

17/08/2017

Paul MOULINS

Polytech Montpellier

Rapport De Stage

Mise en fonctionnement du robot hexapode R.HEX

Table des matières

[Avant-Propos 2](#_Toc490745910)

[Introduction 3](#_Toc490745911)

[A) Présentation du robot R.HEX 3](#_Toc490745912)

[B) Objectif du stage 4](#_Toc490745913)

[I) L’existant 5](#_Toc490745914)

[II) Intégration de l’électronique 5](#_Toc490745915)

[A) Présentation des contraintes 5](#_Toc490745916)

[B) Réalisation du prototype de carte 7](#_Toc490745917)

[III) Le codeur 10](#_Toc490745918)

[A) Présentation du codeur 10](#_Toc490745919)

[B) Principe général du réseau CAN 10](#_Toc490745920)

[C) CANopen et Fonctionnement du capteur 16](#_Toc490745921)

[IV) Paramétrage du codeur 25](#_Toc490745922)

[A) Les registres 25](#_Toc490745923)

[B) Initialisation des capteurs 28](#_Toc490745924)

[V) Simulation Matlab 30](#_Toc490745925)

[VI) Mouvement de la patte 31](#_Toc490745926)

[VII) Liste des composants 31](#_Toc490745927)

[Conclusion : 32](#_Toc490745928)

[Annexe : 34](#_Toc490745929)

# Avant-Propos

Je tiens à présenter mes remerciements aux personnes qui m’ont aidé dans la réalisation de ce stage.

Tout d’abord, Je tiens à remercier mes encadrants du Laboratoire d’Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Madame Karen GODARY-DEJEAN, Monsieur René ZAPATA ainsi que Monsieur Pascal LEPINAY pour l’aide qu’ils m’ont apporté tout au long du stage. Je souhaiterai remercier également Monsieur LATORRE pour ces conseils ainsi que Monsieur Olivier MOÏSE pour son soutien au cours des réalisations techniques de mon stage.

Je souhaite aussi manifester tout particulièrement ma gratitude à l’égard de Monsieur Éric Dubreuil qui a su se montrer disponible et à l’écoute et dont l’aide m’a permis d’aborder efficacement les étapes clés de ce projet.

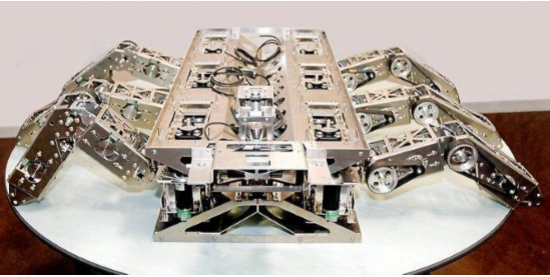
# Introduction

## Présentation du robot R.HEX

Le robot hexapode R.HEX est l’un des projets de l’équipe EXPLORE du département de Robotique du LIRMM. Leur mission est la conception et le développement des outils théoriques et expérimentaux de la robotique mobile et d’exploration de l’environnement.

L’objectif du projet est de créer un robot hexapode inspiré des fournis, de grande taille, avec de grandes capacités motrices, sensorielles et cognitives.

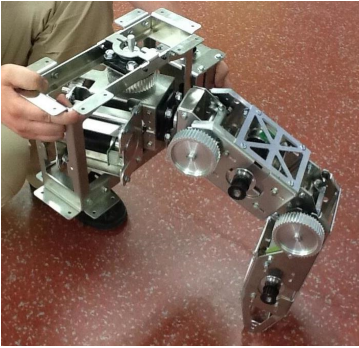
Le robot fait environ 1m20 d’envergure, pèse 60 kilos et possède quatre articulations par patte. L’ensemble est mû pour 27 moteurs.



Structure mécanique de l'hexapode

« Sa principale mission sera d’évoluer dans des environnements fragiles pour des missions d’intervention ou de mesure. Sa structure mécanique, et notamment l’extrémité ponctuelle de ses 6 pattes lui permettront d’avoir un impact minime sur son environnement contrairement à des robots à roues ou à chenilles. Ses différents capteurs et son intelligence embarquée lui fourniront la modélisation en temps réel de son environnement nécessaire à l’élaboration du chemin à parcourir pour réaliser sa tâche. Finalement, sa redondance lui apportera une agilité à se mouvoir et sa capacité de charge rendra possible l’apport de matériel ou le transport de déchets. »

En ce qui concerne les pattes de l’hexapode, celle-ci sont composées de trois segments et de quatre degrés de libertés, deux degrés sur le premier segment, puis un par segments. Les positions de chaque articulation sont données par un codeur absolu utilisant un protocole de communication CANopen. Un étudiant de 4eme année a déjà eu l’occasion de travailler sur une septième patte de test et d’asservir celle-ci.



Une des six pattes de du robot

## Objectif du stage

La mission qui m’a été confié peut se résumer ainsi :

-Mettre au propre et améliorer le système électronique existant permettant le contrôle des pattes de l’hexapode et le rendre générique afin de faciliter sa mise en place et son entretient sur le robot.

-Réaliser une modélisation du robot sur Matlab afin de visualiser les différentes positions dans l’espace atteignables par celui-ci.

-Rendre fonctionnelles les six pattes de l’hexapode (câblage, programmation des codeurs et des drivers de moteur).

-Initier un mouvement de marche sur le robot.

Ce rapport présente dans un premier temps les différentes étapes menant à l’intégration de l’électronique sur chacunes des pattes du robot hexapode, ainsi que les contraintes liées à l’architecture des pattes. Dans un second, on abordera le codeur, son mode de communication ainsi que son fonctionnement général. On verra par la suite les moyens mis à notre disposition pour les programmer. Enfin, on présentera dans la dernière partie le programme Matlab modélisant le robot pour en déduire son espace articulaire.

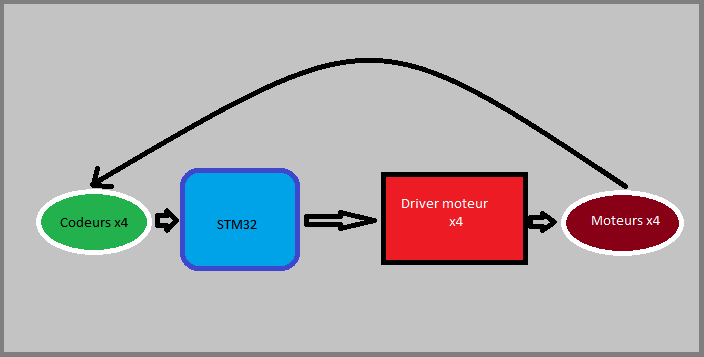
# L’existant

Au début de mon stage, la structure mécanique du robot était achevée, des simulations ont été faite sur un environnement virtuel 3D (auxquelles je n’ai pas eu accès) et un étudient avait déjà eu l’occasion de travailler sur une patte du robot. Celui-ci a, dans un premier temps, lu les valeurs données par les capteurs de positions angulaires (utilisant le protocole CANopen) puis dans un second temps, contrôlé les moteurs grâce à des commandes en PWM. Cet étudient a fourni un rapport sur le contrôle des codeurs ainsi qu’un code permettant de contrôler la patte avec une STM32 et des potentiomètres. Plusieurs choses sont cependant à noter au regard des rendus. Le rapport qui est, il faut le remarquer, d’une aide notable pour la compréhension du fonctionnement des capteurs, se concentre malheureusement plus sur le principe de fonctionnement théorique du CAN et du capteur que sur l’explication de la mise en œuvre du contrôle de la patte en elle-même. De plus le code n’étant pas du tout commenté il a été très difficile de reprendre et de remettre en place les résultats déjà obtenus par ce précédent étudient.

# Intégration de l’électronique

## Présentation des contraintes

L’ensemble du système permettant la mise en mouvement d’une patte peut être schématisé de la façon suivante :



On utilise dans ce montage des codeurs absolus monotour Sendix M3658 de Kübler communiquant par protocole CANopen alimentés en 24V.

Lors du projet précédent une STM32F429zi avait été choisie pour sa capacité à gérer deux bus CAN.

Les moteurs, pour être pilotés ont besoin d’être connectés en amont à des drivers. On ne s’attardera pas vraiment sur ces derniers dans la mesure où l’on réutilisera, sans rien modifier, les paramètres donnés sur les premiers drivers de la patte de test (Copier/coller). Ceux-ci sont alimentés en 24V et transmettent cette alimentation directement aux moteurs.

Au commencement de mon stage, les branchements n’avaient été réalisés que sur la septième patte de test fixée sur une table. Les fils couraient sur la table, des codeurs et des drivers, jusqu’à la carte faisant office de shield à la STM. Le rôle de cette carte n’était que de produire ou de traiter les différents signaux logiques nécessaires au fonctionnement de la patte sans se soucier de l’alimentation des codeurs ou des drivers. La carte possédait également une entrée pour brancher des potentiomètres destinés au contrôle « manuel » de la patte.

L’alimentation du système se faisait par deux arrivées, une première en 24V délivrée par une alimentation de laboratoire et une seconde en 5V sur la STM délivrée par le port USB de l’ordinateur.

Cette configuration plutôt hasardeuse n’étant pas du tout adaptée à une intégration sur le robot, mon stage a alors eu pour but de revoir la mise en place des différents éléments afin de les embarquer sur l’hexapode.

Il a été décidé, pour faire se mouvoir le robot, de garder un contrôleur par patte comme sur celle de test puis de relier l’ensemble des pattes à un contrôleur central gérant le processus de marche.

Trois problèmes se posent alors une fois cette architecture choisie. Premièrement, nous sommes soumis à une contrainte d’encombrement. En effet comme chacun peut le deviner en regardant l’hexapode, l’espace disponible sur le dessus du robot n’est pas vraiment développé. De nombreuses contraintes mécaniques sont à prendre en compte et il ne faut pas non plus oublier la quantité importante de câbles reliant les codeurs, les drivers, les moteurs et le contrôleur.

L’étape de câblage faisant partie intégrante du projet et représentant un temps d’investissement considérable, il a été décidé de rendre chaque élément le plus générique possible : Câbles de mêmes longueurs pour chaque tronçon de patte, connecteurs adaptés au type de signaux transmis et, surtout, UN seul type de structure à fixer sur les pattes.

Troisièmement, nous somme aussi soumis à une contrainte de poids. D’après les remarques faites par mes encadrants, il serait possible que certaines pièces de la structure métallique ne résistent pas à l’effort appliqué sur celle-ci lors du levé ou de la marche. Il est donc nécessaire de garder à l’éprit que n’importe quel élément sera et devra être sextuplé par la suite.

## Réalisation du prototype de carte

La nouvelle carte se base sur l’ancienne réalisée lors du précédent projet. De cette carte, on garde les deux transceiver CAN permettant la transformation des signaux CAN Low et CAN Hight (cf. protocole CAN et CANopen) en deux signaux Rx/Tx compréhensibles par la STM. On décide cependant d’ajouter à cette carte certains éléments supplémentaires.

Les alimentations des codeurs, des actionneurs et du contrôleur qui se faisaient avant de façon externe se feront maintenant directement sur la carte, qui devra pourvoir convertir une unique entrée en 24V en une alimentation en 5V pour la STM. On utilise pour ce faire un TracoPower Tel 3-2411, sa fiche technique est donnée en annexe. Remarque : l’utilisation d’un convertisseur isolé n’est pas nécessaire dans notre cas. Ici la sortie « V- » (-2.5V) du Traco est connectée à la masse du montage pour amener un potentiel de 5V aux bornes de la STM.

Il est important de noter que le routage actuel des cartes en ce qui concerne l’alimentation des STM n’est pas parfait. En effet l’entrée Vin de la STM ne peut être alimentée qu’en 7.5V minimum, il faut donc l’alimenter en externe pour utiliser le 5V à notre disposition. Pour basculer l’alimentation en externe il faut déplacer le jumper JP3 en « E5V » et non plus en « Vin ». On relie ensuite la pin E5V de la STM à l’alimentation 5V de la carte.

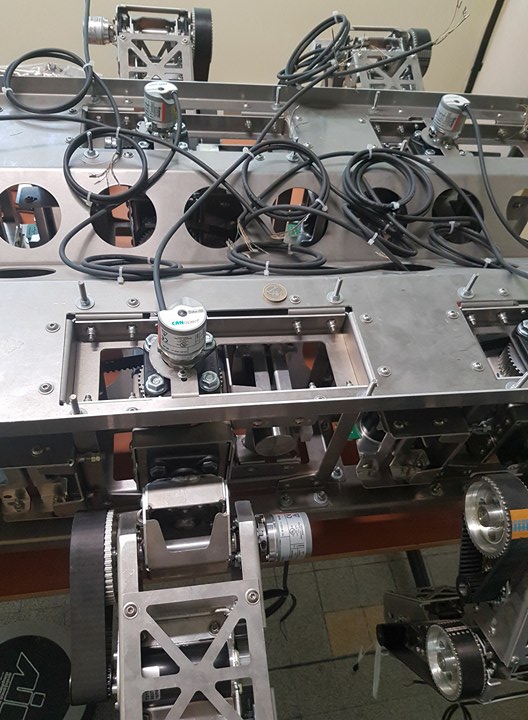
Il a également fallu définir une connectique appropriée pour les bus CAN. Après recherches, il est apparu que les connectiques RJ45 et DB9 étaient celles les plus utilisées dans le domaine du CAN. Le RJ45 a été privilégié par rapport au DB9 aux vues de l’encombrement et du poids de ce dernier.

On souhaite intégrer également sur la carte tout le circuit électrique relatif à la connexion des quatre drivers entre eux et à la STM.

De plus, on garde les potentiomètres et un accès aux signaux logiques des bus CAN.

Une fois les composants et les caractéristiques de la carte définis il a fallu réfléchir au placement de la carte sur le robot puis à la façon de disposer les différents éléments sur celle-ci.

Les pattes de l’hexapode se présentent comme ceci :



Vue du dessus de la première patte gauche

En prenant en compte le passage des câbles, on constate que les seuls emplacements disponibles se situent à l’arrière des premiers codeurs de la patte. On se retrouve alors considérablement contraint pour l’espace et le placement des fixations. Par soucis de symétrie et de longueur minimale de câble (poids) on choisit de placer les interfaces des drivers toujours vers l’axe central du robot. Pour maximiser l’espace et après plusieurs essais, la disposition retenue a été d’empiler les drivers les uns sur les autres et de les maintenir ensemble par des tiges métalliques fixées à la carte. La STM sera fixé sous cette carte.

Toujours dans un souci de généricité et de symétrie on souhaite ne réaliser qu’une seule carte pour toutes les pattes, on doit donc toujours positionner les cartes selon la même orientation. Pour des raisons tout à fait arbitraires on place l’alimentation générale coté capteur, on place par contre les prises RJ45 vers l’arrière de l’hexapode pour un accès facile aux deux bus CAN et au port RJ45 de la STM. Vu que les drivers des moteurs sont toujours orientés vers l’intérieur et que les cartes sont toutes identiques, l’alimentation des moteurs et la logique allant aux drivers doivent avoir une place adaptée aux configurations gauches et droites. Puisque l’on suppose que l’on ne débranchera pas l’alimentation des drivers au niveau de la carte, on place leurs alimentations au centre de celle-ci, sous l’empilement de carte. Les signaux logiques sont quant à eux placés sur le bord de la carte afin d’y avoir facilement accès en cas de lecture à l’oscilloscope. Potentiomètres et sorties CAN sont placés à côté.

Le schéma électrique du montage et le routage du prototype final sont à consulter en annexe.

***Remarques*** ***:*** Concernant le montage, cette architecture en étage est FORTEMENT propice aux courts circuits, surtout entres les drivers moteur. Il conviendrait donc à l’avenir de rajouter entre les quatre carte une couche d’isolant (intercalaires en pastique, par exemple).

# Le codeur

## Présentation du codeur

On a choisi un codeur absolu monotour Sendix M3658 de Kübler avec capteur magnétique. Le terme « absolu » désigne le fait que l’angle soit mesuré par rapport à une position de référence et « monotour » que l’on est capable de connaitre la position du capteur qu’entre 0 et 360 degrés.

Le codeur est basé sur le réseau CAN avec interface CANopen (couche application). Concernant la communication avec celui-ci on retrouve 5 fils, le vert, le jaune, le gris, le blanc et le marron qui correspondent respectivement aux CAN\_HIGH, CAN\_LOW, CAN\_GND, 0V et V+.

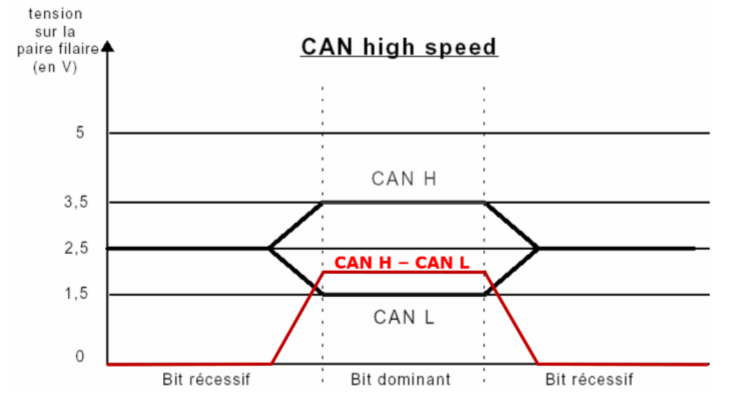
La résolution de l’angle est sur 16bits, on a donc une valeur de 0 à 65535. La vitesse de rotation maximale est de 6000tr/min et l’alimentation peut se faire sur une plage de 8-25V pour une consommation d’environ 25mA.

## Principe général du réseau CAN

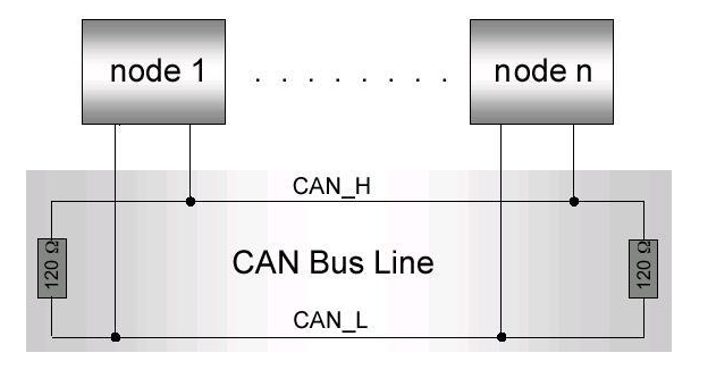
Le réseau CAN est un réseau série composé de 2 fils de données : CAN HIGH et CAN LOW. Les états logiques sont codés par différence de potentiel entre les 2 fils.

Deux configurations existent : un mode à basse vitesse « CAN low speed » jusqu’à 125Kb/s et un mode à haute vitesse « CAN high speed » allant jusqu’à 1Mbit/s. Les tensions sur les fils varient en fonction des modes utilisés. Notre système fonctionne en CAN high speed avec une vitesse de 1000 Kbit/s.

Dans une trame CAN, on parle de bit récessif lorsque la différence des tensions est de 0V ; il correspond à un 1 logique. On parle de bit dominant quand on a 2V ; il correspond à un 0 logique.



Un bus CAN peu se représenter come ceci :



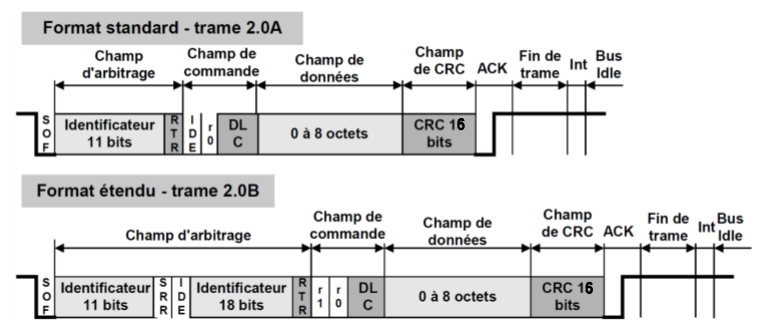
*Il est important de noter la présence d’une charge en bout de ligne.*

Le protocole CAN comprend deux versions, CAN 2.0.A et CAN 2.0.B. la seule différence entre les deux est la longueur de l’identifiant qui passe de 11bits pour le A à 29bits pour le B.

Il existe plusieurs types de trame :

* **Data Frame** : trames transportant des données d’un producteur vers des consommateurs, sans garantie de traitement.
* **Remote Frame** : trames de requête en polling émises par un maître vers un ou plusieurs esclaves, pour demander le renvoi d’une trame de données.
* **Error Frame** : trames émises lorsqu’une station détecte une erreur de transmission sur le bus.
* **Overload Frame** : trames émises pour demander un laps de temps supplémentaire entre des trames (de données ou de requête) successives.

Une trame se compose de la manière suivante :



La trame de données sert à envoyer des informations aux autres nœuds.

Une trame de données se compose de 7 champs différents :

* Le début de trame ou SOF (Start Of Frame) matérialisé par 1 bit dominant,
* Le champ d'arbitrage (identificateur) composé de 12 ou 30 bits,
* Le champ de commande (ou de contrôle) composé de 6 bits,
* Le champ de données composé de 0 à 64 bits (de 0 à 8 octets),
* Le champ de CRC composé de 16 bits,
* Le champ d'acquittement composé de 2 bits,
* La fin de trame ou EOF (End of Frame) matérialisée par 7 bits récessifs.

#### Ordre de transmission des bits

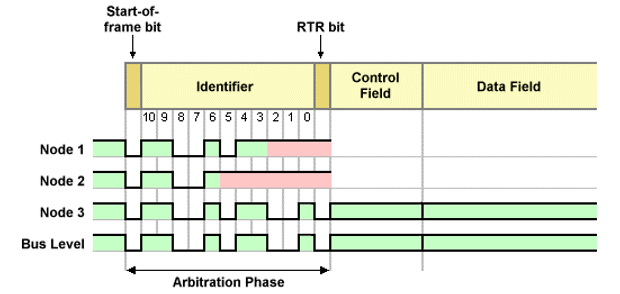
Les champs sont transmis dans l'ordre du SOF à l'EOF. Dans chaque champ de la trame, les bits sont transmis du plus fort au plus faible.

#### Champ d'arbitrage

Le champ d'arbitrage est composé de 11 bits d'identification pour CAN 2.0A et 29 bits pour CAN 2.0B suivis par le bit RTR (Remote Transmission Request) qui est dominant.

Ce champ sert d'identifiant pour la donnée transportée dans le champ de données.

Si plusieurs messages provenant de nœuds différents sont émis en même temps, il faut pouvoir définir lequel des messages sera émis. Voici un schéma montrant le principe de sélection :



On comprend alors que plus l’identifient est petit, plus le message est prioritaire.

#### Champ de commande

Le champ de commande est composé de 6 bits.

Le bit de poids fort est utilisé pour différencier le type de trame :

Dans le cas d'une trame standard (sur 11 bits), le bit de poids fort est dominant,

Dans le cas d'une trame étendue (sur 29 bits), le bit de poids fort est récessif,

Le bit suivant n'est pas utilisé.

Les 4 bits de poids faibles appelés DLC (Data length Code) représentent le nombre d'octets du champ de données (PAYLOAD) embarqué.

Ce nombre d'octets peut varier de 0 à 8, soit 9 valeurs stockées avec les 4 bits du champ DLC. Les valeurs DLC supérieures à 9 ne seraient donc pas utilisées (de 9 à 15).

#### Champ de données

Le champ de données peut varier de 0 à 8 octets.

Dans le cas d'une trame de requête le champ de données est vide.

#### Champ de CRC

Le champ est composé de 15 bits de CRC (Cyclic Redundancy Check) et d'un bit dit délimiteur (« CRC delimiter ») toujours récessif.

Le CRC est calculé à partir de l'ensemble des champs transmis jusque-là (c'est-à-dire le SOF, le champ d'arbitrage, le champ de commande et le champ de données; les bits de transparence ne sont pas pris en compte). L'ensemble constitue le polynôme f(x).

L'algorithme consiste dans un premier temps à multiplier f(x) par 215.

Ensuite le polynôme f(x) est divisé (modulo 2) par le polynôme g(x)=x15+x14+x10+x8+x7+x4+x3+x0.

Une fois les divisions successives effectuées, le reste constitue la séquence de CRC.

La distance de Hamming de l’algorithme utilisé est de 6, ce qui signifie que 5 erreurs au maximum sont détectables.

Grâce à ce système de détection, le taux d’erreur enregistré est très faible. De plus, le réseau est capable de différencier les erreurs ponctuelles des erreurs redondantes. Ainsi, tout périphérique défaillant peut être déconnecté du réseau afin de limiter les perturbations. Le réseau entre alors en mode « dégradé ».

#### Champ d'acquittement ACK

Le champ est composé d'un bit d'acquittement ACK (ACKnowledge) et d'un bit dit délimiteur (« ACKnowledge delimiter ») toujours récessif.

Tous les récepteurs qui ont bien reçu le message doivent l'acquitter en émettant un bit dominant pendant la durée du bit ACK, ce qui permet au nœud émetteur de savoir qu'au moins un des nœuds récepteurs a reçu le message.

Si un nœud récepteur n'a pas ou mal reçu le message, il ne peut pas se servir de ce mécanisme pour signaler l'erreur, puisqu'il suffit qu'une station réceptrice envoie un bit dominant pour masquer tous les bits récessifs. Pour signaler le dysfonctionnement, il doit émettre une trame d'erreur.

#### Fin de trame

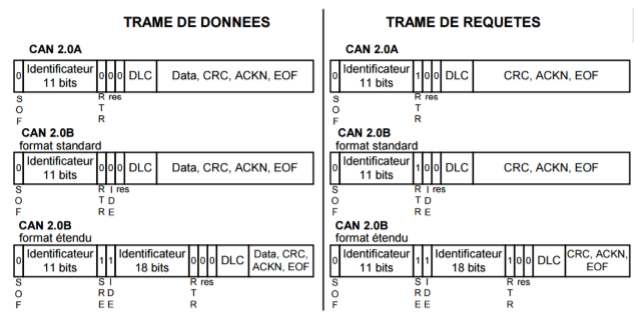
Suite de 7 bits récessifs qui indiquent la fin de la trame.

#### Intertrame

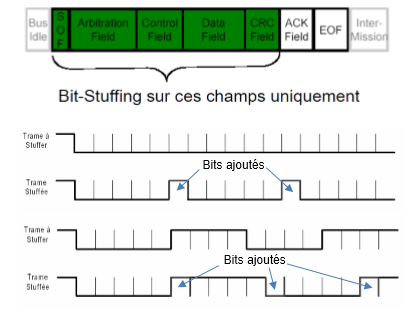
Suite de 3 bits récessifs qui séparent 2 trames consécutives.

Les trames qui nous intéressent sont celles de requêtes et de données.

Les trames de données sont structurées de la même manière que précédemment. Les trames de requête ne comportent pas de champ de données (donc DLC = 0) et le bit RTR doit être récessif.



Le dernier point à prendre en compte est le « bit-stuffing ». Après 5 bits identique, un bit de remplissage de niveau inverse est ajouté.



## CANopen et Fonctionnement du capteur

Le CANopen est un protocole de communication s’appuyant sur une couche applicative du réseau CAN. Le profil de communication est défini sous l’appellation CIA DS301.

Toutes les informations données dans cette partie sont extraites de deux documents fournis par Kuebler sur la page internet du codeur : le « CANopen Application Note » et le « Technical Manual ». Le premier explique les différents types de communication et donne des exemples des trames alors que le second est le document référent en ce qui concerne le protocole CANopen et les dictionnaires d’objets.

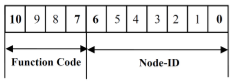
#### Dictionnaire d’objets

Un codeur peut se programmer en modifiant certaines valeurs présentes dans sa mémoire EEPROM. La mémoire contient ce que l’on appelle un dictionnaire d’objet, cela correspond à une série de registres accessible pour certains en lecture ou encore en écriture. On y accède en envoyant un type de trames composées de plusieurs éléments. Un registre est défini par le biais d’un index de 2 octets et d’un sous-index de 1 autre octet, soit 3 octets au total.

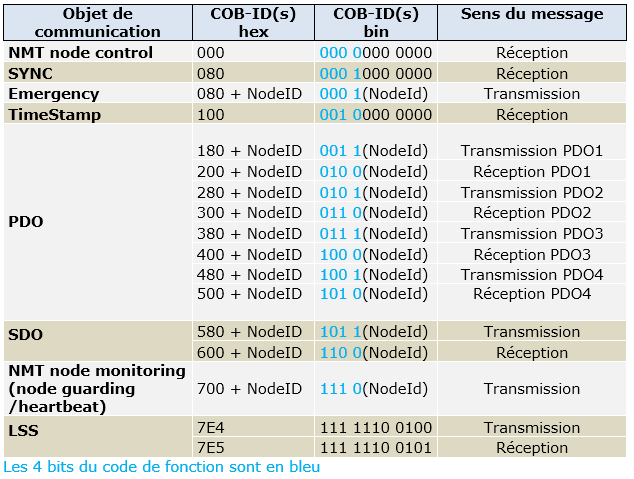
On accède plus généralement à ces registres en envoyant, puis en recevant, des trames composées de plusieurs éléments. Il existe plusieurs types de demandes qui entrainent plusieurs types de réponses, cependant on retrouve toujours au minimum : un identifient COB, une instruction puis un index et son sous-index.

#### L’identifiant COB

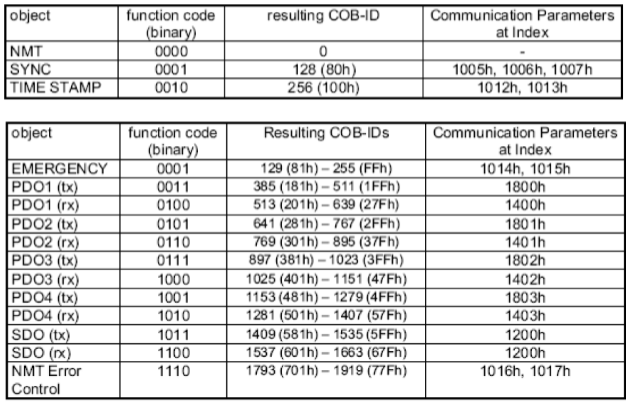
Cet identifiant est composé d’un code définissant la fonction de la trame (4 bits) et d’un numéro de nœud (7 bits). Le numéro de nœud est unique et défini chaque capteur, c’est avec cette information là que l’on s’adresse à lui. La valeur peut varier de 0x01 à 0x7F, soit de 1 à 127 en décimal. L’adresse 0x00 est, quant à elle, une adresse ultra-prioritaire servant à propager une information sur l’ensemble du bus (node par node). L’identifiant de base est le 0x3F, il peut être modifié par un accès SDO.



Le code de fonction spécifie le type de communication, on peut lister l’ensemble suivant :



Les identifiants sans nœud représentent les objets de broadcaste, leur priorité est maximale.



#### Les types de communication

Comme on peut le voir dans les tableaux précédents, il existe plusieurs types de communication. On retrouve en CANopen quatre types différents :

* NMT, Heartbeat et Life-Guarding :
* SYNC, Emergency, Timestamp :
* SDO : Paramétrage du codeur
* PDO : Transmission de données en temps réel

Il faut bien se rendre compte que ces éléments se composent tous de la même façon : identifient + n° registre + Data. Seules les valeurs véhiculées sont différentes.

*Conseil* : *garder en main le document « CANopen Application Note » pour illustrer cette partie.*

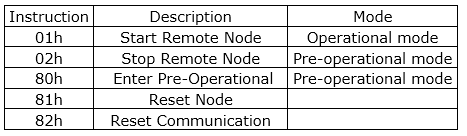
###### Instruction MNT

Ces instructions gèrent le réseau, on définit le mode de fonctionnement des codeurs en leur transmettant des instructions NMT. Celles-ci se caractérisent par un identifient de valeur 0 et donc le plus prioritaire possible.

L’instruction est composée de l’identifiant 0x000 et de deux autres octets.



Les instructions possibles sont les suivantes :



Pour faire simple, on peut programmer le codeur en mode préopérationnel et le mettre en fonctionnement en lançant le mode opérationnel. On fera bien la différence entre la première mise en mode préopérationnel et une mise en mode préopérationnel après un mode opérationnel.

###### SDO : Service Data Object

Ce type de trame permet un accès en lecture ou en écriture (pour certains registres) au dictionnaire d’objets.

Un objet est caractérisé par un index et son sous-index et par une valeur. La totalité des objets ont une valeur par défaut définie en paramètre d’usine. On peut retrouver la liste de ces objets dans le manuel distribué par Kuebler.

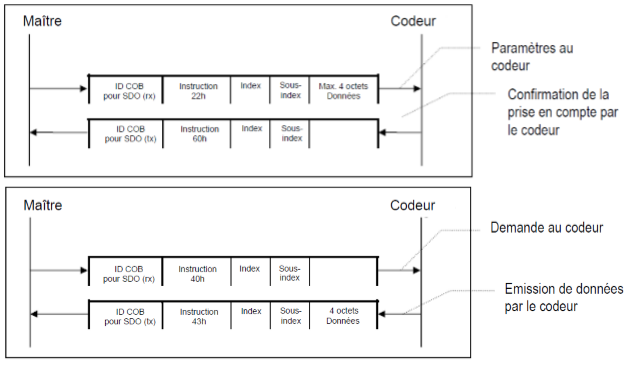
En se reportant aux tableaux présentés précédemment on voit que l’on dispose de 2 identifiants COB pour émettre et recevoir des trames :

- SDO (tx) (Codeur vers Maître) : 580h + Numéro de nœud

- SDO (rx) (Maître vers Codeur) : 600h + Numéro de nœud

Le codeur nous renvoie une confirmation pour chaque trame reçue.

Ci-dessous un exemple de transmission :



Le principe est toujours le même :

- Du maître vers le codeur :

* On envoie l’identifiant (600h+numéro du codeur),
* On ajoute un octet d’instruction (dépend de la taille en écriture, 40h en lecture),
* On définit l’index et le sous index,
* On joint la data en cas d’écriture, rien en lecture.

- Du codeur vers le maître :

* On reçoit l’identifiant (580h+numéro du codeur),
* On lit l’instruction (60h en confirmation d’écriture, dépend de la data en lecture),
* On récupère le numéro de l’index et du sous index,
* Si lecture on récupère la data, sinon rien.



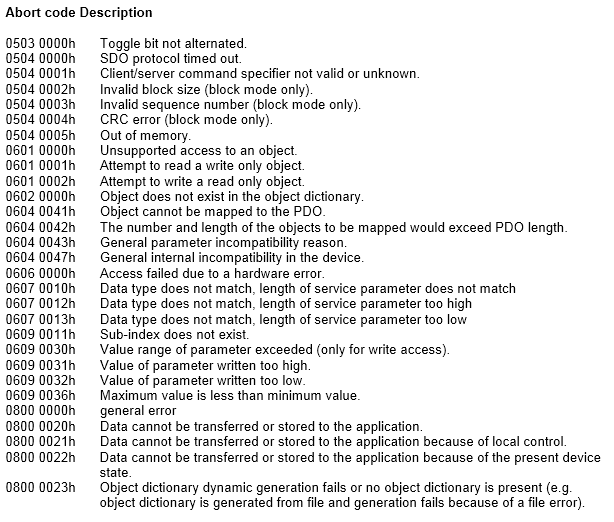
L’instruction permet de préciser la taille de la donnée qui peut être de 1 à 4 octets. Le DLC du CAN va donc de 5 à 8.

Comme vu précédemment, chaque envoie de trame a pour retour une réponse de l’encodeur. Lors d’une écriture, le codeur peut renvoyer deux instructions différentes :

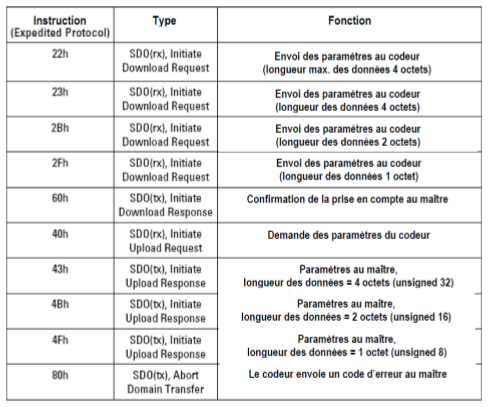
- 60h : Indique que l’objet de communication reçu par le codeur est correct.

- 80h : Signifie qu’il y a une erreur dans l’objet reçu.

Lorsque l’on récupère un message d’erreur, le champ de la Data contient un message pouvant permettre de retrouver l’origine de l’erreur.



Voici un tableau regroupant la liste des instructions possibles :



###### PDO : Process Data Object

On utilise ce type de communication pour échanger des données en temps réel. Il existe trois types de transmissions différentes (TPDO) et trois autres types de réceptions (RPDO).

Un PDO est décrit par 2 objets dans le dictionnaire :

- Le PDO Communication Parameter : Indique comment est transmis l’objet. L’index de cet objet prend les valeurs 1800h, 1801h et 1802h dans le tableau.

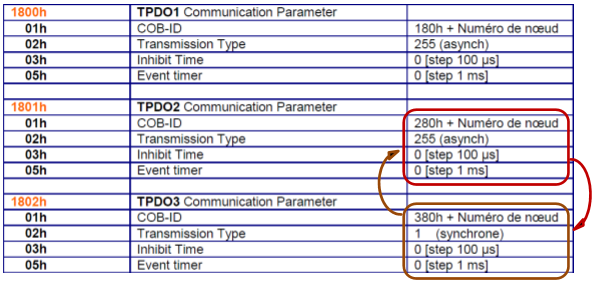
- Le PDO Mapping Parameter : Indique quelles sont les données transportées. L’index de cet objet prend les valeurs 1A00h, 1A01h et 1A02h dans le tableau.

L’objet index 6004h, sous-index 00h correspond à la position du codeur et l’objet index 6030h, sous-index 01h correspond à la vitesse du codeur.

***Attention : le manuel présente une erreur ! La vitesse est mappée sur l’objet 1A02h et non dans l’objet 1A01h !***

PDO Communication Parameter :

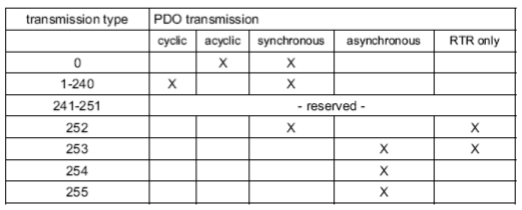
Cet objet indique la façon dont est transmis le PDO, l’information est répartie sur plusieurs sous index.



***Attention : le manuel présente une inversion !***

On transmet en premier l’identifiant de l’objet et le type de transmission (de 0 à 255), puis la durée minimale entre deux émissions de message et enfin la période d’émission de la valeur de l’objet. La période d’évènement est commandée par une horloge interne au codeur, la période peut aller de 1ms à 65535ms. Cette fonction n’est utilisable que si la valeur du type de transmission est égale à 254 ou 255.

Voici la liste des différents types de transmissions possibles :

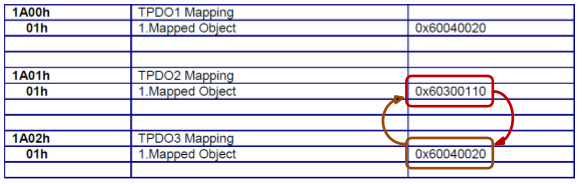


Une valeur comprise entre 1 et 240 signifie que le PDO sera envoyé de manière synchrone et cyclique. Cette valeur représente la quantité d'impulsions SYNC nécessaires pour transmettre les PDO. Un type de transmission de 252 ou 253 indiquent que le PDO ne sera envoyé que sur demande via une RTR.

Le type 254 signifie que l'événement sera déclenché en fonction de l'application (spécifique à l'application), tandis que le type 255 dépend du périphérique (spécifique au périphérique).

On peut modifier ces paramètres grâce à des SDO.

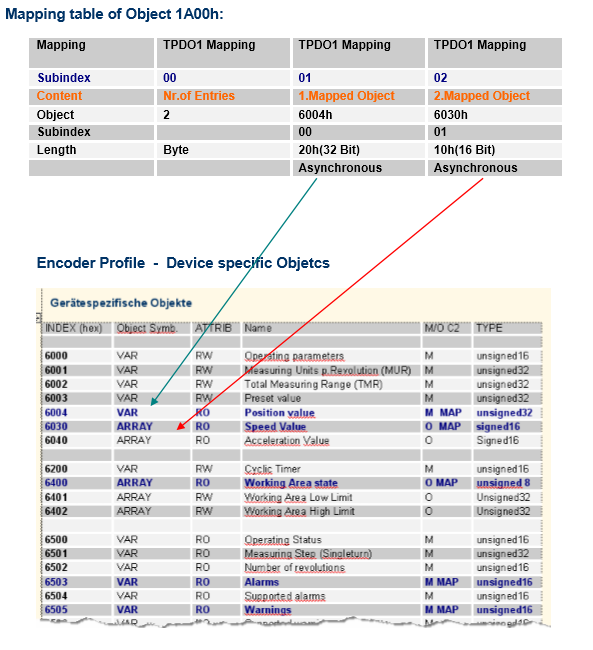
PDO Mapping Parameter :



***Attention : le manuel présente une inversion !***

L’objet 1A00h permet de mapper le PDO n°1, l’objet 1A01h le PDO n°2 et l’objet 1A02h le PDO n°3.

Il est possible de mapper autant d’objets qu’il en existe, tant que la longueur maximale des données ne dépasse pas 8 octets.

Par exemple, ci-contre :

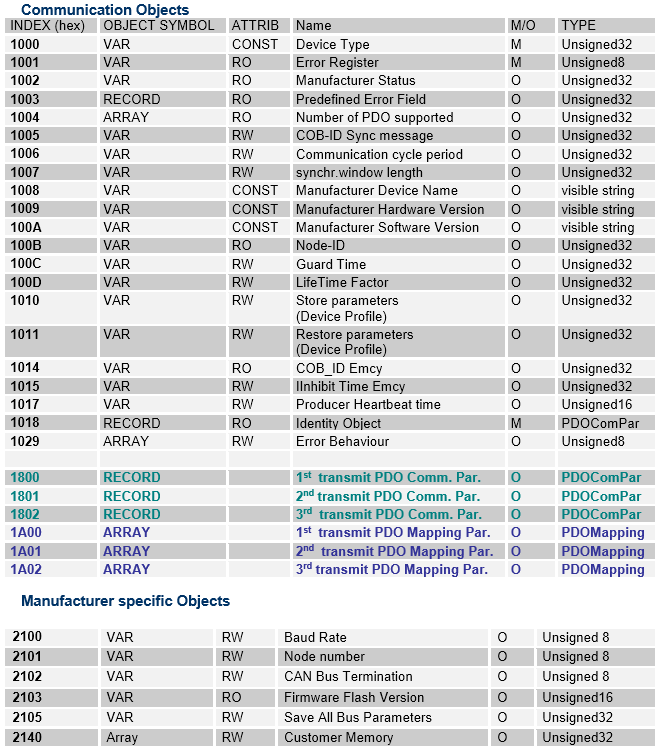
# Paramétrage du codeur

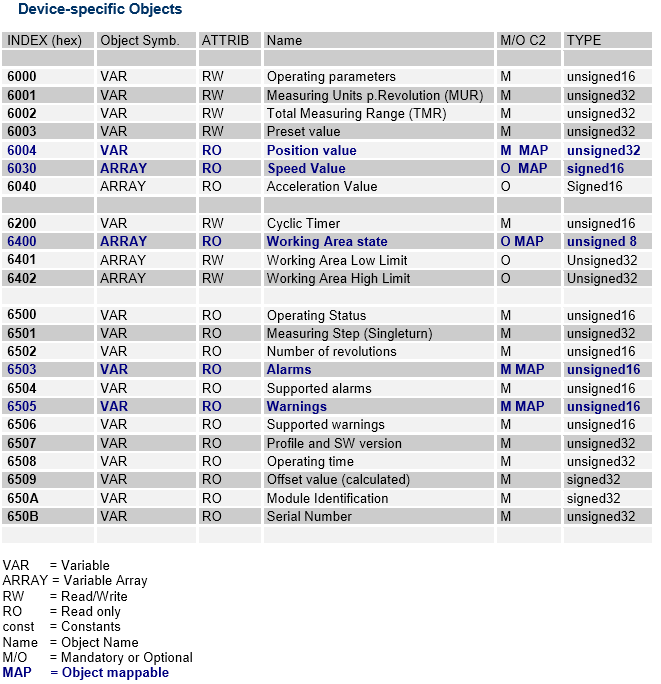
Maintenant que nous avons vu le principe général de programmation nous allons aborder plus en détail le type d’information que nous envoyons au codeur afin de l’initialiser et récupérer ses valeurs. Dans un premier temps nous allons voir les registres à écrire puis, dans un second temps, voir comment est fait le code d’initialisation des capteurs.

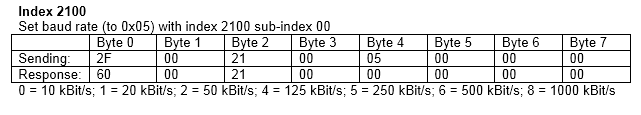
## Les registres

Lors de la première mise sous tension, le codeur est chargé avec les paramètres d’usine. La note applicative fournie par Kuebler peut être très utile pour comprendre cette partie.

Voici la liste de l’ensemble des registres de la bibliothèque :





Par exemple, pour programmer la vitesse de transmission du codeur, on trouve les informations suivantes :

Ce tableau présente les champs :

- « Instruction », Octet 0.

- « Index », Octet 2 et 1 (Remarque : attention à l’ordre…).

- « Sous-Index », Octet 3.

- « Data », Octet 4 à 7.



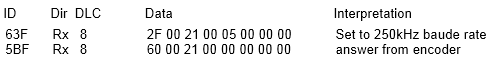
***Il ne manque que la partie identifiant COB à rajouter au début pour former la trame complète à envoyer.***

On se rappellera également que l’instruction indique le nombre de Data présentes.

Pour un accès en écriture au niveau du maitre :



On a donc au final ceci pour l’envoi au capteur et sa réponse associée :



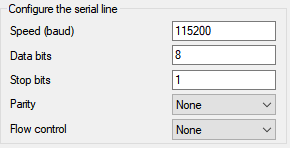
## Initialisation des capteurs

Cette partie ne prétend pas expliquer le fonctionnement du code qui est en soit suffisamment commenté de part lui-même. On ne décrira ici que les étapes aboutissant à un codeur fonctionnel.

Deux programmes sont à notre disposition, un pour l’initialisation des codeurs et un autre pour faire bouger une patte vers une position donnée. On ne discutera pas de ce dernier ici dans la mesure où un détail du premier et les commentaires qui lui sont appliqués permettent de comprendre le fonctionnement de celui-ci sans problème.

La programmation des codeurs se fait capteur par capteur. Pour le montage, on procède de la façon suivante :

* **RETIRER TOUTE ALLIMENTATION 24V DU MONTAGE !**
* Prendre le prototype, déconnecter tout ce qui pourrait l’être aux deux RJ45.
* Prendre un des câbles « RJ45 – ‘’Nappe’’ », connecter une extrémité au prototype **SUR LA PRISE CAN1** (cf. la carte, celle la plus au-dessus de la RJ45 de la STM). Connecter l’autre extrémité à la « multiprise » de jonction d’un bus.
* Connecter **UN SEUL CODEUR** à la sus-citée multiprise.
* Brancher la STM à un ordinateur et ouvrir un terminal :



**Attention : Lire attentivement toute la phrase suivante**

* Mettre sous tension le prototype puis faire attention à ne pas encore toucher au **Bouton Bleu**
* Lire la partie suivante **PUIS** suivre les informations du terminal.

Le principe de fonctionnement du programme est simple. Lorsque l’on (*Ne pas encore appuyer dessus*) clique sur le bouton bleu de la STM, un compte à rebours de 10 secondes est lancé ce qui nous permet de sélectionner le type de paramétrage que l’on veut réaliser. Le programme permet de donner un nom au codeur, soit 3F (qui et le nom de base), soit 3E (choix arbitraire de nom). On applique également d’autres paramétrages que l’on détaillera par la suite.

Lorsque le compte à rebours est lancé, il nous reste 10 secondes pour choisir un mode de programmation en cliquant sur le bouton bleu :

* Ne rien toucher… ne fait rien…
* Appuyer une seule fois programme le codeur en « 3F »
* Appuyer une seule fois programme le codeur en « 3E »

**Bien vérifier dans le terminal le choix effectué !**

On peut alors redémarrer le programme en cliquant sur le bouton reset.

**Pour sauvegarder** le paramétrage, il faut mettre hors tension le codeur.

Lors de l’initialisation on modifie les paramètres suivants :

* La vitesse de transmission : 250kHz à la base, 1000kHz maintenant.
* Le nom du nœud : « 3F » à la base.
* La résistance de terminaison : Active à la base, Désactivée maintenant.

On pourrait aussi modifier le sens de comptage, la vitesse de lecture des données, etc. cependant, cela sera laissé aux soins de ceux qui auront la charge de reprendre le projet. La configuration actuelle est *suffisante* pour faire fonctionner une patte. Ces autres réglages peuvent être effectués dans un souci de confort.

# Simulation Matlab

Afin de faire se mouvoir le robot il a été décidé de mettre en place un simulateur pour pouvoir tester différents types de mouvements. Tout d’abord, puisque l’on dispose sur le robot des valeurs articulaires des pattes, on se basera sur un modèle géométrique direct de l’hexapode. On cherchera également à connaitre l’espace articulaire du robot puis, dans un dernier temps à modéliser une approche géométrique inverse du modèle de celui-ci. Le travail réalisé sera fait sous Matlab 2015.

Avant de pouvoir faire bouger l’hexapode, encore faut-il pouvoir modéliser sa structure. L’hexapode se compose d’un corps et de six pattes représentées par un unique modèle de jambe répété six fois en différentes origines. On retrouve dans le répertoire Matlab un ensemble de fonctions et scripts, on affiche la patte dans le fichier « Test\_affichage » et le corps dans « Test\_Corps ».

La composition du corps se base grandement sur celle de la patte. En effet, toutes les fonctions de création, de dessin ou de tracé de repère sont dérivées de celles de la patte. Le corps et le modèle de patte sont décrits sous forme d’une structure reprenant plusieurs éléments tels que par exemples les repères ou les points terminaux. L’ensemble des composantes sont décrites dans les fichiers Matlab.

La modélisation du robot dans son ensemble est faite dans le fichier « Main », on y retrouve le MGD ainsi que la représentation de l’espace articulaire d’une patte.

Le dernier script, appelé « Test\_MGI », se propose de donner un début de solution pour une approche géométrique inverse du problème. Par soucis de simplicité le modèle du MGI ne prend pas en compte la seconde articulation en partant du corps. De plus, par manque de temps seuls quelques essais ont pu être réalisés sans vraiment aboutir à une conclusion correcte.

On peut voir dans le fichier trois tentatives de résolution du MGI, celles-ci ne sont pas concluantes, elles seront cependant laissées puisque cela ne gêne en rien la lecture du fichier. Le problème pourrait venir d’une erreur de signe sur une des articulations, à voir. Un bon test est de chercher à ramener la patte dans sa position tendue vers l’avant.

Comme dit dans les lignes précédentes, la modélisation s’est arrêtée au MGI par manque de temps. Il aurait fallu cependant encore modéliser un mouvement de marche sur ce simulateur. Pour ce faire plusieurs méthodes sont disponibles sur le net, on trouve par exemple les marches dynamique ou encore en équilibre constant. Dans cette dernière catégorie on peut noter les marches inspirées du monde des insectes avec, entre autre, un mouvement utilisant deux pattes d’un côté et une de l’autre, le tout en symétrie axiale.

Ce type de mouvement constitués de deux blocs indépendants pourrait s’avérer être un moyen intéressant de faire se mouvoir le robot dans la mesure où les mouvements sont relativement faciles à distinguer et à mettre en place.

# Mouvement de la patte

Le code C fourni avec ce document permet comme indiqué plus haut de programmer les codeurs. On lui combine également le code permettant de faire bouger une patte en fonction des valeurs lui étant confiées. Sans rentrer dans les détails, les initialisations sont identiques, seul la fonction main est différente (à décommenter). Concernant le mouvement de la patte, on réalise les opérations suivantes :

* Initialisation de la lecture CAN à 1000 Kb/s.
* Passage en mode préopérationnel pour les quatre codeurs.
* Reset du zéro de l’incrémenteur dans la position courante de la patte (optionnel).
* Passage en mode opérationnel pour les quatre codeurs.
* Initialisation des PWM.
* Passage par une boucle infinie :
  + Obtention des valeurs d’angles.
  + Initialisation du mouvement vers les positions désirées en fonction de l’écart de celles-ci avec les positions actuelles correspondantes.

Par souci de non surcharge, on n’intègrera aucune commande manuelle par les potentiomètres.

On peut constater sur le prototype de carte un interrupteur allant du premier bornier de la carte jusqu’à l’emplacement n°6 du driver de moteur le plus haut dans l’empilement. Cet interrupteur permet de sécuriser la mise en mouvement de la patte. En effet, pour éviter tout déplacement brusque du mécanisme dès la mise sous tension du système, les drivers moteurs exigent un passage de cet interrupteur de la position ouvert à fermé.

# Liste des composants

|  |  |  |  |
| --- | --- | --- | --- |
| COMPOSANTS POUR **SIX** PATTES | | | |
| ***Composant*** | ***Quantité*** | ***Fournisseur*** | ***Ref.*** |
| Traco Power | 6 | RS | 396-5180 |
| RJ45 fem. | 12 | RS | 257-8779 |
| Connecteur Moteur | 24 | RS | 484-1782 |
| Bornier Alim. Entrée | 54 | RS | 710-0104 |
| RJ45 mal. | 12 | RS | 413-386 |
| Nucleo F429ZI | 6 | RS | 917-3775 |
|  |  |  |  |

# Conclusion :

L’objectif initial de ce projet était, dans un premier temps, la réalisation d’un système électronique permettant la mise en mouvement de chacune des pattes de l’hexapode R.HEX puis, dans un second temps, l’implantation d’un mouvement de marche sur le robot.

La première étape fut de définir un cahier des charges. Ce document, établi en collaboration avec les différentes personnes participant au projet, à permit d’identifier les nombreuses contraintes liées à l’architecture de l’hexapode. Afin de répondre à ce cahier des charges, un système générique applicable à l’ensemble des six pattes a été définit. De la même manière, un moyen de programmer les drivers des moteurs ainsi que l’ensemble des codeurs des pattes a été trouvé. On est maintenant dans la capacité d’alimenter une patte dans son ensemble, de la programmer totalement et de la faire se mouvoir avec précision. De plus, dans l’optique de créer un mouvement de marche, le robot a été simulé sous Matlab afin de juger de son évolution dans l’espace.

Tout en prenant en compte ces réalisations, il est à souligner que la marche n’a pu être expérimentée sur l’hexapode, seule une patte du robot a pu être testée en plus de celle de d’expérimentation. Mettre en mouvement le robot nécessite, comme on peut l’imaginé, d’avoir câblé la totalité de celui-ci or, il faut noter que dans l’état actuel des choses, le câblage de l’hexapode n’a été réalisé que de manière partielle.

Il reste à câbler sur le robot l’ensemble des deux premiers moteurs de chaque patte mais aussi à brancher les connectiques des quatre codeurs des cinq autres pattes de l’hexapode. Il ne faudra pas oublier également de souder et de placer tous les composants sur les six cartes permettant le fonctionnement des pattes du robot.

Le branchement prend, de manière générale, du temps, beaucoup de temps. Si une estimation devait être faite quant au temps restant de branchement et de programmation, le chiffre d’une semaine complète ne serait pas déraisonnable en prenant en compte une équipe de 3 à 4 personnes.

De plus, rien ne permet pour l’instant de piloter de façon centralisée les six pattes. Durant mon stage, une communication en réseau par le port Ethernet de la STM avait été envisagée et était en cours de réalisation. Cependant, n’étant pas terminé, je n’ai pas eu l’occasion de la mettre en pratique lors de mon stage.

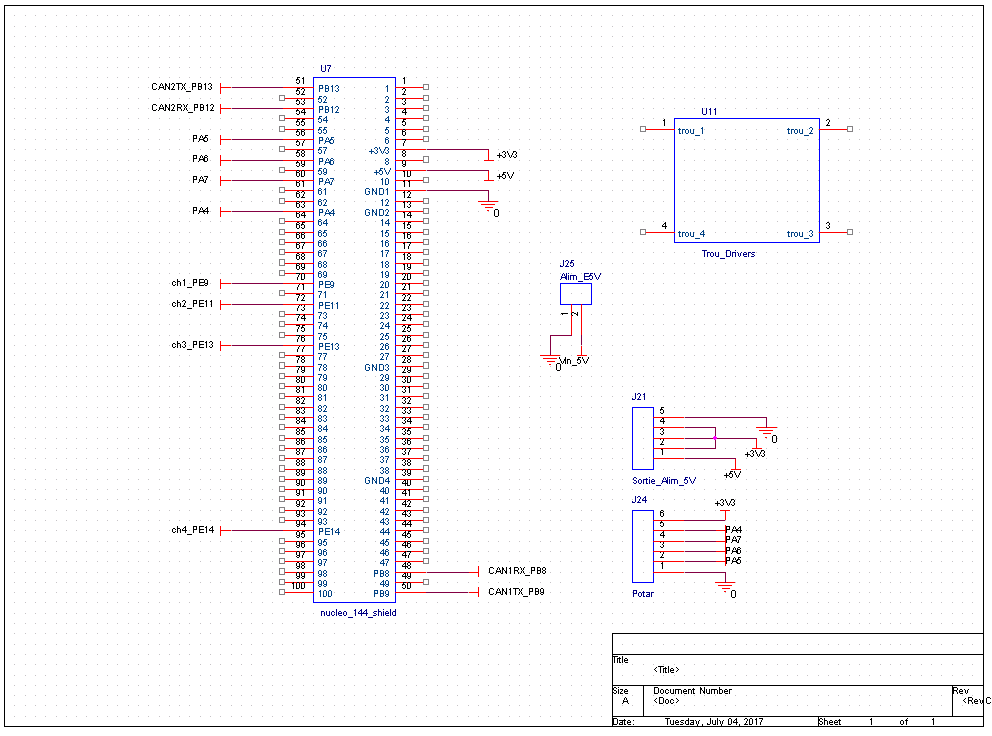
Par ailleurs, ce projet possède de nombreuses pistes d’amélioration. Premièrement, chaque patte utilise deux bus CAN. Aussi, il serait intéressant de limiter le nombre de bus dans la mesure où la technologie utilisée permet en théorie de joindre l’ensemble des capteurs du robot sur un seul bus CAN. Il serait également utile de diminuer la charge supportée par le robot en repensant l’architecture de la carte et des quatre drivers lui étant associés. Il a été mentionné lors de mon stage qu’une carte unique permettant de piloter l’ensemble des quatre moteurs serait en développement. Enfin, lorsque l’hexapode sera entièrement fonctionnel dans sa motricité, il sera judicieux de gérer correctement les données renvoyées par les codeurs ainsi que leurs priorités.

Sur le plan technique, ce projet m’a permis d’obtenir de nombreuses compétences importantes pour un futur ingénieur MEA. Premièrement, j’ai appris à réaliser des schémas électriques sous le logiciel Cadence, et surtout à concevoir des PCB. De plus, j’ai été amené à rechercher et sélectionner des composants adaptés aux contraintes de mon système. Enfin, ce stage m’a permis de me familiariser encore un peu plus à la STM32 et à l’environnement Eclipse.

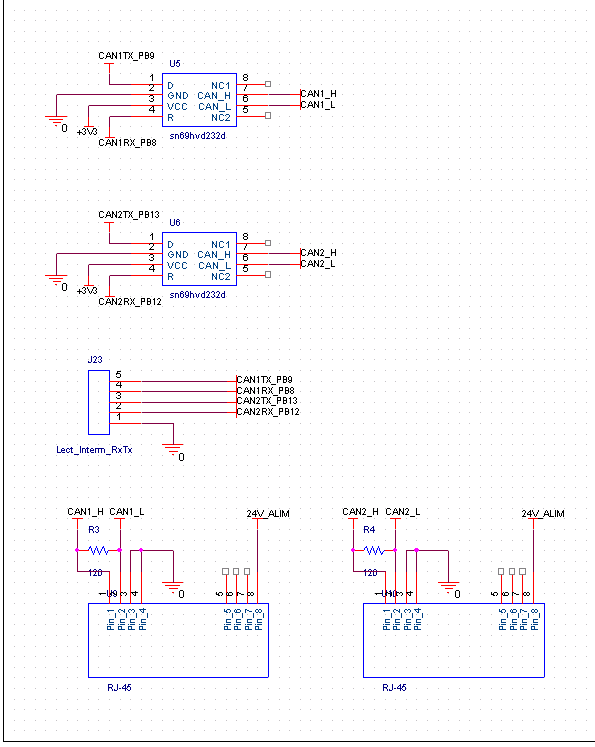
# Annexe :

Les images suivantes ne sont que des illustrations des schémas réalisé sous Cadence et donnés dans les dossiers associés à ce rapport.

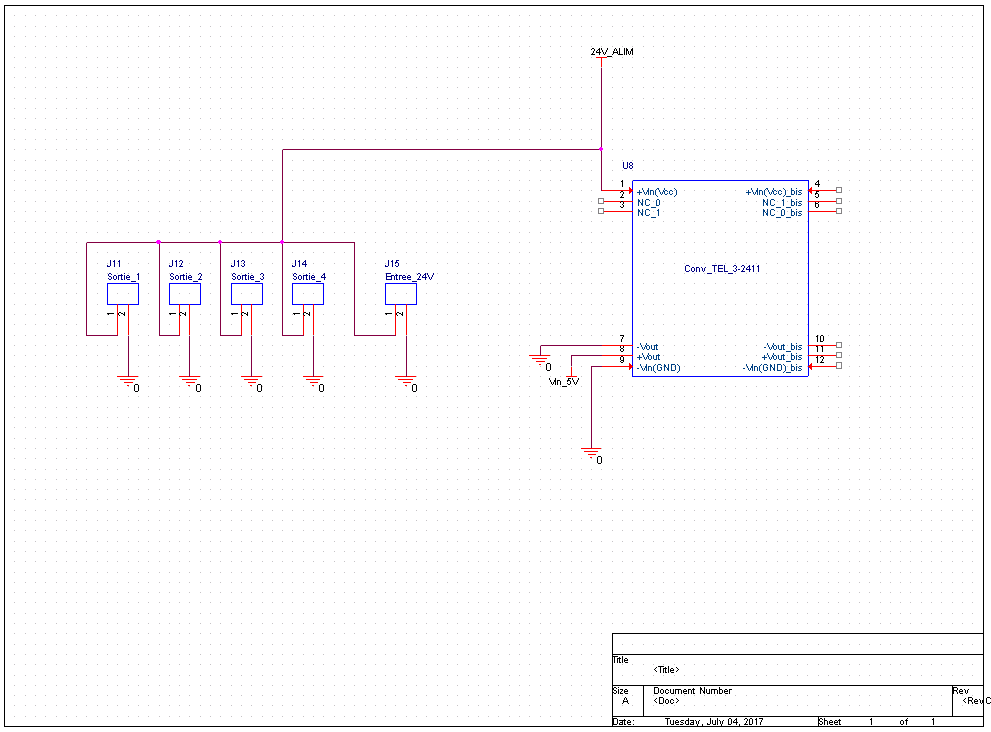
#### Schéma électrique du montage :



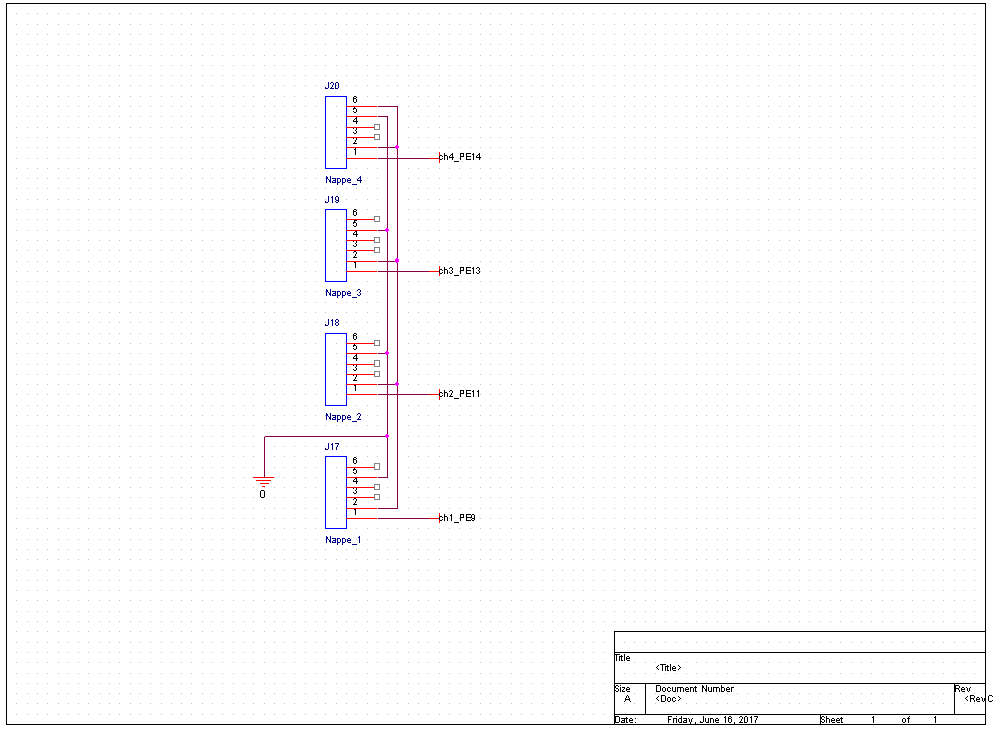
Page 1



Page 2

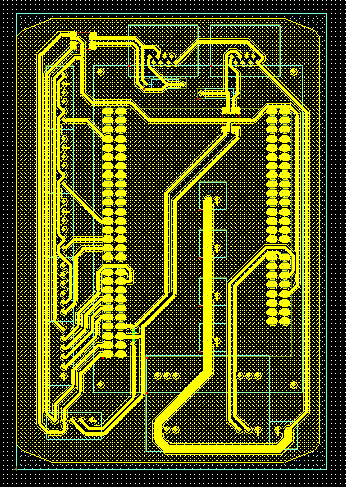


Page3

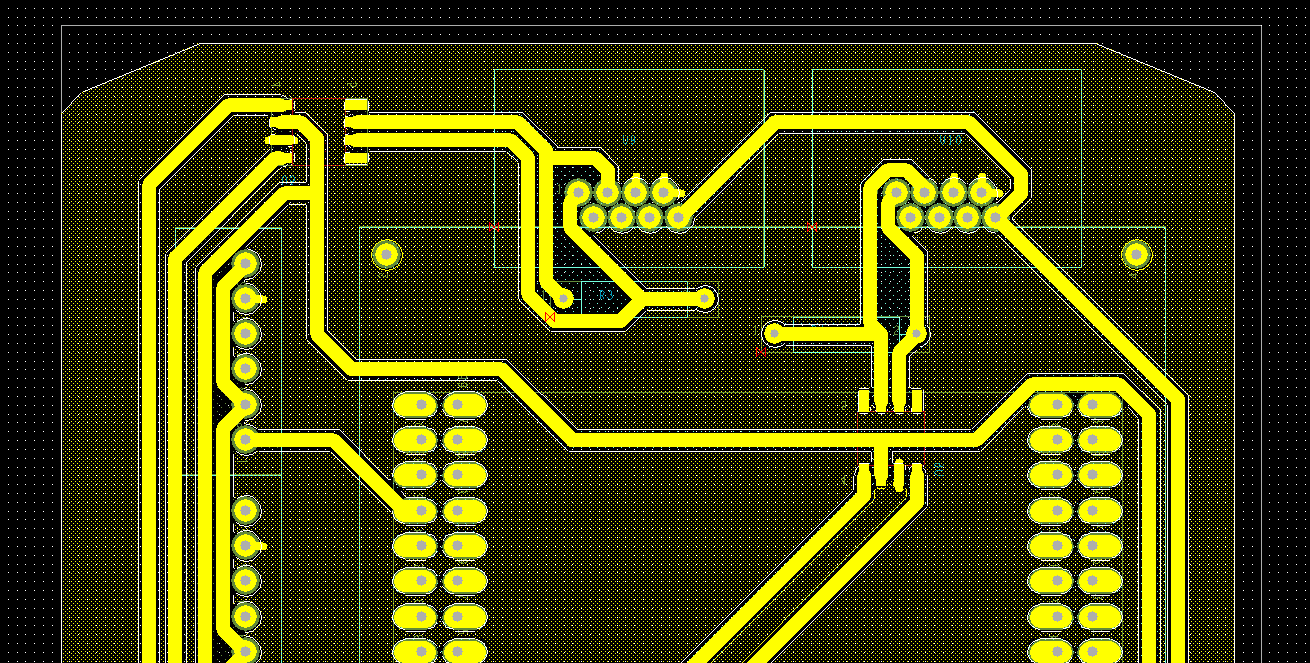


Page 4

#### Routage du prototype final :



Vue d'ensemble



Zoom sur la partie haute de la carte

#### Fiche technique du transformateur :

La fiche technique pourra être trouvée dans les fichiers joints.