits. Or lors de la communication série, on écrit sur 8bits. Ainsi, nous avons choisi de diviser la position en deux octets que nous allons envoyer séparément : $byte1 = \frac{position\ avant}{256}$ et $byte2 = position\ avant - byte1 * 256$

Pour pouvoir différencier les valeurs du train avant de celles du train arrière, nous avons utilisé un bit de marquage : nous ajoutons 32 en décimal au byte3 (donc pour le train arrière) :

$$byte3 = \frac{position\ arrière}{256} + 32 \quad \text{et} \quad byte4 = position\ arrière - byte3 * 256$$

Nous avons fait un code sous Simulink permettant de récupérer les octets et de reconstituer les bonnes valeurs des codeurs avant et arrière.

2.3 Régulation du vérin en position

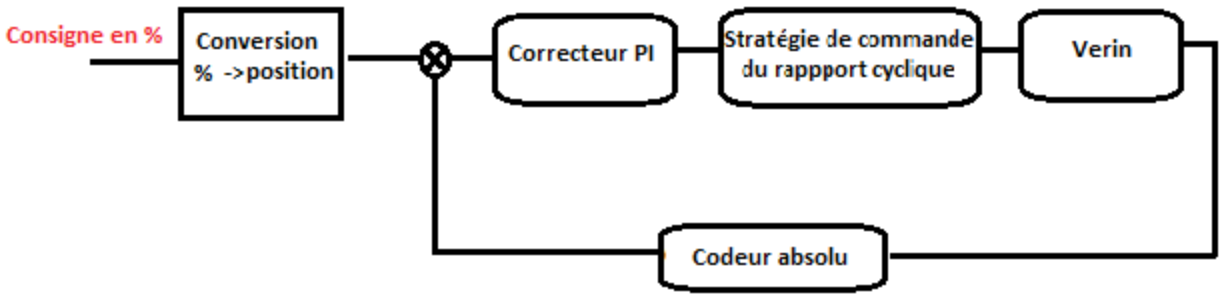


Figure 9 : schéma de la régulation de la direction

Nous pouvons voir le schéma de régulation, on remarquera que l'on a également choisi de commander en pourcentage. On convertit cette consigne en position pour pouvoir réguler. Nous avons réglé le PI expérimentalement, dans le but d'avoir une direction sans à-coup. Le bloc de stratégie de commande du rapport cyclique permet de définir sa valeur en fonction de la position à atteindre : plus la position à atteindre est éloignée, plus le rapport cyclique sera élevé. Pour des raisons techniques, nous avons limité ce rapport à 0,4.

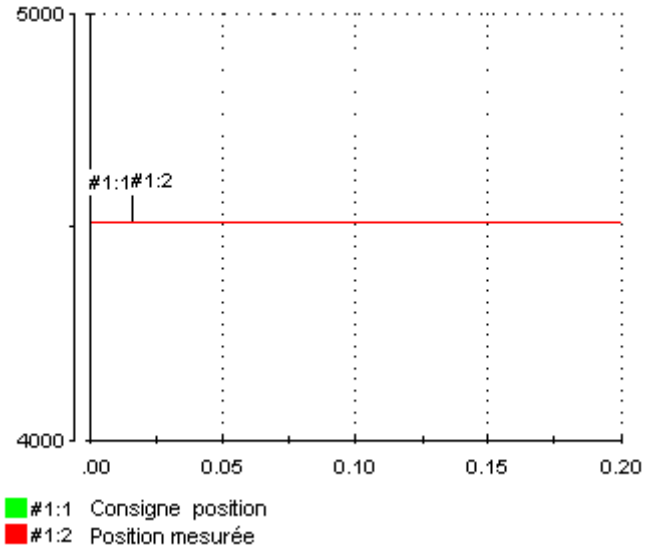


Figure 10 : Réponse en position

Expérimentalement, nous constatons sur la figure 10 que la position finale est bien atteinte (la consigne et la position mesurée sont confondus).

2 .4 Stratégie de commande

La direction a nécessité de la réflexion car nous avons eu quelques soucis. En effet, lors des premiers tests nous avons remarqué qu'à l'initialisation (allumage du véhicule) la régulation de direction plantait. Cela était du au fait que les codeurs retournait des valeurs fausses à l'allumage. Donc nous avons réalisé un compteur pour retarder l'envoi des valeurs vers la dSPACE.

Quand le vérin atteint la butée (non physique), le rapport cyclique envoyé au vérin ne cessait d'augmenter. Nous avons donc imposé des limites virtuelles, qui bloquent le rapport cyclique. Dans la même optique, le rapport cyclique ne s'arrêtait que lorsque la consigne était entièrement atteinte. Cela produisait des comportements indésirables, la direction n'était jamais entièrement fixe. Nous avons donc instauré une fourchette dans laquelle nous considérons que la consigne est atteinte.

Pour ce qui est de la vitesse du vérin, nous avons choisi cette stratégie : plus nous sommes loin de la valeur à atteindre, plus le rapport cyclique est grand. Plus nous nous rapprochons de la consigne, plus le rapport cyclique diminue. Pour des raisons de sécurité, nous avons limité le rapport cyclique à 0,4. Ainsi lors de la conduite du véhicule, les virages ne seront pas trop brusques.

3. Validation des deux trains

Une fois la commande finie, nous l'avons validée sur un boîtier de puissance issu d'un autre RobuCar. Nous avons vérifié que les roues et la direction répondaient correctement aux consignes données. Lorsque les boîtiers de puissance finaux de notre RobuCar ont été terminés, nous les avons montés sur le véhicule. Mais lors des premiers tests nous avons eu des problèmes : le véhicule ne répondait pas aux commandes envoyées. Notre commande ayant été validée sur un bloc opérationnel, le problème ne pouvait être que hardware.

Traction : Nous avons tout de suite cherché du côté du câblage. Nous avons vérifié que les câbles correspondaient aux niveaux des nombreuses connectiques, ce n'était pas toujours le cas. Après réparation (soudures et protection), la commande était de nouveau opérationnelle.

Direction : Nous avons remarqué que la direction se comportait bizarrement. En effet, on ne pouvait tourner que dans une seule direction et la vitesse ne variait pas (toujours très rapide). Nous avons donc regardé les câblages au niveau des connectiques (donc nous plonger dans les documentations). Nous avons trouvé l'erreur les câbles étaient décalés d'une pin... D'où le comportement étrange. Après réparation, tout marchait à nouveau normalement.

4. Acquisition et sauvegarde du courant, tension et vitesse

Les trois mesures nécessaires pour la supervision d'un moteur électrique sont le courant, la tension et la vitesse. En effet ces trois grandeurs représentent les inconnues des équations des résidus d'une MCC. Les autres grandeurs étant des paramètres propres aux moteurs.

Pour cela, on a rajouté des outils de mesures à savoir des capteurs de courant à effet hall et des sondes de tension classique pour chacune des roues. Les codeurs incrémentaux déjà présents nous permettent de mesurer la vitesse.

Afin de pouvoir analyser le comportement du véhicule et diagnostiquer de futurs problèmes, la sauvegarde des grandeurs mesurées s'avère nécessaire.
Le logiciel ControlDesk offre la possibilité de sauvegarder en temps réel uniquement la globalité des variables utilisées sous une structure de donnée complexe (extension .mat). Nous ne pouvions donc pas sélectionner les variables à sauvegarder. De plus l'exploitation de ce fichier sous *Matlab* s'avère très compliqué.

Pour résoudre cette problématique on a utilisé un Parser spécifique en python (TRCParser) qu'on a modifié afin d'exploiter le fichier .mat et d'en extraire les adresses des variables (courant, tension et vitesse) qu'on souhaite sauvegarder. Pour communiquer directement avec la Dspace sans passer par l'environnement de supervision ControlDesk, il existe une librairie de communication en C appelée « Clib ». En effet on a réalisé un programme en C qui va directement chercher les valeurs des variables désirées – dont l'adresse a été extraite en utilisant le parser - à travers la libraire Clib. Ces données sont stockées en temps réel dans un fichier .txt .

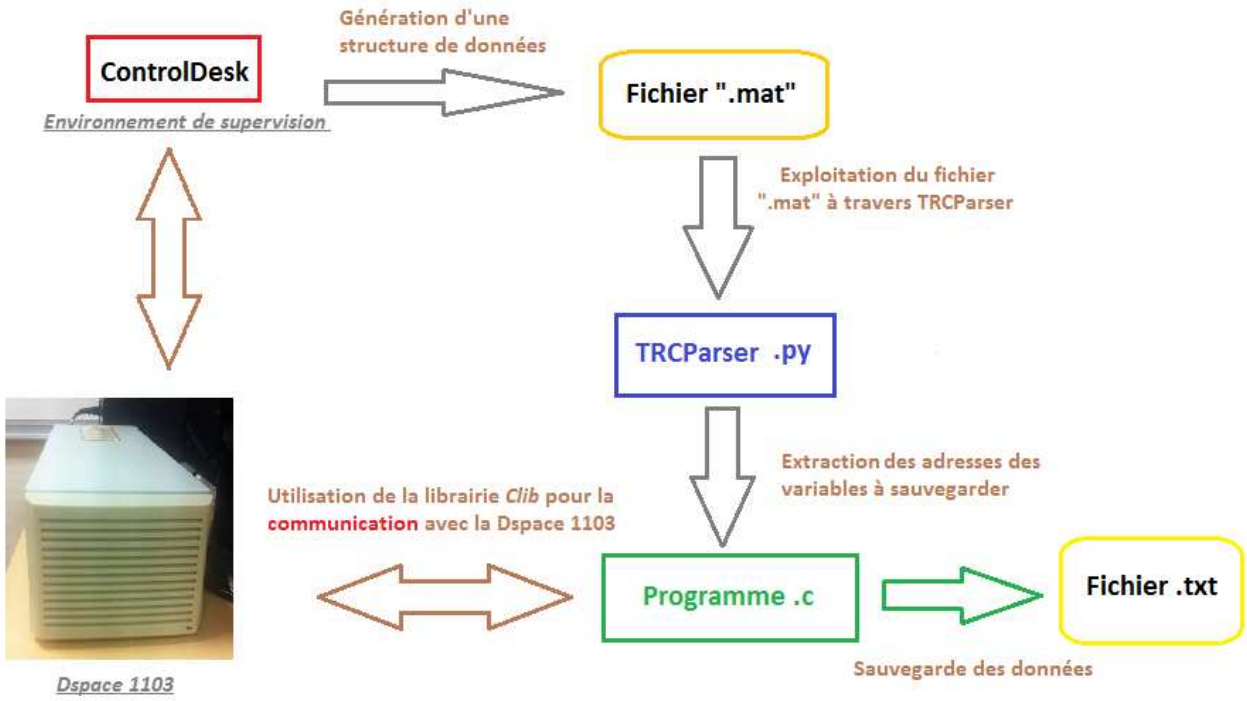


Figure 11 : Architecture de sauvergarde

5. Centrale inertielle

La mesure de l'orientation du véhicule, des accélérations longitudinale et latérale nécessite l'utilisation d'une centrale inertielle. En effet ces différentes mesures seront utilisées dans le future pour le diagnostique, la supervision et la réalisation d'une commande plus précise du véhicule (exemple : prise en compte des glissements pour la traction).

On a utilisé la centrale inertielle Mti de Xsens qui est composée de 9 capteurs :

- 3 accéléromètres
- 3 gyromètres pour mesurer les vitesses angulaires
- 3 magnétomètres pour obtenir la direction du champ magnétique terrestre.

Figure 12 : Centrale inertielle Xsens MTi

Cette centrale intègre son propre processeur ainsi qu'une mémoire permettant de stocker les informations relatives à son étalonnage et les options d'enregistrement qu'on aura choisi lors de la configuration. La liaison est assurée par un câble USB série. En effet la carte d'acquisition de la Dspace ne 'est pas équipée d'un port USB, on a donc connecté la centrale directement au PC.

Xsens, le fournisseur de la centrale inertielle met à disposition des utilisateurs un ensemble de librairies en C++ pour la configuration et l'acquisition des mesures.

Le travail qui a été réalisé pour cette partie est un programme en C++ qui utilise ces librairies ainsi que la librairie Clib pour la transmission des données depuis le PC vers la Dspace. L'utilisation de la librairie Clib est ici nécessaire car la centrale est connectée sur le PC et non sur la Dspace.

6. Interface graphique sous ControlDesk

Lorsque notre système a fini par bien fonctionner et que l'on ne devait plus modifier le layout de manière récurrente, nous avons réalisé une interface graphique ergonomique. Durant toute la durée du projet, nous avons travaillé avec un unique layout qui nous servait à tout afficher sur la même page. Nous avons donc par la suite créé plusieurs layouts ayant chacun un thème spécifique pour facilité l'utilisation du RobuCar. Nous avons créé six layouts différents :

6.1 Poste de pilotage

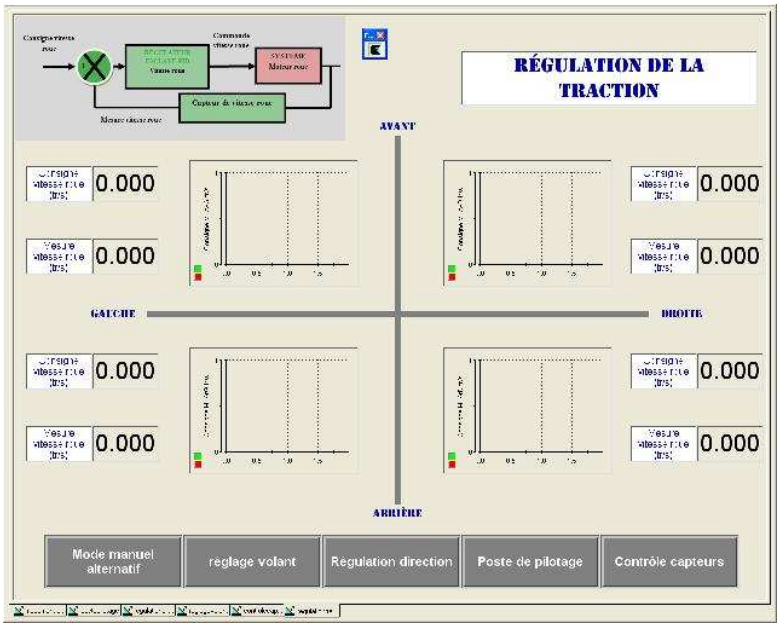
Comme son nom l'indique, c'est la page principale d'utilisation. Il y a toutes les informations utiles pour la conduite du véhicule. Nous avons donc intégré les informations suivantes :

- Vitesse du véhicule en km/h
- Rapport de vitesse sélectionné
- Sens de marche (avant/arrière)
- Mode de direction sélectionné
- Marche/Arrêt (avec LED)
- Boutons menant aux autres layouts

6.2 Réglage volant

Ce layout permet à l'utilisateur de voir les commandes au niveau du volant. Une photo du volant permet de voir clairement à l'écran les attributions des boutons.

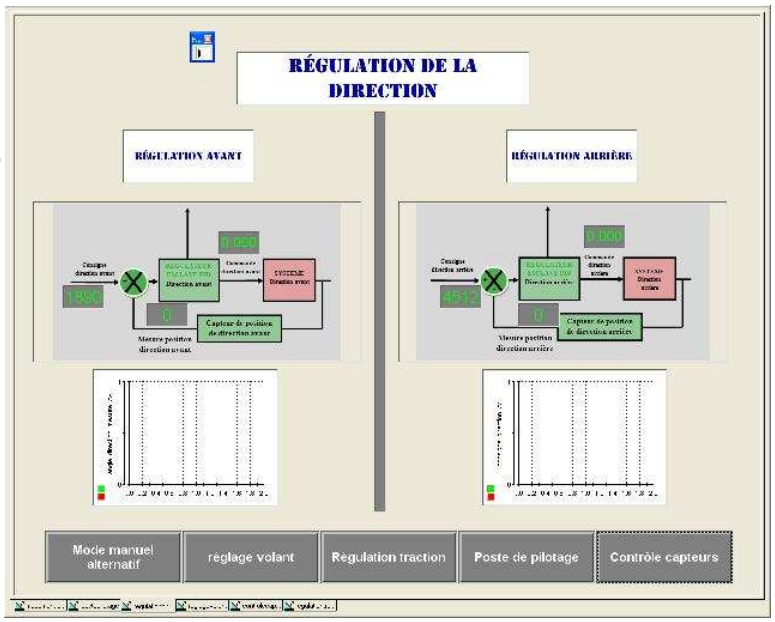
6.3 Régulation traction



Dans ce layout, nous présentons notre régulation de la traction du RobuCar. Les quatre roues sont représentées, la régulation est représentée par un schéma sur lequel nous avons placé les valeurs suivantes :

- Consigne de vitesse de la roue
- Capteur de vitesse de la roue
- Graphique permettant de voir l'évolution de ces deux valeurs.
- Boutons menant aux autres layouts

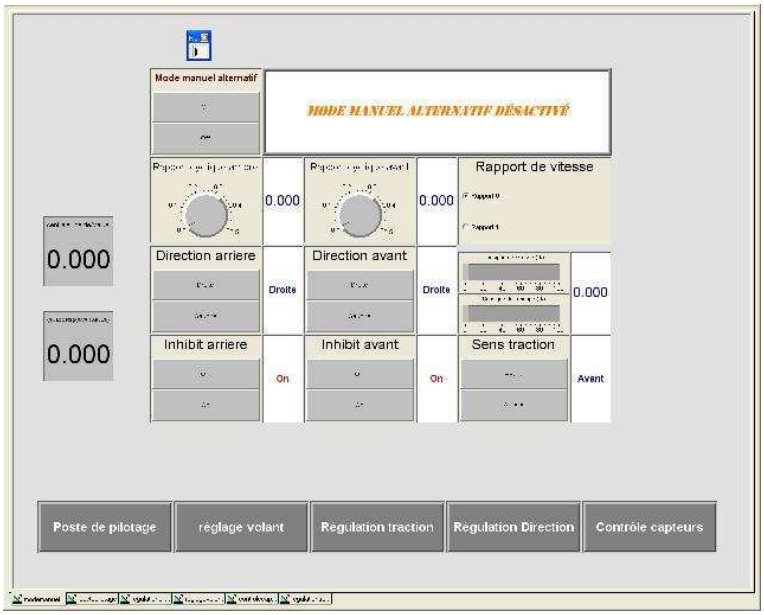
6.4 Régulation direction



Dans ce layout, nous présentons la régulation de la direction du RobuCar. Les deux vérins de direction sont représentés. Pour chaque vérin nous avons représenté le schéma de régulation sur lequel nous avons mis les valeurs de :

- Consigne direction
- Capteur direction
- Commande appliquée (rapport cyclique)
- Graphiques représentant la consigne et le capteur
- Boutons menant aux autres layouts

6.5 Mode manuel alternatif

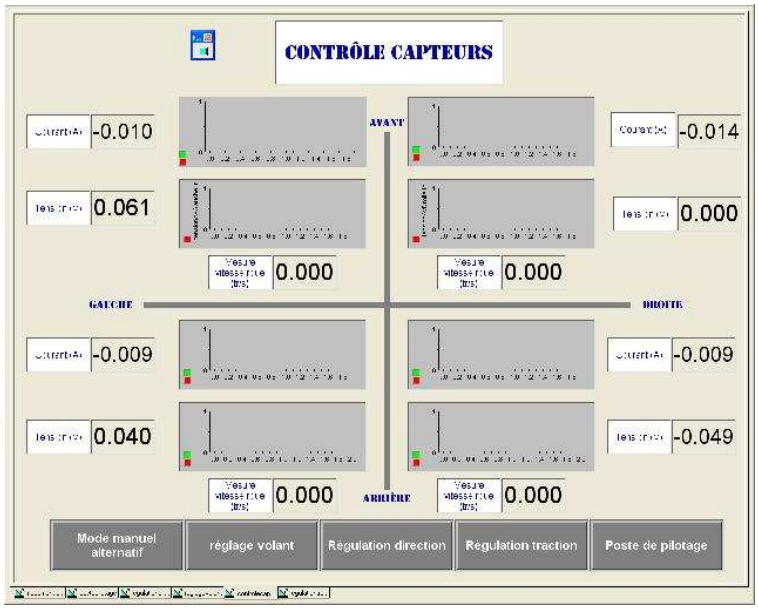


Ce layout permet d'activer la commande manuelle alternative. Cette commande remplace la commande par le volant, elle supprime la régulation de direction, mais garde la régulation de traction. Ce mode peut permettre à l'utilisateur de commander le RobuCar en cas de bugs éventuels comme par exemple le blocage de la direction. L'utilisateur peut débloquer manuellement la direction à partir de cette interface. Voici les différentes variables que nous pouvons commander :

- Inhibit (vérins droit et gauche)
- Sens de la direction
- Rapport cyclique appliqué

- Valeur de la vitesse (en pourcentage)
- Valeur du freinage (en pourcentage)
- Sens de marche (avant/arrière)
- Bouton d'activation du mode
- Boutons menant aux autres layouts

6.6 Contrôle capteurs



Ce layout répertorie les valeurs des capteurs de courant et de tension ainsi que celles de la vitesse de chaque roue. Deux graphiques représentent respectivement la tension en le courant pour chaque roue. A terme, ils serviront pour la supervision du véhicule.

6.7 La gestion marche / arrêt

Lors de la réalisation de l'interface graphique sous ControlDesk, nous nous sommes rendu compte que nous n'avions pas de bouton de marche/arrêt. L'utilisateur pouvait déplacer le véhicule directement, sans avoir à le « démarrer ». Nous avons donc généré un bouton ayant ce genre de fonction.

Il s'agit ici d'immobiliser le véhicule, ainsi dans le bloc Simulink nous n'agissons que sur les variables suivantes : la direction, l'accélération, le freinage et le rapport de vitesse. L'utilisateur peut donc quand même se déplacer dans les différents layouts. Mais le véhicule ne réagira pas. La solution choisie est l'utilisation de switchs qui selon la condition (bouton marche/arrêt) envoient une valeur initiale constante ou les coordonnées du volant.

7. Remplacement d'une Dspace 1103 par une Dspace MicroAutobox et modification des programmes

Lors de notre PFE nous avons été amenés à adapter les programmes d'une Dspace 1103 qui équipait un autre Robucar à une Dspace MicroAutobox.

La Dspace 1103 est utilisé pour des applications de recherche en laboratoire. En effet sa taille assez encombrante (nombre d'entrées / sorties important) n'en fait pas un excellent outil pour des applications embarquées telle que les véhicules autonomes. Il existe une autre catégorie de Dspace à savoir la MicroAutobox, très compacte et qui offre des performances assez proches de celle de la 1103.

Figure 13 : Dspace MicroAutobox

Le travail réalisé consiste en la modification des programmes Matlab Simulink et ControlDesk puis leur validation.

La programmation de la Dspace 1103 se fait via Matlab 2006, la Microautobox quant à elle nécessite Matlab 2010. Ce changement de version et de Dspace a nécessité d'effectuer plusieurs modification. En effet les blocs de communications (CAN, RS232) ont été changés et reconfigurés et les différents programmes de suivi et d'acquisition en C recompilés.
Au niveau du programme ControlDesk il a juste fallu changer les chemins d'accès.

Une fois les modifications terminées, on s'est rendu compte que le programme de suivi automatique d'un véhicule ne fonctionnait pas. Après analyse du problème on a conclue que le programme nécessitait une puissance de calcul supérieur à celle fournie par la MicoAutobox.

On a donc modifié le pas de calcul ce qui nous a permis de résoudre le problème. Le véhicule a été ensuite testé et validé sur la piste d'essai.

8. Optimisation de l'espace pour la carrosserie

Une fois la commande fiable, et l'interface graphique finie, nous avons entrepris de remonter proprement le RobuCar. La principale problématique réside dans le manque de place. Nous avons donc optimisé les longueurs de câbles, cherché le meilleur emplacement pour la dSPACE 1103 et sa carte d'E/S.

L'installation d'un adaptateur de tension a été indispensable. En effet, l'ordinateur doit être à bord du véhicule et fonctionne avec du 220V, tout comme la dSPACE. Les boîtiers de puissance étaient trop grands et touchaient la carrosserie, nous avons donc été contraint de les raccourcir. Pour fiabiliser le système, nous avons vérifié les soudures au niveau des connectiques extérieures et avons placé des protections en plastique sur celles ci. Nous avons fait des chemins de câbles en utilisant des colliers *colson*, pour rendre le travail plus propre et que les fils n'aillent pas n'importe où.

Après cela, nous avons testé le code à nouveau, le système ne réagissait plus comme avant. Nous avons encore dû chercher les erreurs de câblage. Finalement nous avons trouvé que deux fils s'était dessoudés. Après réparation, le système était à nouveau entièrement opérationnel et nous avons pu installer la carrosserie.

Tests effectués sur piste d'essai

Deux semaines avant la fin du projet, nous avons fini de remonter le véhicule avec sa carrosserie. Nous avons vérifié sur cales que la carrosserie ne gênait en rien la direction, par exemple lorsque l'on braque à fond, les roues ne doivent pas toucher la carrosserie. Nous savions que la voiture avait un problème récurrent aléatoire sur le train avant pour la traction en régulation. Ainsi pour déplacer le véhicule jusqu'à la piste d'essai du laboratoire Lagis (à 50m de polytech'Lille), nous avons roulé en boucle ouverte à faible allure. Cette piste nous a permis de vérifier la fiabilité et la sécurité de notre véhicule. Les tests réels mettent en évidence des points à améliorer ou corriger :

- Le problème de régulation du train avant.(non réglé par manque de temps)
- L'accélération trop brutale. (réglé)
- Les « rapports de boîte de vitesse ». (réglé)

- Le blocage de la direction avant en buté. (réglé)
- La régulation de la traction. (non réglé par manque de temps)

1. Régulation de la traction :

Sur la piste d'essai nous avons pu tester notre régulation des roues. Cette régulation est fonctionnelle en ligne droite (quand le train avant ne plante pas). Le problème se pose en virage : avec l'inertie du véhicule la roue intérieure tourne moins vite que la roue extérieure. Sur les voitures, il y a un différentiel qui prend en compte ce phénomène. Actuellement sur la voiture, il n'y a rien qui en tient compte.

Ainsi la consigne sera de 1 pour la roue intérieure et extérieure. En théorie, le véhicule déraperait si c'était le cas, or la puissance des moteurs ne le permet pas. En pratique, on observe que la roue intérieure est de 0.8 et l'extérieure de 1. La roue intérieure n'atteint pas la consigne, le PI continu d'augmenter la commande et finalement le variateur finit par s'arrêter. Puis les quatre variateurs s'arrêtent immobilisant le véhicule.

La solution est simple, il faut écrire un programme qui en fonction du virage pris envoie des commandes prenant en compte le phénomène évoqué. Nous avions commencé une étude déterminant le centre instantané de rotation (CIR) afin de pouvoir déterminer la vitesse de chaque roue en fonction de l'angle de braquage et d'une vitesse de référence. Nous n'avons malheureusement pas eu le temps de l'implémenter. Cette étude s'appuyait entre autre sur les valeurs que retourne la centrale inertielle. (cf ANNEXE).

2. Blocage de la direction avant en buté :

Nous avions fait des essais sur cales pour vérifier que la direction n'aillait pas en buté. La buté en question, c'est la biellette de direction... Donc pour la conserver, nous avons limité la commande. Si le capteur renvoie une valeur supérieure à la limite fixée, alors on arrête la régulation de la direction (on entre en boucle ouverte). Pendant les premiers roulages, nous nous sommes retrouvés avec la direction bloquée en buté. Nous avons tout d'abord cru à un problème mécanique de part la forme de la pièce. Mais les réparations que nous avons tentées n'ont pas fonctionné. Nous avons donc regardé l'écran de supervision de la régulation de direction et avons constaté que nous n'envoyions pas de rapport cyclique pour faire bouger le vérin.

Avec l'inertie du véhicule, en virage il est possible de dépasser les seuils de sécurité que nous avions fixés. Ces seuils de sécurité se transforment dans cette situation en pur danger. Nous les avons donc revus pour éviter ces blocages dangereux. Actuellement la voiture est plus sûre, et la direction ne se bloque plus.

3. Bug du train avant :

Au cours du développement de notre projet, nous avons remarqué que le train avant avait un problème récurent au niveau de la régulation de la traction. Sur cales, le problème était assez rare, mais dès que nous avons posé la voiture, le problème s'est révélé être plus important que nous le pensions. Nous avons cherché l'origine de cette erreur. Tout d'abord dans la direction, car c'était lorsque l'on tournait le volant que les roues s'emballaient. Nous n'avons rien trouvé dans cette voie là. Nous avons ensuite pensé à un problème de capteur, car la voiture ne plante pas en boucle ouverte. Nous pensons que cela pourrait être dû à la manière dont les capteurs sont connectés à la dSPACE : ils ne sont pas branchés directement sur la dSPACE comme ceux du train arrière. En effet, la régulation du train arrière fonctionne sans soucis. Les capteurs du train avant sont ramenés à l'intérieur du bloc de puissance avant sur une connectique type d-sub, puis des câbles les relient à la dSPACE. Nous pensons qu'il peut y avoir une perte de qualité du signal dû à cette connectique.
Nous n'avons pas eu le temps de corriger ce problème étant donné que la voiture à été remonté tardivement.

4. Accélération trop brutale :

Nous avons fait un choix de commande basé sur un véhicule réel. C'est à dire que nous avons voulu reproduire la même interface de conduite que dans une voiture normale. Nous avons donc opté pour 5 rapports de vitesse. Cependant, à cause des caractéristiques des moteurs électriques, il faut lisser l'accélération. En effet, un moteur électrique fournit le couple instantanément. Pour améliorer le confort du conducteur, il faut créer une rampe qui permet d'atteindre ce couple de manière progressive. Nous avons effectué cette rampe pour une meilleure qualité de conduite quand le véhicule était sur cales. Lorsque nous avons posé le véhicule et fait les premiers tests sur la piste d'essai, nous nous sommes rendu compte que les passages de vitesse étaient toujours trop violents : l'accélération du véhicule était bien trop importante sauf pour la première vitesse. Nous avons corrigé ce problème, en configurant les variateurs des moteurs. Nous avons augmenté le temps de réponse, ce dernier était vraiment trop court. Après cette modification, les accélérations étaient moins brusques.

5. Rapports de boîte :

Après avoir fait tester la voiture à plusieurs personnes, il en est ressorti que les rapports de vitesse étaient mal équilibrés. Nous avions fait le choix d'une première vitesse très faible pour les manœuvres délicates (typiquement faire rentrer le véhicule dans la salle de projet). Ainsi, même pied au planché, la voiture avançait à très faible allure. Les retours nous ont indiqué que cette vitesse était trop faible et qu'il fallait l'augmenter. Ainsi, nous avons modifié notre stratégie de commande pour les rapports de vitesse. Nous avons redéfini ces rapports de la sorte :

Rapports	Avant	Après
Première	15,00%	20,00%
Deuxième	30,00%	44,00%
Troisième	50,00%	63,00%
Quatrième	75,00%	82,00%
Cinquième	100,00%	100,00%

Figure 14 : Tableaux des rapports de vitesse

Ce tableau représente les pourcentages de la vitesse maximale que l'on peut atteindre en fonction du rapport sélectionné. Par exemple, en troisième la vitesse maximale sera de 63% de la vitesse maximale que le moteur peut atteindre. Nous avons ainsi corrigé la trop faible vitesse en première.

6. Fiabilité du câblage :

Lors du premier essai, nous nous sommes rendu compte que les batteries n'étaient pas assez stables dans le RobuCar. En effet, elles se sont déplacées lors de virages/freinages trop violents. Les connectiques se sont abîmées suite à cette inertie, entraînant la perte des valeurs des codeurs de la direction. Ainsi, la régulation de la direction était HS. Grâce à notre « mode manuel alternatif », nous avons pu rentrer la voiture au garage en contrôlant les vérins de direction en boucle ouverte. Nous avons rapidement trouvé le problème le jour suivant en démontant la carrosserie : la connectique était débranchée. Nous l'avons donc fixé au châssis avec un système de vis-écrou. Pour

le problème des batteries, nous les avons calées à l'aide de l'armature en acier fourni avec le châssis et quelques coups de perceuses. Après remontage et tests sur la piste, il n'y avait plus de problèmes de connectiques.

Analyse des résultats

Connectique, Câblage :

Une des principales difficultés que nous avons rencontré, c'est le hardware. En effet, le RobuCar que nous avons récupéré était complètement nu : il n'y avait sur le châssis que les roues avec leur moteur, et les deux vérins de direction. Les boîtiers de puissance étant absents, le technicien de Lagis (Mr Michel POLLART) a fabriqué ces boîtiers en clonant ceux des RobuCars existant. La difficulté est évidente, il y a énormément de fils dans ce boîtier et donc le câblage n'est pas forcément fiable (l'erreur est humaine).

Temps réel :

Le fait que notre système soit embarqué et en temps réel nous a posé quelques soucis. En effet, les fonctions embarquées « Matlab embedded function » ne supportent pas toute les fonctions Matlab. Ainsi, nos codes n'étaient pas faciles à mettre en place. Plusieurs fois, nous avons du revoir la manière de faire pour réaliser l'objectif qu'on s'était fixé. Par exemple lors de la récupération des données de l'Arduino, nous avons voulu utiliser une embedded function pour reconstituer les valeurs des codeurs mais les fonctions utilisées n'étaient pas supportées. Ainsi nous avons décidé de le faire autrement en utilisant la logique et le binaire. La récupération des données du RS232 fonctionne parfaitement.

Lors de la stratégie de commande concernant le freinage et l'accélération, nous avions mis des variables intermédiaires qui sauvegardaient l'ancienne valeur voulu pour la comparée à la nouvelle. Cependant, le compilateur nous a adressé une erreur. Nous en avons déduit qu'il ne voulait pas faire de sauvegarde en temps réel. Nous avons donc cherché une autre solution : des blocs de sauvegarde existent pour la dSPACE sous Simulink. Après configuration des registres, nous avons été en mesure de sauvegarder en temps réel et ainsi pu réaliser proprement notre stratégie.

Procédure d'allumage :

Nous avons constaté que nous ne pouvions pas charger le programme dans la dSPACE n'importe comment. Il faut respecter une procédure pour ne pas faire planter le système. Après plusieurs tests, nous nous sommes rendus compte que lorsque ControlDesk était en mode animation et que la voiture était alimentée, le changement du .sdf (après construction sous matlab) provoquait un problème : la direction s'emballe et vient taper en buté. Ainsi nous avons mis en place une procédure à suivre pour initialiser correctement le RobuCar et pouvoir le conduire sans soucis.
Voir la procédure en **ANNEXE**.

Manuel d'utilisation :

La stratégie de commande est différente des autres RobuCars. Ainsi, pour les futurs utilisateurs, nous avons créé un manuel d'utilisation. Nous informons ainsi l'utilisateur des particularités de notre véhicule. Il ne tient que sur une page car la plupart des utilisateurs ne lisent pas les manuels d'utilisation... (cf **ANNEXE**).

Réflexions :

Au début de notre projet, nous sommes partis de rien. Ainsi, il était dur de se projeter dans l'avenir pour imaginer le RobuCar rouler. Cependant, nous avons tout fait pour réaliser notre objectif. Nous avons sollicité l'aide de Michel POLLART pour la remise en forme du véhicule, grâce à lui et à son travail nous avons pu voir notre projet rouler.

Nous nous étions fixés un planning à respecter pour pouvoir finir le projet dans les temps. Nous nous sommes aperçu qu'il était délicat d'estimer le temps que les tâches nous prendraient. L'objectif était de faire rouler la voiture, nous n'avons cependant pas pris en compte le temps que nous prendraient les réglages. En effet, sur cales la commande était correcte mais sur la piste d'essai, nous avons bien vu qu'il y avait des failles à corriger. Le manque de temps nous a permis de n'en corriger qu'une partie seulement. Il aurait été intéressant d'améliorer la régulation de la traction, mais le temps restant ne le permettait pas, bien que nous avions une solution.

Développement du projet :

A la fin de notre projet, nous nous sommes interrogés sur les futures améliorations possibles sur le véhicule, les problèmes à résoudre avant de pouvoir continuer le développement etc. Nous proposons donc les idées ci dessous :

- Régler le problème de la régulation de la traction avant : Pour faire avancer le projet, il faudra résoudre ce souci que nous avons remarqué. Une fois ce souci réglé, la régulation du véhicule entier pourra être développée.

- Différentiel : pour faire une bonne régulation de traction, il faudra simuler l'effet d'un différentiel dans la commande. Ainsi, le véhicule pourra prendre des virages tout en étant régulé.

- Ordinateur portable : L'espace dans le RobuCar est très faible. Le changement de la tour et de l'écran 17 pouces par un ordinateur portable serait parfaitement adapté dans le cas d'un véhicule autonome.

- Ecran tactile : Dans le cas où l'ordinateur portable ne serait pas possible, nous pourrions instaurer un écran tactile pour facilité la commande (la souris n'est pas adaptée à un système embarquée).

- Utilisation de l'Arduino avec le CAN (=> implémentation du télémètre laser sur le RS232) : Actuellement l'Arduino communique en RS232 avec la dSPACE. Comme c'est l'unique périphérique, cela ne pose pas de soucis. Mais on ne peut pas implémenter le télémètre laser (en RS232 également). Il faudrait donc faire communiquer l'Arduino en CAN pour libérer le RS232 pour le télémètre laser.

- Branchement des sondes de tension : Pour améliorer la supervision.

- Installation des capteurs de vitesse extérieurs : Une redondance capteur est utile pour la supervision. Vérifier qu'il n'y a pas de glissement.

- Bouton AU à distance : Il sera indispensable lors des tests de suivi de véhicule (après implémentation du télémètre laser).

- Implémentation centrale inertielle : elle pourra fournir de nombreuses informations comme la vitesse réelle du véhicule.

CONCLUSION

Ce projet a été très enrichissant car il nous a permis de voir plein de domaines différents. En effet, la réalisation de la commande nous a permis d'appliqué les cours d'automatique, l'Arduino nous a permis de programmer en C et de revoir la communication série. Le fait que l'on travaille sur un véhicule réel nous a permis de faire de la technique comme souder des câbles, mais aussi de la mécanique lorsque nous avons du optimiser l'espace... De plus ce projet était parfaitement adapté pour la formation que nous avons suivi en IMA section Système Autonome : le RobuCar est un véhicule autonome. Notre mission était de faire revivre l'épave que nous avions récupérée, mais par la suite de nombreux capteurs seront installés et l'on pourra faire en sorte que le RobuCar évolue de manière autonome (suivi de véhicules par exemple) en réalisant une supervision plus poussée.

La démarche à suivre était également très intéressante : nous avons réalisé une commande en étudiant le système que nous avions. Nous avons donc du nous documenter sur le châssis en question et sur la dSPACE. Une fois la commande réalisée, il a fallut la valider d'un point de vue sécurité et fiabilité lors des tests que nous avons réalisés. Nous avons également du valider le hardware avec les boîtiers de puissance faits maison. Toute cette démarche scientifique était vraiment intéressante à suivre.

Cette expérience nous a vraiment enrichies d'un point de vue technique. Nous avons pu découvrir le temps réel et les contraintes que cela impose. Le développement d'un projet de longue durée s'est révélé être différent des projets précédents (plus courts) : les consignes changent au fur et à mesure. Nous avons atteint l'objectif principal qui était de faire rouler le RobuCar à nouveau en réalisant une commande sûre, et fiable.

Annexe

Procédure d'allumage du RobuCar

1°) Vérifier que l'**arrêt d'urgence est enfoncé**.

2°) Mettre le contact (coupe circuit).

3°) Vérifier que l'adaptateur de tension est allumé, sinon l'allumer.

4°) Vérifier que la dSPACE est allumée, sinon l'allumer.

5°) Vérifier que l'ordinateur est allumé, sinon l'allumer.

6°) Sur l'ordinateur, ouvrir le projet « projetrobucar1103.cdx » dans *Bureau/projetrobucar1103* avec *ControlDesk*.

7°) Une fois le poste de pilotage affiché, **enlever le bouton d'arrêt d'urgence.**

8°) Appuyer sur le bouton « marche » de l'interface pour pouvoir conduire le RobuCar.

Manuel d'utilisation

Comment allumer le RobuCar :

Suivre la procédure d'allumage.

Comment démarrer :

Pour pouvoir démarrer la voiture, cliquez sur le bouton marche sur ControlDesk ou appuyez sur le bouton start du volant. **La LED rouge doit passer au vert pour pouvoir conduire le véhicule**.

Pour plus de renseignement sur les commandes au volant, regardez l'interface « **réglage Volant** » dans ControlDesk.

Particularités du véhicule :

A part la première, les autres rapports ont une vitesse minimum en consigne, donc **si vous arrêtez d'accélérer, le véhicule ralentira** pour se caler sur cette vitesse, **mais ne s'arrêtera pas.**

La marche arrière se passe uniquement lorsque le rapport est à 0. Même à vitesse nulle, si vous passez en marche arrière alors que vous êtes en première, vous irez en marche avant.

Les freins sont très puissants, allez y avec prudence. Il est conseillé d'utiliser les rapports pour réduire la vitesse.

Pour passer la vitesse supérieure, il faut atteindre une certaine vitesse. Si vous passez la vitesse mais que vous ne sentez aucune accélération, c'est que la vitesse n'est pas passée. Rétrogradez accélérez un peu plus et passez la vitesse.

En cas de problème sur le véhicule, vous pouvez utiliser le « **mode manuel alternatif** ». Il permet de contrôler le véhicule en boucle ouverte (donc sans volant) uniquement avec la souris.

/!\ Précaution /!\

Lors de l'accélération, **ne pas écraser la pédale à fond ! Allez y progressivement.**

Le tableau des entrés / sorties de la dSPACE 1103 permet de repérer rapidement à quoi correspond chaque câblage. Ces informations sont essentielles pour la compréhension de l'électronique, dans le but d'une amélioration possible.

	Courant	Moteur	Codeurs incrémentaux	Codeurs Absolus	CP31	CP30
Roue ArG	ADC_C17	DAC_C1	INC_2			
Roue ArD	ADC_C18	DAC_C2	INC_1			
Roue AvG	ADC_C19	DAC_C3	INC_6			
Roue AvD	ADC_C20	DAC_C4	INC_5			
Vérin avant				Arduino pin 10 (clock) et 11 (data)	Pin 11 = PWM9	Pin 3 = DIR- Pin 20 = Inhibit
Vérin arrière				Arduino pin 12 (clock) et 13 (data)	Pin 10 = PWM7	Pin 1 = DIR- Pin 18 = Inhibit

Figure 1 : Tableau des E/S

	B-	A2+
Roue ArG	ADC_C4	ADC_C3
Roue ArD	ADC_C2	ADC_C1
Roue AvG	ADC_C8	ADC_C7
Roue AvD	ADC_C6	ADC_C5

Figure 2 : Mnémoniques des entrées utilisées pour les sondes de tension.

```
int binaryArray_av[14];
int binaryArray_ar[14];
int i = 0;
int j = 0;
int pos_av=0;
int pos_ar=0;


void getPosition();             //  Récupération des données, communication SSI avec les capteurs
void transformation();          //  Mise en forme de la donnée, on enléve le bit de start et on passe du
binaire en décimal
void sendPosition();            //  Envoi des données provenant des deux capteurs  en RS232,
                                //  les signaux émis par l'arduino sont en TTL une conversion TTL -> RS232
est nécessaire à l'aide d'un MAX232 par exemple.

void setup()
{

  SetParity('n');               // Initialisation de la liaison RS232
  SetStopBits(1);
  SetWordLength(8);
```

```
  DDRB= B00000000;              //Initialisation du PortB, Utilisation des pins 12 (PB6) et 10 (PB4) en sortie (Clk)
  DDRB = (1<<PB6)|(1<<PB4);     //Pins 11 et 13 (Data)

  delay(500);
  Serial1.begin(9600);
}


void loop()
{
  getPosition();
  transformation();
  sendPosition();
  delay(2);
}


void getPosition()
{
  for(j = 0;j < 14; j++)
  {
    binaryArray_av[j] = 0;
    binaryArray_ar[j] = 0;
  }

  delayMicroseconds(40);

  for(j = 0; j < 14; j++)                  // Les codeurs absolus transmettent des données en 13 bits
  {
   PORTB = (0<<PB4)|(0<<PB6);              // Géneration de l'horloge CLK à 0
   delayMicroseconds(3);                   // Le premier bit reçu correspond à un bit de start qui ne fait pas partie de la donnée

   binaryArray_ar[j] = (PINB & (1<<7));    // A chaque coup d'horloge les capteurs transmettent un bit,
   binaryArray_av[j] = (PINB & (1<<5));    // Lecture les deux pins et on stock la donnée de chaque capteur
                                           // dans son tableau.


   PORTB = (1<<PB4)|(1<<PB6);              // Géneration de l'horloge CLK à 1
   delayMicroseconds(3);
  }

  delayMicroseconds(100);
}
```

```
void transformation()
{
  for (i=0; i < 14; i++)                    // Remise en forme de l'information, pour les capteurs une valeur différente de 0 correspond à 1.
  {
    if (binaryArray_av[i]!=0)
      binaryArray_av[i]=1;

    if (binaryArray_ar[i]!=0)
       binaryArray_ar[i]=1;
  }

  pos_av = 0;
  pos_ar = 0;

 for (i = 1; i < 14; i ++)                  // Suppression du bit de start
 {
   pos_av = (pos_av<<1) | (binaryArray_av[i]);
   pos_ar = (pos_ar<<1) | (binaryArray_ar[i]);
 }

}


void sendPosition()
{


  /*Les données du capteurs   */
  int byte1 = pos_av/256;                   // La liaison RS232 nécessite des trames de 8 bits
  int byte2 = pos_av - byte1*256;           // Découpage de la donnée en deux octets

  int byte3 = pos_ar/256;
  int byte4 = pos_ar - byte3*256;


  Serial1.write(byte2);
  delay(1);
  Serial1.write(byte1);

  delay (20);

  Serial1.write(byte4);
  delay(1);
  Serial1.write(byte3+32);                  // Signature de l'octet de poid fort afin d'identifier les données provenant du capteur arrière

}


void SetStopBits(int nbStopBits)
{
```

```
  if (nbStopBits == 1)
    UCSR0C = UCSR0C | B00000100;
  else
    UCSR0C = UCSR0C | B00000000;
}


void SetParity(int parity)
{
  if ((parity == 'O')|(parity == 'o'))
    UCSR0C = UCSR0C | B00110000;

  else if ((parity == 'E')|(parity == 'e'))
    UCSR0C = UCSR0C | B00100000;

  else // ((parity == 'N')|(parity == 'n')))
    UCSR0C = UCSR0C | B00000000;
}


void SetWordLength(int wordlength)
{

  if (wordlength == 5)
    UCSR0C = UCSR0C | B00000000;

  else if (wordlength == 6)
    UCSR0C = UCSR0C | B00000010;

  else if (wordlength == 7)
    UCSR0C = UCSR0C | B00000100;

  else if (wordlength == 9){
    UCSR0C = UCSR0C | B00000110;
    UCSR0B = UCSR0B | B00000100;
  }
  else
    UCSR0C = UCSR0C | B00000110;

}
```

Figure 3 : Programme Arduino pour l'acquisition et la transmission en RS232 des données de deux codeurs absolus communiquant en SSI

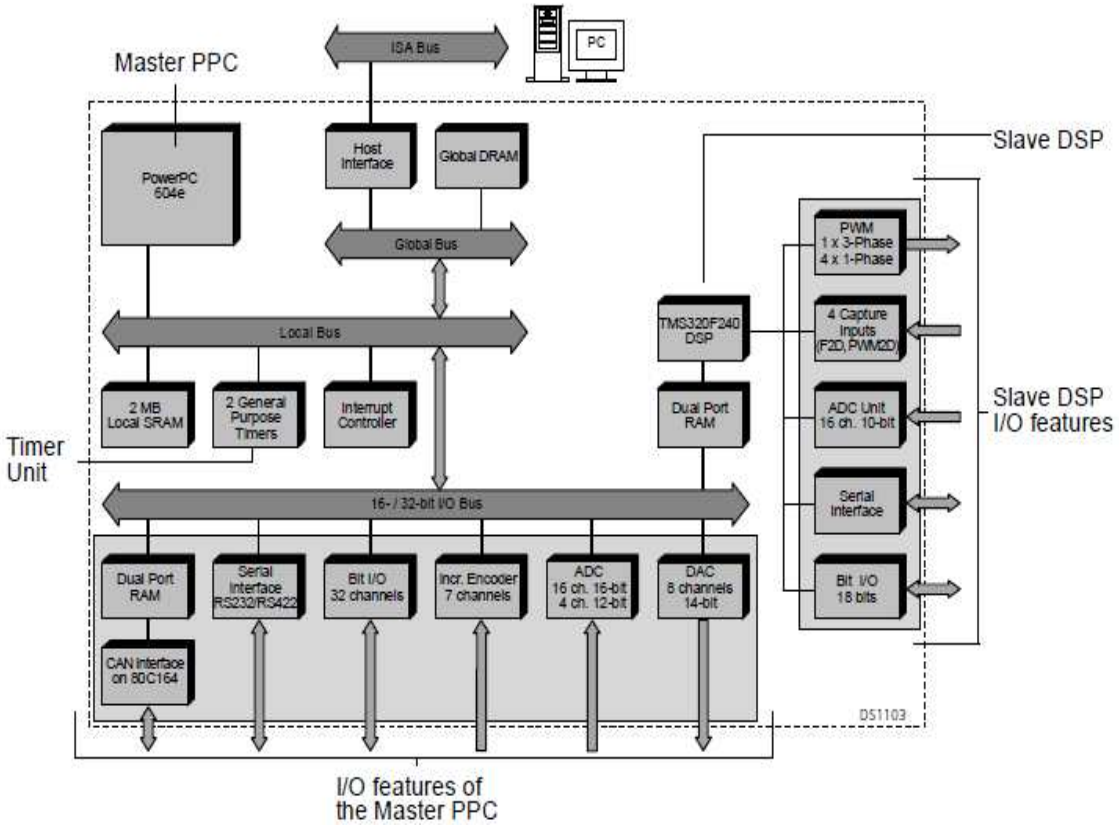


Figure 4 : Architecture et fonctionnalités d'une dSPACE 1103

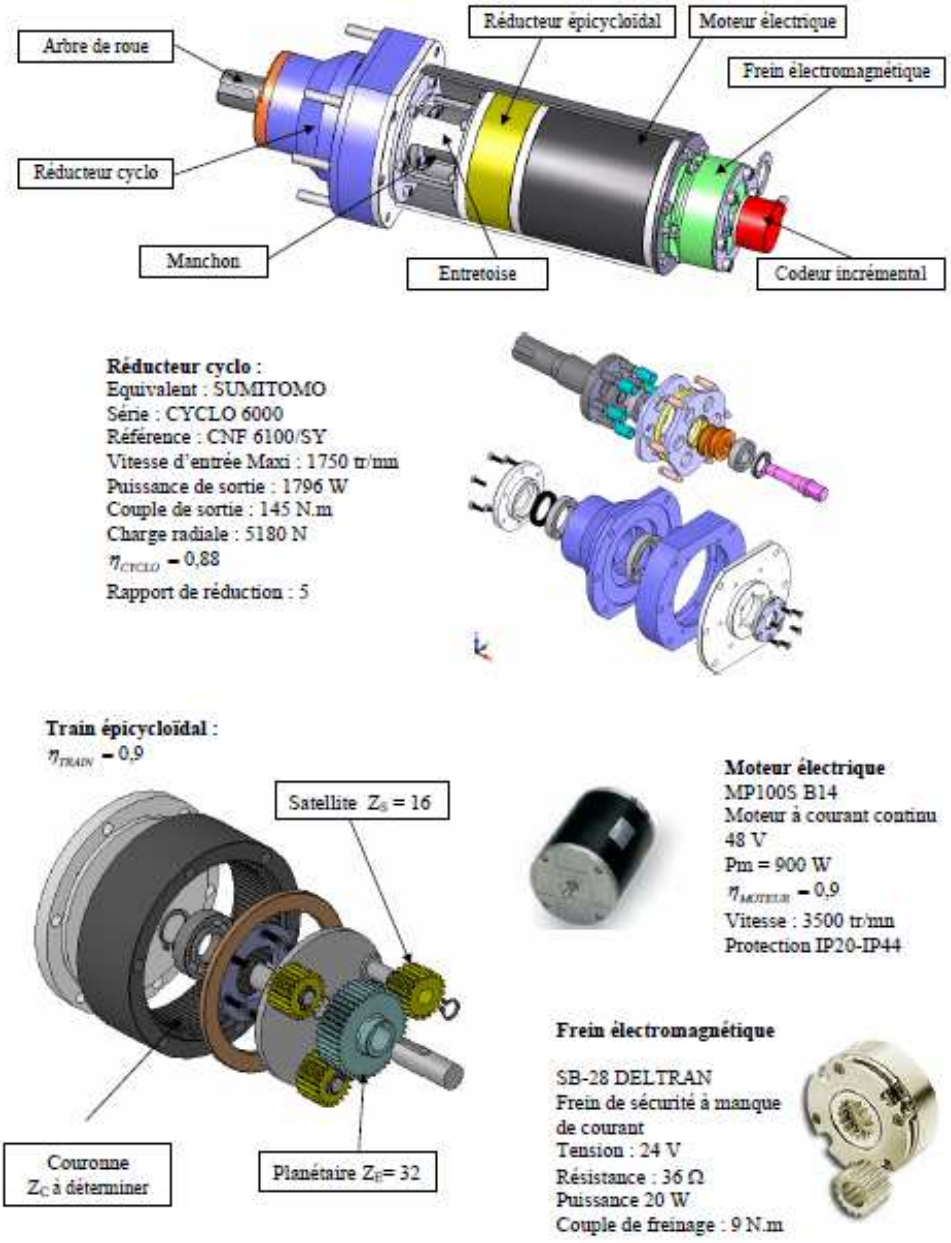


Figure 5 : Description du système de traction et de freinage

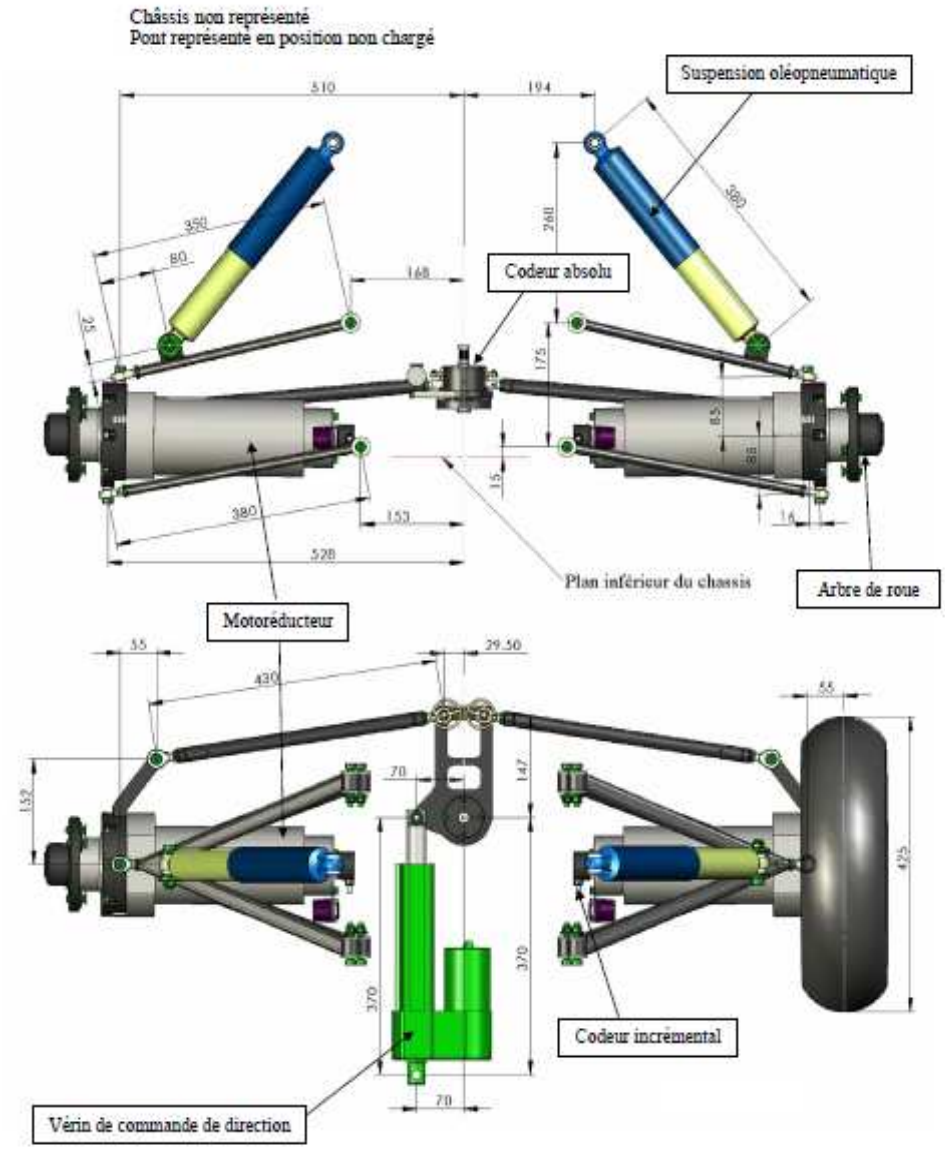
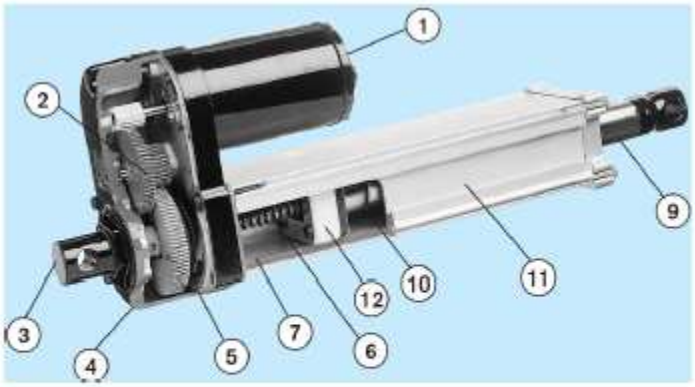


Figure 6 : Description de la direction et de la suspension

1 - Moteur CC + protection thermique
2 - Train d'engrenages
3 - Chape à incréments de 30°
4 - Boîtier de réduction étanche
5 - Embrayage de surcharge
6 - Vis ACME ou vis à billes
7 - Ecrou à billes avec frein de maintien
9 - Tube d'extension acier inox
10 - Ecrou
11 - Tube de protection profilé alu
12 - Dispositif anti-rotation

Sens de déplacement de la tige
Sortie : tension positive
Rentrée : tension négative



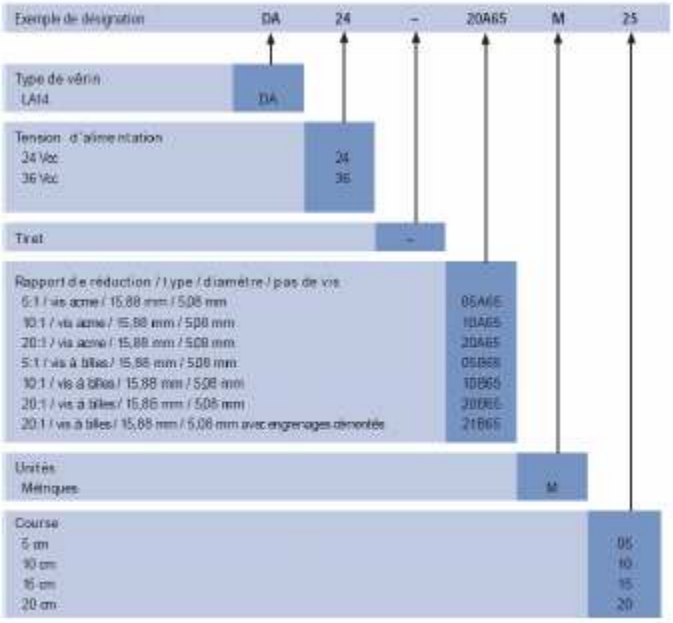


Tableau des valeurs limites

Modèle	Charge dynamique max. [N]	Vitesse à charge min. [mm/s]	Vitesse à charge max. [mm/s]
DA24-05A65	1100	54	32
DA36-05A65	1100	54	32
DA24-05B65	2250	61	37
DA36-05B65	2250	61	37
DA24-10A65	2250	30	18
DA36-10A65	2250	30	18
DA24-10B65	4500	30	19
DA36-10B65	4500	30	19
DA24-20A65	2250	15	12
DA36-20A65	2250	15	12
DA24-20B65	4500	15	12
DA36-20B65	4500	15	12
DA24-21B65	6800	15	11
DA36-21B65	6800	15	11

Figure 7 : Description du vérin électrique

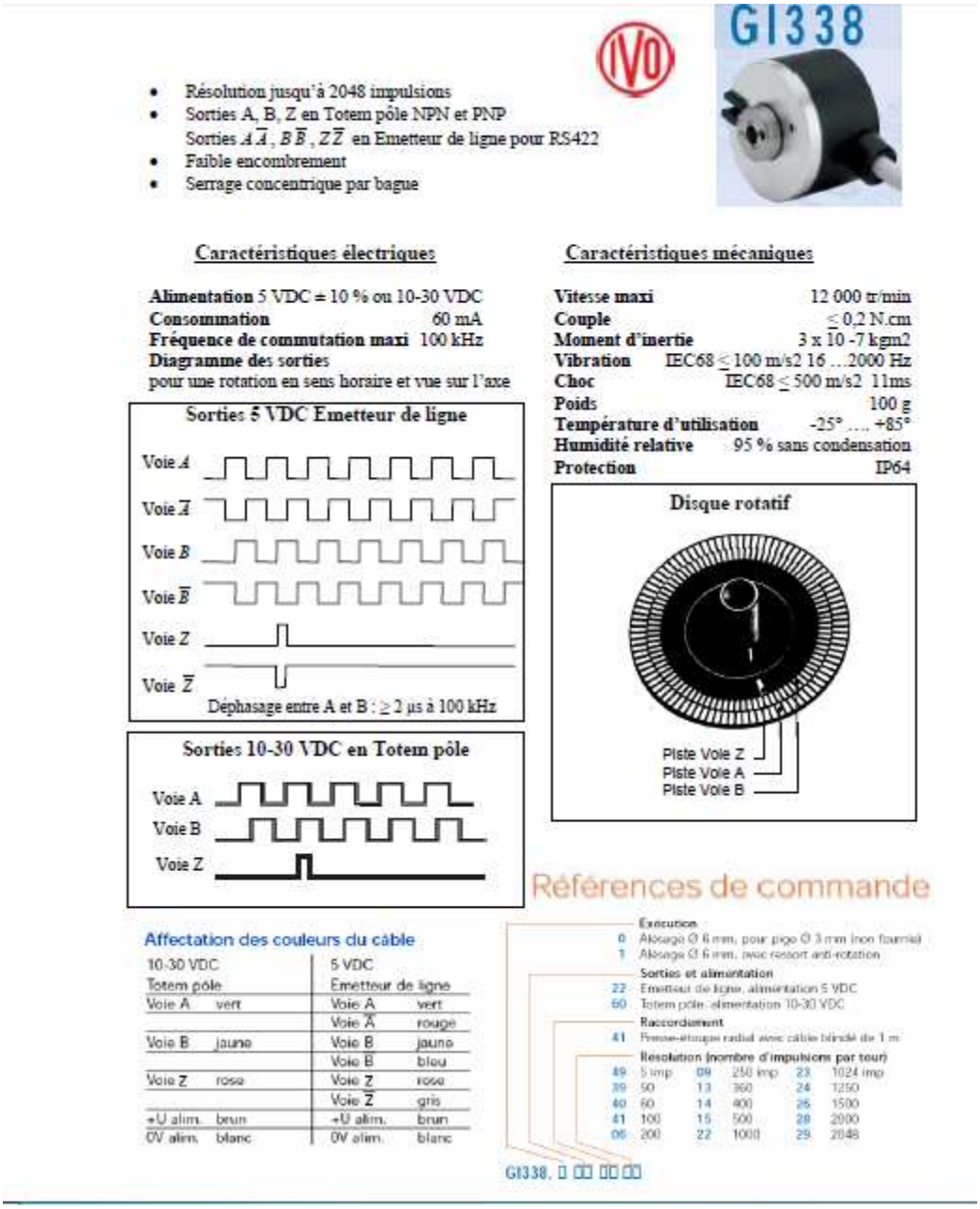


Figure 8 : Description du codeur incrémental

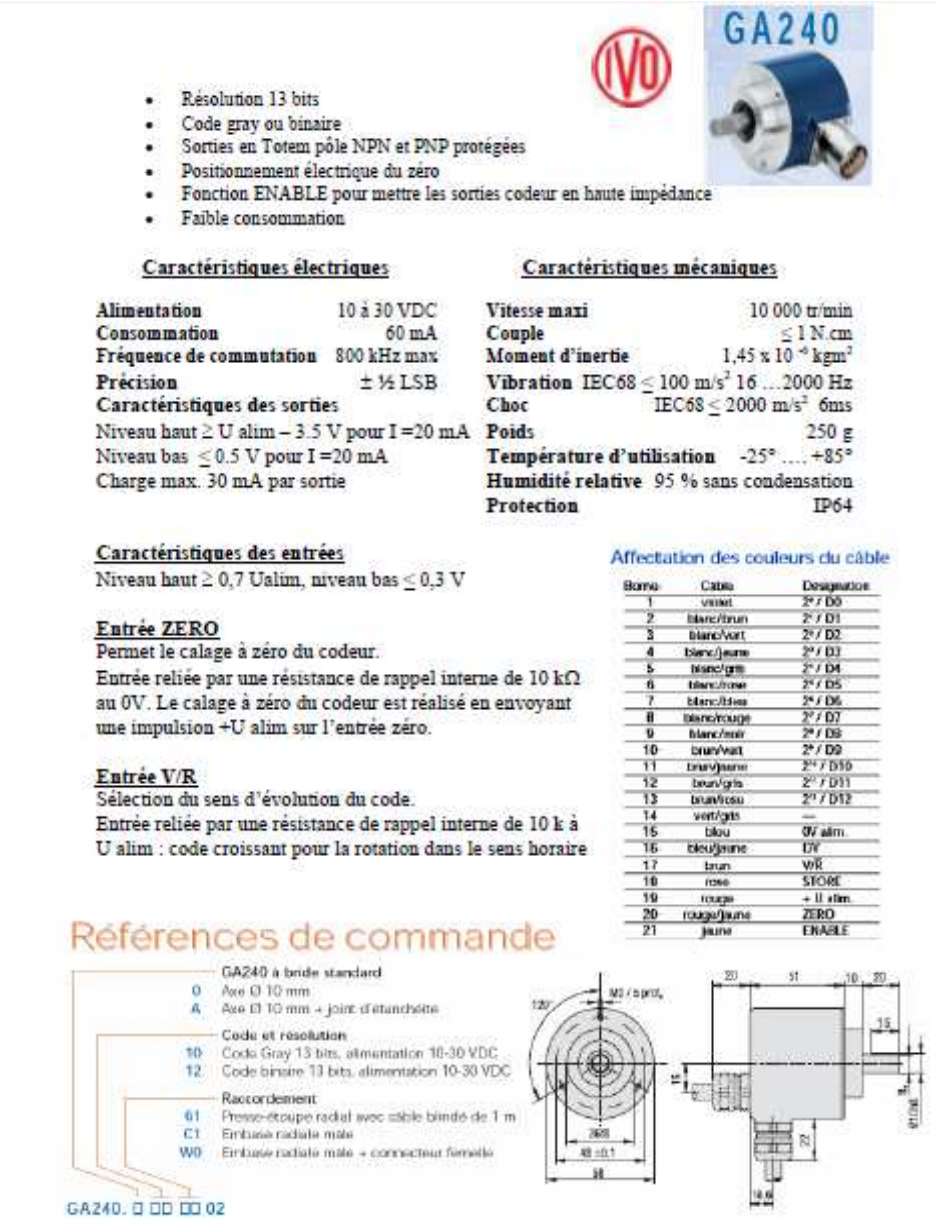


Figure 9 : Description du codeur absolu