

Projet universitaire du Master ASE

– RÉNOVATION D'UN ROBOT PUMA 500 –



Groupe d'étudiants :

WOUSSEN Anthony

NDAO Mohamadou Habib

ROZWAG Simon

ARCICASA Julien

GUO Kaiqi

BOUDEJELTHIA Abdelfettah

Responsable de projet :

EDEL Michel

REMERCIEMENTS

Notre groupe d'étudiants tient à remercier toutes les personnes qui ont activement contribué au bon déroulement de ce projet.

Nous tenons, avant tout, à adresser nos remerciements au responsable de projet, **Michel Edel** pour ses conseils avisés et son pragmatisme qui nous a permis d'orienter notre projet sur la bonne voie.

Nous tenons, aussi, à remercier les deux techniciens, **Kamel Ladrouz** et **Philippe Hennin**, pour leurs aides précieuses, leurs partages d'expériences et leurs actives contributions dans la concrétisation de ce projet. De surcroît, leurs convivialités et leurs volontariats sont à souligner.

Enfin, nous tenons à remercier tout les autres intervenants ayant contribués, même brièvement, dans le cadre de ce projet et qui nous ont permis d'assurer le bon déroulement des missions demandées.

TABLE DES MATIÈRES

REMERCIEMENTS	2
INTRODUCTION	4
PARTIE I : PRÉSENTATION GÉNÉRALE	5
a) Le robot PUMA 500	7
b) Les servomoteurs	8
c) Le système de commande	11
PARTIE II : CARTE DE CONTRÔLE	13
a) Le microcontrôleur PIC32MK1024MCF	15
b) Lecture des encodeurs	16
c) Lecture du potentiomètre de position	17
d) Pilotage des servomoteurs	19
e) Communication avec un ordinateur	20
PARTIE III : ALIMENTATION, CARTE DRIVER ET MANETTE DE COMMANDE	23
a) Réalisation de la partie alimentation	24
b) Les freins électro-mécaniques	27
c) Étude du couple des moteurs	31
d) Les cartes drivers	34
e) La conception de la manette de commande	41
PARTIE IV : ASSERVISSEMENT ET PILOTAGE GÉNÉRAL	53
a) Établissement des coordonnées des articulations du robot Puma	54
b) Asservissement	57
CONCLUSION	67
BIBLIOGRAPHIES	68
ANNEXES	71

INTRODUCTION

La tendance actuelle tend à favoriser la rénovation d'anciens automates, jugés jusque-là obsolètes, dans un but écologique. C'est en se basant sur cette idée que notre projet universitaire de 1ère année de Master ASE consiste en la rénovation d'un bras mécanique de type PUMA 500. Plus particulièrement, notre mission portera sur l'élaboration d'un nouveau système de commande usant de technologies modernes. Pour ce faire, notre groupe constitué de 6 étudiants, dirigé par un responsable de projet et secondé de deux techniciens, a pour mission d'étudier et d'élaborer les possibilités technologiques en vu d'un l'objectif final qui sera la conception du nouveau système de commande ainsi que la remise en fonction du robot PUMA 500 à des fins pédagogiques.

Avant toute chose, la première démarche a effectué sera d'étudier de manière générale le robot et son fonctionnement pour, ensuite, établir un cahier des charges comportant les différentes fonctions nécessaires et les possibles apports qui viendront enrichir le système de l'automate. Suivi d'une première identification globale des différentes parties qui constitueront le nouveau système de commande. Une fois cela achevée, une répartition en trois binôme pour accomplir les différentes tâches sera décidé et chaque binôme effectuera une recherche approfondie des solutions envisageables et compatibles avec les exigences du cahier des charges.

La prochaine étape du projet consistera à élaborer et concevoir les différents éléments constituant le nouveau système de commande. Le premier binôme aura pour mission d'établir un système capable de piloter avec précision et rapidité les servomoteurs qui constituent le bras mécanique. Le second binôme élaborera une commande manuelle et garantira une alimentation régulé pour l'automate. Finalement, le dernier binôme contrôlera l'asservissement du bras mécanique et établira un repère en trois dimensions dédié au robot pour garantir la fiabilité et la robustesse de ce dernier ainsi que la praticité de son pilotage.

Une fois ces tâches complétées indépendamment l'une de l'autre, il suffira ensuite de réunir le travail effectué par chaque binôme dans l'optique de concevoir un système de commande complété et modernisé permettant de piloter efficacement et intégralement le robot PUMA 500.

PARTIE I : PRÉSENTATION GÉNÉRALE

Rédigé par Anthony Woussen et Mohamadou Habib Ndao

Présentation



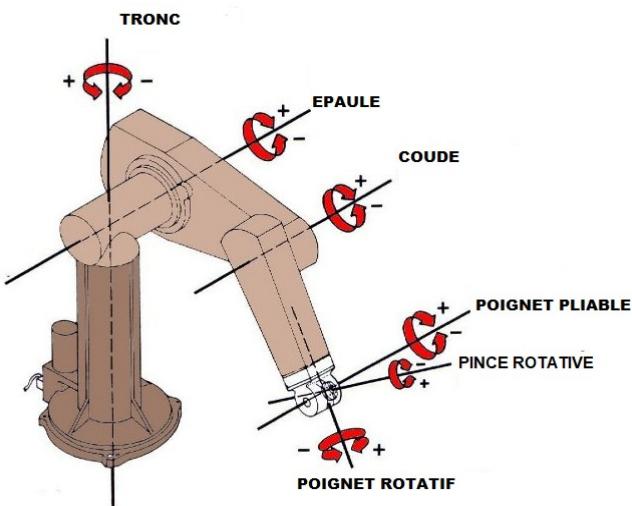
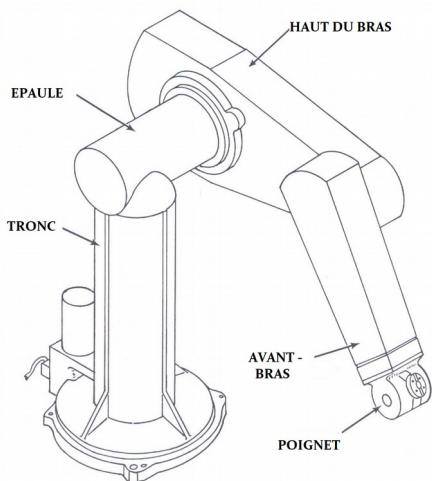
Durant cette présentation générale, nous présenterons les trois principaux éléments constituant le robot PUMA 500 que sont : le corps du bras mécanique et ses contraintes de déplacement, les servomoteurs dotés d'un encodeur et d'un potentiomètre de position, ainsi que le système de commande. À l'aide de cette étude, nous avons pu établir un cahier des charges mettant en évidence les différentes tâches et nécessités technologiques pour rénover le robot PUMA 500, présenté ci-dessous :

Cahier des charges

- Étudier le fonctionnement du robot PUMA 500
- Faire une recherche sur les technologies viables pour l'alimentation et le contrôle du robot
- Concevoir un nouveau système de commande modernisé
- Élaborer une manette de commande pour piloter directement l'automate via une communication de type CAN Bus
- Étudier l'asservissement du robot PUMA 500
- Établir un repère en trois dimensions pour le pilotage du robot
- Mettre en place les dispositifs de sécurité pour prévenir tous risques lors de la manipulation de l'automate (*pas abordé durant ce projet*)

a) Le robot PUMA 500

Le robot PUMA 500 (Programmable Universal Machine for Assembly), développé en 1983 par la société Unimation, est un bras robotique constitué de cinq parties mécaniques (voir figure 1).



PUMA 500

Le robot PUMA 500 est articulé grâce à six servomoteurs offrant une mobilité sur six axes rotatifs applicables sur les différentes parties de l'automate (voir figure 2). Chaque articulation possède un angle de rotation qu'il ne peut mécaniquement pas dépasser (voir Tableau 1).

Tableau 1 : Angles de rotation des articulations du robot PUMA 500

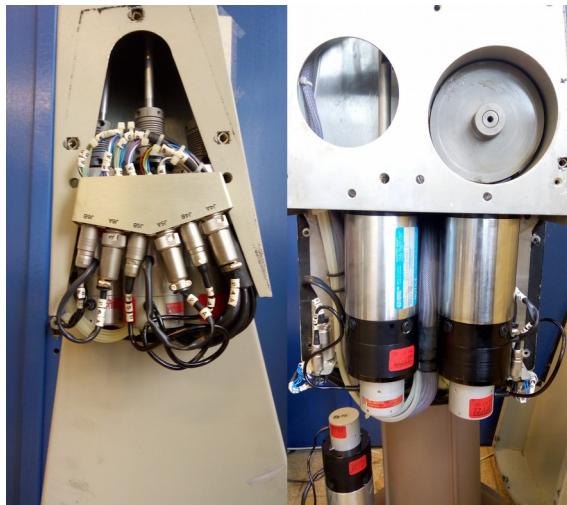
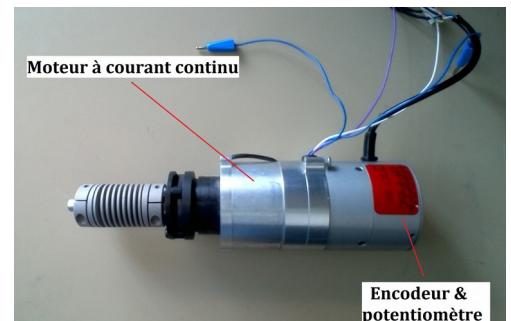
Articulations	Angles de rotation
TRONC	320°
ÉPAULE	250°
COUDE	270°
POIGNET ROTATIF	300°
POIGNET PLIABLE	200°
PINCE ROTATIVE	532°

b) Les servomoteurs

Les servomoteurs du robot PUMA 500 sont constitués de trois parties distinctes : le moteur à courant continu, l'encodeur et le potentiomètre de position.

- Moteur à courant continu

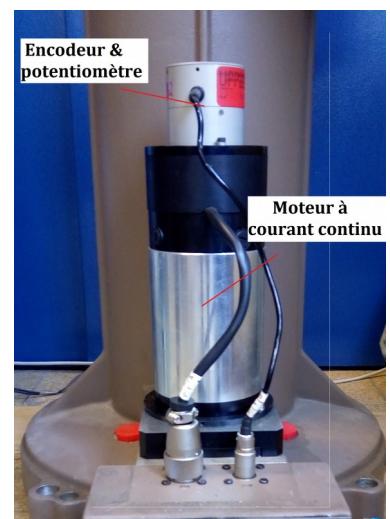
Les moteurs du robot PUMA 500 sont tous des moteurs à courant continu sans balais fonctionnant sous une tension nominal de 40 Volts et 10 Ampères maximal.



Il existe deux tailles pour les moteur à courant continu qui constituent le robot PUMA 500. Les plus petits moteurs sont utilisés pour piloter le poignet du bras mécanique.(voir Figure 3 – petit servomoteur) Les plus grands moteurs sont utilisés pour piloter le reste du robot. (voir Figure 5 – grand servomoteur)

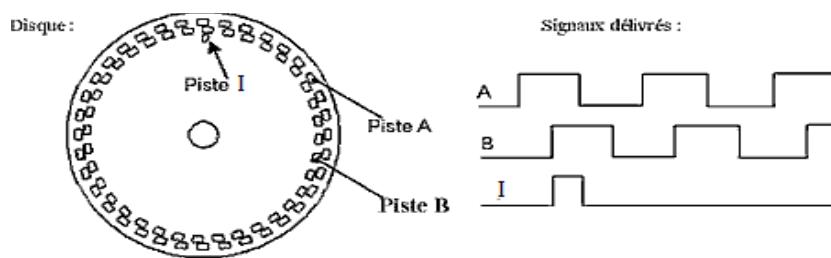
Les emplacements des moteurs se situent dans le haut du bras et l'avant-bras du robot. (voir Figure 4 – emplacement des moteurs)

Les moteurs sont tous équipés d'un frein électromécanique. Ces freins représentent une sécurité pour éviter tous risque lors d'une coupure de l'alimentation. Ainsi, les freins du robot sont activés par l'intermédiaire d'un relais interne, qui bloque la position des moteurs en l'absence de tension à ses bornes. Pour pouvoir débrayer totalement les freins du robot, il faut lui appliquer une tension d'au moins 24 Volts. Les freins sont contrôlés simultanément et ne peuvent pas être gérés individuellement car ils sont tous reliés entre eux.



- Encodeur (ou codeur incrémental)

Un encodeur est un dispositif permettant de déterminer avec précision la position angulaire d'un moteur. Pour ce faire, il dispose d'un disque tournant à la même vitesse et en même temps que le rotor du moteur et sur lequel il y a, à ses extrémités, une multitude d'encoches, tous espacées d'un même angle. (voir Figure 6). Des signaux sont alors délivrés grâce à une diode électroluminescente et un photo-transistor placés sur les côtés opposés du disque. (voir Figure 7)



Chaque impulsion sur les voies A et B correspond à un angle parcouru par le moteur. Plus il y a d'encoches sur le disque et plus la position du moteur est précise. Une impulsion sur le signal de l'index (I) indique le moment où le moteur a fait un tour complet sur lui-même.

Le sens de rotation d'un moteur se détermine en fonction du déphasage à 90° entre la voie A et la voie B de l'encodeur. (voir Figure 8)

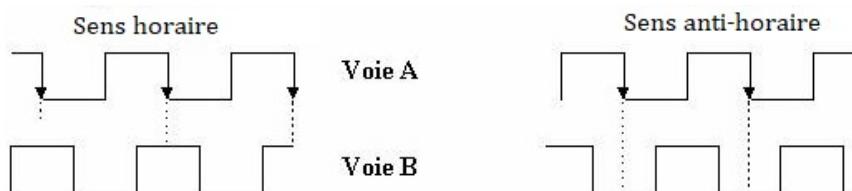
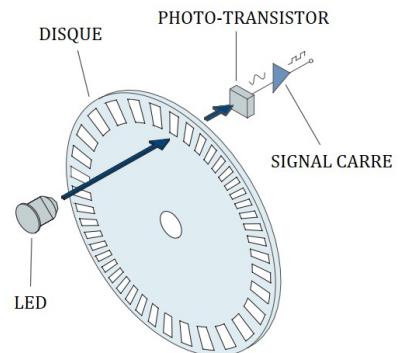


Figure 8 – Sens de rotation en fonction du déphasage entre la voie A et B

Les six encodeurs du robot PUMA 500 présentent chacun un nombre d'encoche total différent et un rapport réducteur vis-à-vis des parties du bras mécanique et des moteurs qui ont été référencés dans le tableau ci-dessous :

Tableau 2 : Rapport réducteur et nombre d'encoche total par encodeur

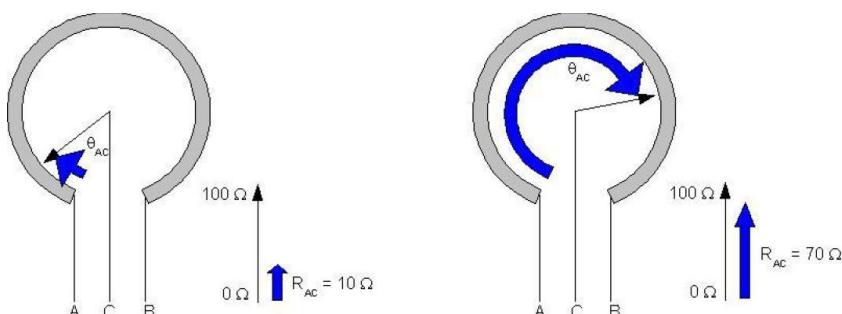
Articulations	Nombre d'encoche / tour	Réducteur
TRONC	1000	1:120
ÉPAULE	800	1:120
COUDE	1000	1:120
POIGNET ROTATIF	1000	1:120
POIGNET PLIABLE	500	1:120
PINCE ROTATIVE	1000	1:120

Le rôle du réducteur est d'augmenter la précision du pilotage du robot puisqu'il faut que le moteur effectue 120 tours complets pour que la partie mécanique en effectue un seul. Ainsi, le nombre d'encoche déterminant la position du moteur est multiplié par 120 pour déterminer la position du bras mécanique, augmentant donc drastiquement la précision de la lecture de sa position via les encodeurs.

Enfin, chaque encodeur du robot PUMA 500 doit être alimenté en 5V continu pour pouvoir fonctionner correctement.

- Potentiomètre de position

Le principe du potentiomètre de position trouve son intérêt dans le fait que la position du curseur est déterminée par la position angulaire du moteur. Ce qui permet, réciproquement, de déterminer la position angulaire du moteur en fonction de la tension mesurée au borne du potentiomètre.(voir Figure 9)



c) Le système de commande

Le système de commande a pour rôle de lire et d'interpréter les différents signaux générés par les encodeurs et les potentiomètres de position. Il gère aussi le pilotage des servomoteurs par la régulation énergétique appliquée à ces derniers.

L'asservissement du robot PUMA 500 et de ses servomoteurs est aussi assuré par le système de commande.

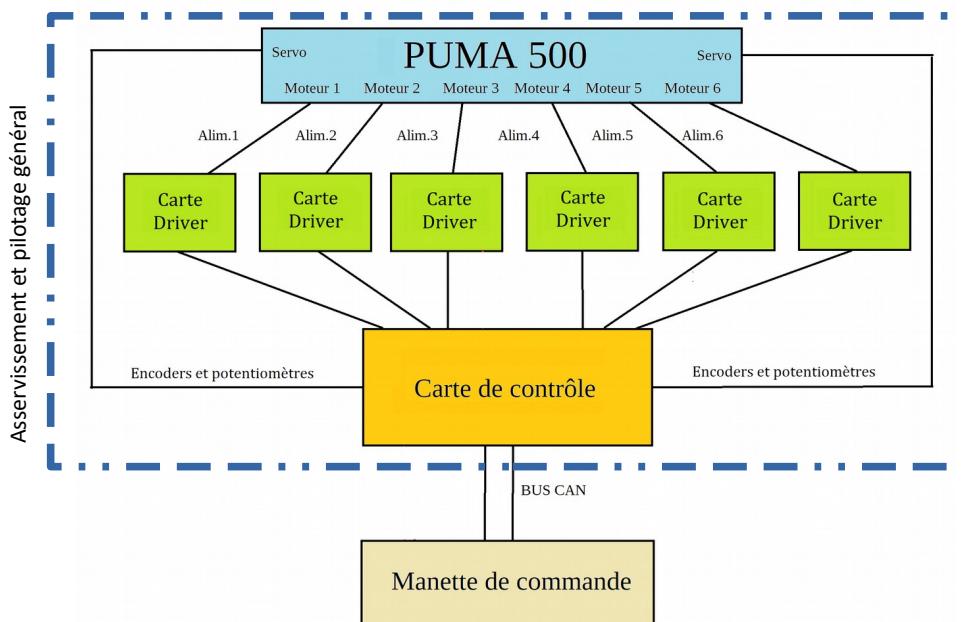


de commande du robot PUMA 500

De plus, ce dernier offre un moyen de piloter, de manière homogène et en temps réel, le robot PUMA 500 par le biais d'une manette de commande. Il est aussi possible d'enregistrer plusieurs positions spécifiques qui seront, par la suite et en fonction du mode sélectionné, reproduites automatiquement et de manière cyclique.

Dans le cadre de ce projet, l'ancienne console et la manette de commande du robot PUMA 500, (voir Figure 10) étant inutilisable, il nous incombe d'élaborer un nouveau système de commande avec des technologies modernes.

Nous avons alors établi les différentes parties qui composeront notre nouveau système de commande (voir Figure 11) et qui accompliront les différentes tâches évoquées précédemment.



système de commande

- Les cartes drivers :

Elles ont pour fonction d'opérer la régulation énergétique des servomoteurs et de faire la passerelle entre la partie électronique à faible puissance, et la partie électromécanique à forte puissance.

- La carte de contrôle :

Elle a pour fonction de lire et d'interpréter les informations fournies par les servomoteurs. Elle en déduit alors leur position respective et les pilote par le biais des cartes drivers , en considération des ordres émis par la manette de commande.

- La manette de commande :

Elle a pour fonction d'offrir un moyen de piloter le robot PUMA 500 manuellement. Cela se fera grâce à une communication de type CAN Bus entre la manette de commande et la carte de contrôle.

- Asservissement et pilotage général :

C'est une étude théorique dont la tâche sera d'asservir le robot pour assurer sa précision, sa fiabilité et sa robustesse. Elle devra aussi établir un repère en trois dimensions dans lequel l'automate se déplacera en tout unité.

Conclusion

Comme notre étude du robot PUMA 500 l'a montré, ce dernier est constitué de cinq parties mécaniques entraînées par six servomoteurs. Eux-mêmes constitués respectivement d'un encodeur et d'un potentiomètre de position, qui permettent de déterminer avec précision la position angulaire de chaque moteur et ainsi pouvoir piloter avec efficacité le déplacement de l'automate. Nous avons eu l'occasion d'étudier les différentes caractéristiques des six servomoteurs et du robot en général. Suite à cela, nous avons établi un cahier des charges listant les différentes tâches et possibilités pour contrôler le robot. Pour finir, nous avons réfléchi à une composition globale de ce que sera le nouveau système de commande du robot PUMA 500.

Ayant identifié les différentes parties qui constituent le nouveau système de commande, et dans le but d'optimiser notre temps de production, une répartition des tâches en trois groupes a été effectué et est présentée ci-dessous :

Noms des binômes	Parties attribuées
Anthony Woussen & Mohamadou Habib Ndao	Carte de contrôle
Simon Rozwag & Julien Arcicasa	Alimentation et manette de commande
Guo Kaiqi & Boudejelthia Abdelfettah	Asservissement et pilotage général

PARTIE II :

CARTE DE CONTRÔLE

Rédigé par Anthony Woussen et Mohamadou Habib Ndao

Présentation

La carte de contrôle représente le point centrale du système de commande et son rôle est multiple. Elle doit être capable de lire et d'interpréter les différentes informations fournies par les servomoteurs et ainsi pouvoir déterminer leurs positions angulaires exactes. Elle doit aussi diriger le bras mécanique efficacement et avec précision en pilotant chaque moteur par le biais des cartes drivers. Recevant des instructions venant de la manette de commande, la carte de contrôle doit être capable de proposer plusieurs modes de fonctionnement tel qu'un pilotage direct ou un pilotage automatique par répétition cyclique de positions préalablement enregistrées.

Au vu du nombre de tâches dont la carte de contrôle doit effectuer simultanément, notre idée de départ fut de produire six cartes de contrôle qui géreraient respectivement leur propre servomoteur et qui communiqueraient entre elles grâce à une communication de type CAN Bus.

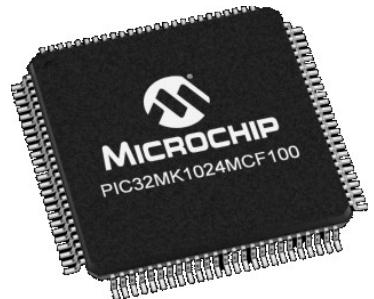
Cependant, dans un souci de réduire au maximum la production de cartes électroniques et l'utilisation des divers composants, nous avons tenté de trouver une solution technologique qui nous offrirait la possibilité de contrôler l'intégralité du robot PUMA 500 en ne produisant qu'une seule carte de contrôle et en n'utilisant qu'un seul microcontrôleur.

Nous avons alors entamé nos recherches d'un microcontrôleur avec pour critère de sélection d'être capable de gérer les six servomoteurs à lui seul. Il va s'en dire que les cartes électroniques universelles telles que Arduino ou Raspberry pi sont à exclure car elles ne sont pas spécialisées dans le domaine de gestion des servomoteurs et sont donc très limitées pour cette application. Idem pour les micro-contrôleurs de type 16F, 18F ou 24F.

Finalement, notre choix s'est porté sur le microcontrôleur pic32mk1024mcf (voir figure 12) .

a) Le microcontrôleur PIC32MK1024MCF

Le microcontrôleur pic32mk1024mcf est une technologie récemment élaborée par la société Microchip. Le sigle MCF (Motor Control Family) nous indique que c'est un microcontrôleur spécialisé dans le contrôle de plusieurs moteurs et plus précisément, il est capable de gérer jusqu'à six servomoteurs. Ce qui est en adéquation avec l'utilisation dont nous voulons en faire. (voir Annexe caractéristique pic32mk1024mcf)



Les différents outils qui ont été utilisé pour programmer le microcontrôleur pic32mk1024mcf dans le cadre de ce projet sont les suivants :

- Le logiciel de développement **MPLAB X IDE** associé au compilateur XC32, fournis gratuitement par la société Microchip sur leur site internet, permet d'écrire le programme du pic32mk1024mcf.



- Le programmateur **Pickit 3** permet de transférer le programme écrit avec le logiciel MPLAB X IDE dans le microcontrôleur.

- L'outil de développement, **MPLAB Harmony**, est un firmware (ou logiciel interne) gratuit et dédié à la programmation des microcontrôleurs PIC32. Son interface intuitive et facile d'utilisation permet de simplifier la configuration des microcontrôleurs PIC32 (voir annexe MPLAB Harmony) en générant automatiquement un programme selon les options choisies. Cependant, MPLAB Harmony se limite à la configuration et il ne tient qu'à l'utilisateur de faire le programme principal.



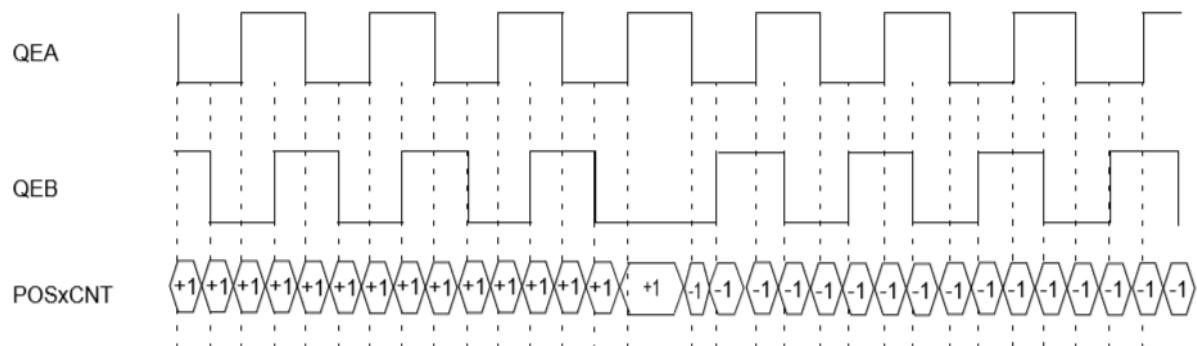
b) Lecture des encodeurs

L'une des missions principales de la carte de contrôle est la lecture des encodeurs. Grâce à cela, nous pourrons alors être en mesure de connaître avec précision la position angulaire de chaque servomoteur, et donc indirectement celle du robot PUMA 500.

Comme nous l'avons vu durant la présentation des servomoteurs (voir PRÉSENTATION GÉNÉRALE – les encodeurs), la position angulaire des moteurs est déterminée par le nombre d'impulsion comptabilisé sur les voies A et B. Il existe un module spécifique appelé QEI (Quadrature Encoder Interface) disponible chez certains microcontrôleurs, dont le pic32mk1024mcf, qui est dédié à la lecture de ces encodeurs.

Voir Annexe – code : QEI initialisation

Le principe se résume simplement à un registre nommé POSxCNT auquel le microcontrôleur va venir incrémenter ou décrémenter sa valeur en fonction des états dans lesquels se trouvent les signaux sur la voie A et B. (voir Figure 16)

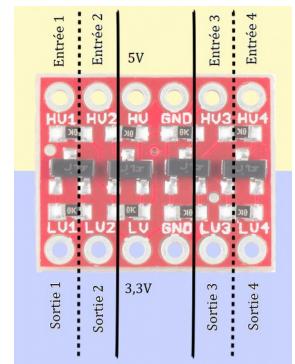


Lorsque nous voudrons enregistrer une position spécifique du robot PUMA 500, il nous suffira alors de stocker les valeurs des registres POSxCNT de chaque servomoteur dans des variables dédiées. Ces mêmes variables serviront ensuite de points de repère durant le pilotage automatique de l'automate.

Voir Annexe – code : enregistrement de position

Toutefois, Il est nécessaire d'initialiser la valeur du registre POSxCNT après chaque mise sous tension du microcontrôleur dans une position prédéfinie du robot PUMA 500 que nous nommerons la position initiale. (voir Lecture du potentiomètre de position)

A noter que les encodeurs du robot PUMA 500 doivent être alimenté impérativement en 5V et délivrent des signaux de 5V. Or, certaines entrées du microcontrôleur pic32mk1024mcf ne tolère que des tensions allant jusqu'à 3,3V. L'utilisation d'un convertisseur 5V vers 3,3V est donc nécessaire. (voir Figure 17)



c) Lecture du potentiomètre de position

Comme cela a été stipulé précédemment, il nous est nécessaire d'initialiser la valeur des registres POSxCNT lorsque le robot PUMA 500 est dans sa position initiale. Si cette condition n'est pas remplie, alors nous n'aurions aucun repère fixe sur lequel nous baser pour déterminer avec précision la position des servomoteurs via leur encodeur.

Mais alors que la lecture des encodeurs nécessite une initialisation au démarrage, il n'en ai pas de même pour la lecture du potentiomètre de position. En effet, comme cela a été présenté précédemment (voir PRÉSENTATION GÉNÉRALE – potentiomètre de position), la valeur de la tension au borne du potentiomètre de position dépend directement de la position angulaire de son moteur et ne nécessite donc pas d'initialisation.

De ce fait, et malgré que la détermination de la position d'un moteur en passant par son potentiomètre de position soit bien moins précise qu'avec son encodeur, cela sera tout de même suffisant pour pouvoir déterminer la position initiale du robot PUMA 500.

Pour ce faire, il nous faut appliquer une conversion analogique/numérique ou ADC (Analog/Digital conversion) pour avoir la possibilité de lire la tension aux bornes du potentiomètre de position par le biais du microcontrôleur pic32mk1024mcf.

Voir Annexe – code : ADC initialisation

Le principe d'une conversion analogique/numérique est le suivant :

À partir d'une tension de référence, AVDD (dans notre cas, ce sera 3,3V) et le nombre de bits de résolution, ou encore le nombre de bits sur lequel se fera la conversion, (dans notre cas, l'ADC du pic32mk1024mcf sera sur 12 bits), le micro-contrôleur va alors calculer le rapport entre ces deux valeurs que l'on nomme le quantum.

Le quantum correspond à la valeur du LSB (Least Significant Bit), ou encore bit de poids faible.

$$quantum = \frac{AVDD}{2^{\text{résolution}}} = \frac{3,3}{2^{12}} = 0,8 \text{ mV}$$

Le microcontrôleur va ensuite comparer la valeur de la tension appliquée sur son entrée analogique en fonction d'une plage de tension déterminée proportionnellement au quantum. (voir Figure 18)

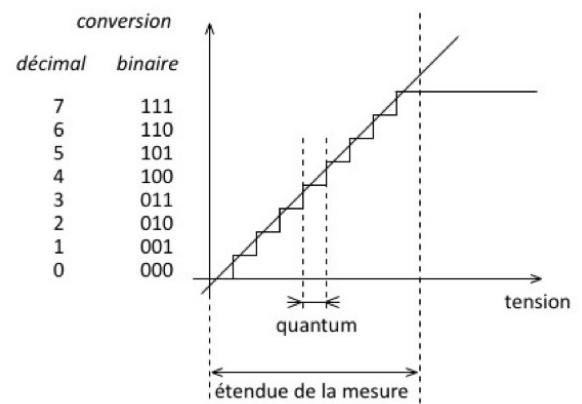
Une fois, la comparaison terminée et la valeur binaire estimée proportionnellement au quantum, cette dernière sera alors stockée dans le registre dédié ADCDATAx.

Il nous est alors très facile de retrouver la tension analogique mesurée en multipliant la valeur du registre ADCDATAx par la valeur du quantum.

Grâce à cela, nous pouvons déterminer la tension appliquée au bord du potentiomètre de position et ainsi en déduire la position initiale du robot PUMA 500.

Dans le cadre de ce projet, nous souhaitons que la position initiale de l'automate corresponde à une position verticale, ainsi la tension appliquée aux potentiomètres de position doit être de 1,65V soit ADCDATAx = 4125.

Dans un souci de précision, la position initiale sera aussi déterminée par une impulsion sur l'index des encodeurs indiquant une résolution complète des servomoteurs.



Voir Annexe – code : Position initiale

d) Pilotage des servomoteurs

Une fois, la lecture des encodeurs et des potentiomètres de positions effectuée, la carte de contrôle a aussi pour mission de piloter les servomoteurs.

Pour ce faire, nous utiliserons les cartes drivers qui vous seront présentées plus en détail ultérieurement. (voir CARTE DRIVER)

Ce qu'il faut savoir, à ce stade de l'étude, c'est que du point de vue de la carte de contrôle, il est possible de piloter les servomoteurs individuellement avec trois signaux en sorties.

Deux sorties sont alors utilisées comme signaux directionnels et une sortie est utilisée comme un signal PWM (Pulse Width Modulation), ou encore MLI (Modulation de Largeur d'Impulsion).

Le principe du PWM est de faire varier à haute fréquence le rapport cyclique d'un signal, c'est à dire le temps à l'état haut par rapport à la période, pour réguler « analogiquement » la valeur moyenne de ce même signal. Dans le cas d'un moteur, cela est utilisé pour faire varier sa vitesse avec un signal Tout Ou Rien, soit un signal où uniquement deux états sont possibles : le « 1 » logique et le « 0 » logique. (voir Figure 19)

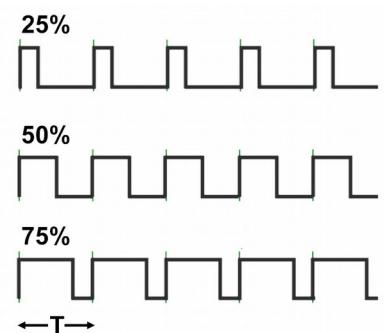


Tableau 3 : État des signaux de pilotage des servomoteurs

IN1	IN2	Rapport cyclique PWM	Moteur
0	0	X	Arrêt
1	1	X	Arrêt conflictuel
1	0	1	Rotation horaire à pleine vitesse
0	1	1	Rotation anti-horaire à pleine vitesse
1	0	$0 < X < 1$	Rotation horaire à vitesse X %
0	1	$0 < X < 1$	Rotation anti-horaire à vitesse X %

Voir Annexe – code : PWM initialisation

Voir Annexe – code : Pilotage manuel

e) Communication avec un ordinateur

Pour assurer le bon déroulement du programme ainsi que pour avoir un retour visuel des différentes informations lues et envoyées par le microcontrôleur pic32mk1024mcf, une communication avec un ordinateur a été ajouté.

Pour cela, nous avons utilisé une communication de type UART (Universal Asynchrone Reception Transmission).

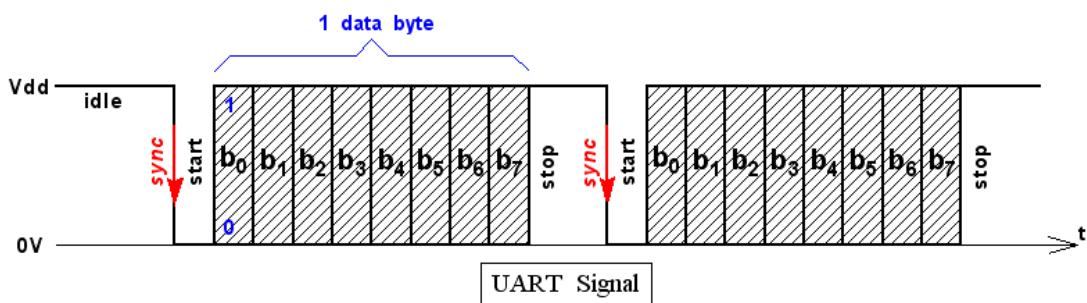


Cette communication se fait à l'aide d'un câble Prolific (voir Figure 20) qui relie la carte de contrôle avec le port USB d'un ordinateur.

USB- to-Serial

Le principe d'une communication de type UART correspond à l'envoie sur la broche TX et à la réception sur la broche RX d'un trame de donnée (voir Figure 21). Étant donné la nature asynchrone de la communication, il est impératif d'ajouter un bit de start et un bit de stop entre les octets de données signalant respectivement le moment où la transmission/réception commence et le moment où elle se termine.

De plus, il faut, au préalable, imposer la même configuration concernant le nombre de bit transmit par seconde, appelé le BaudRate, chez le microcontrôleur et l'ordinateur.



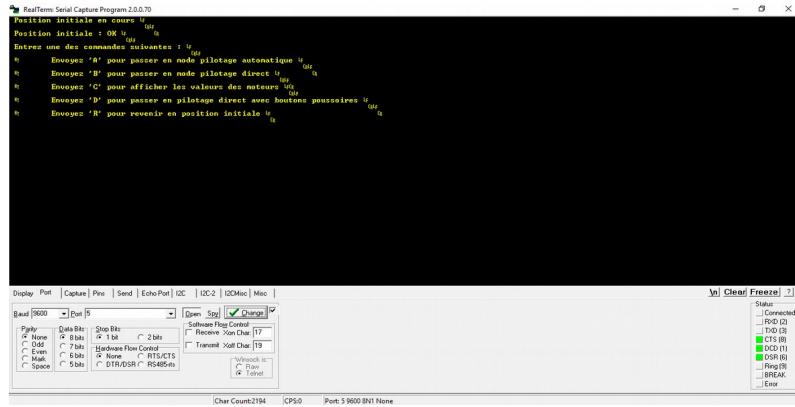
D'autres options telle que la possibilité d'utiliser des bits de parités pour assurer la bonne communication sans perte de donnée entre les différents interlocuteurs peuvent être ajoutées mais nous avons jugé que cela ne sera pas nécessaire dans le cadre de ce projet.

Voir Annexe – code : UART initialisation

Voir Annexe – code : UART Reception

Une fois la communication entre la carte de contrôle et l'ordinateur établie, nous utilisons le logiciel **Realterm** (voir Figure 22) nous permettant de visualiser les différentes données reçues.

Figure 22 – Interface de l'affichage Realterm



Dans le cadre de ce projet, nous avons mis en place un menu de choix des commandes permettant diverses actions et sélectionnable selon le caractère envoyé par l'ordinateur.

- Envoyer 'A' pour passer en mode pilotage automatique

Cette commande permet d'ordonner une répétition cyclique des différentes positions des servomoteurs pré-enregistrées. Il vous sera alors demandé le nombre de cycle voulu allant de 0 à 255. Notre choix arbitraire sur le nombre de position enregistrable s'est limité à 3 positions mais peut facilement être augmenté selon les besoins.

- Envoyer 'B' pour passer en mode pilotage direct

Cette commande permet de positionner chaque servomoteur en envoyant l'angle désiré via l'ordinateur. Ce mode propose aussi d'enregistrer les positions angulaires souhaitées qui seront réutilisées par la suite en pilotage automatique.

- Envoyer 'C' pour afficher les valeurs des moteurs

Cette commande permet d'afficher les différentes valeurs concernant les moteurs.

- Envoyer 'D' pour passer en mode pilotage direct avec boutons poussoirs

Cette commande permet de piloter directement les moteurs grâce à des boutons poussoirs. Il vous sera alors demandé de sélectionner le moteur que vous voulez piloter.

- Envoyer 'R' pour revenir en position initiale

Cette commande permet de revenir en position initiale et de réinitialiser les différentes positions enregistrées pour le pilotage automatique.

Voir Annexe – code : Menu des commandes

Résultats & conclusion intermédiaire

Une fois l'étude de la carte de contrôle achevée, l'étape suivante fût celle de la production d'une carte électronique. Cependant, le microcontrôleur pic32mk1024mcf étant un composant extrêmement petit et doté d'une multitudes de broches, nous avons rencontré des problèmes techniques concernant ce dernier. En effet, le brasage de ce type de composant nécessite un matériel spécial, telle une machine à brasage, dont nous n'en disposions pas à l'université. L'option de faire appel à une société de sous-traitance pour produire la carte de contrôle nous soumettant à un délai de production incertain, nous nous sommes rabattu vers l'utilisation d'une carte fille dotée d'un pic32mk1024mcf (voir Figure 23).



Ce choix d'utiliser la carte fille avec un pic32mk1024mcf nous a permis de passer outre le problème du brasage. Toutefois, un nouvel inconvénient s'est présenté à nous. En effet, cette carte fille, commercialisée par Microchip, n'est pas initialement conçue pour être utilisée sur des projets particuliers comme le notre mais prévue en complément d'une carte de développement, elle aussi, commercialisée par Microchip. Ces cartes de développement sont conçues pour permettre aux professionnels comme aux néophytes de se familiariser avec les technologies proposées par la société Microchip. Ainsi, certains broches du pic32mk1024mcf de la carte fille sont matériellement et exclusivement dédiées à des fonctions spécifiques. Ces contraintes nous empêchant alors d'utiliser le microcontrôleur pic32mk1024mcf à son plein potentiel. Malgré cela, nous avons réussi à produire deux carte de contrôle usant de la carte fille et étant capable de piloter jusqu'à trois servomoteurs chacune. (voir Annexe – Carte de contrôle avec carte fille)

Enfin, dans l'optique de la réalisation future d'une carte de contrôle capable de piloter les six servomoteurs à elle seule en n'usant que d'un seul pic32mk1024mcf, comme cela en était l'intention de départ, nous avons réalisé un typon pour circuit imprimé grâce à logiciel Kicad. (voir Annexe – Typon : Carte de contrôle six servomoteurs)

Pour conclure, nous avons eu l'occasion d'étudier et de réaliser les différentes méthodes permettant au microcontrôleur pic32mk1024mcf de gérer le pilotage et la lecture des différentes informations provenant de plusieurs servomoteurs simultanément. Nous avons eu l'occasion de nous familiariser avec le logiciel Mplab X IDE ainsi que son firmware, Mplab Harmony. Nous avons réussi à relever le challenge de travailler avec un microcontrôleur relativement récent. De même, nous avons réussi à produire une carte de contrôle viable malgré certains compromis. Cependant, suite aux différents tests effectués, nous avons constaté que le système manque de stabilité dû au fait que les servomoteurs ne sont pas asservis. C'est ce qui sera abordé dans la troisième partie ASSERVISSEMENT ET PILOTAGE GÉNÉRAL.

PARTIE III :

ALIMENTATION, CARTE

DRIVER ET MANETTE DE

COMMANDÉ

Rédigé par Simon Rozwag et Julien Arcicasa

a) Réalisation de la partie alimentation

1. introduction de la partie

Dans cette partie, nous allons expliquer comment nous avons procédé pour réaliser la partie alimentation. L'objectif de cette partie est de parvenir à alimenter l'ensemble du robot ainsi que tout ce dont nous avons besoin pour le contrôler.

2. Choix des alimentations

Pour contrôler l'ensemble du robot, différentes puissances sont à fournir afin d'alimenter chaque composant (moteurs, freins, encodeurs, drivers et carte de commande). Tous ces éléments demandent des niveaux de tension et de courant différents. Il nous a fallu, tout au long du projet, étudier en détail les composants que nous allions utiliser afin d'en déduire l'apport nécessaire en énergie que nous devions leur apporter. A titre d'exemple, nous ne connaissons pas les drivers que nous allions utiliser, la tension d'alimentation du boîtier de commande ou encore celle de la manette de contrôle.

3. Étude de l'utilisation d'alimentations d'ordinateurs

Nous voulions tout d'abord réaliser l'alimentation à l'aide d'anciennes alimentations d'ordinateurs fixes qui présentent de nombreux avantages.

Elles fournissent une tension 12V, cela veut donc dire qu'il est possible de débloquer les freins avec deux alimentations en série. Trois alimentations en série permettent de réaliser une tension 36V qui est proche de la tension nominale demandée par chaque moteur. Les alimentations des ordinateurs fournissent également du 5V ou 3,3V qui pourraient alimenter une partie commande composées de microcontrôleurs. Cette récupération permettraient d'économiser de l'argent et de recycler du matériel.

Notre nouvel objectif était donc de voir si ces alimentations seraient suffisantes pour avoir un couple et une vitesse relativement importante en connaissant les caractéristiques des moteurs. Cependant, nous avons rencontré de nombreuses difficultés au cours de cette étape :

-> Les déchetteries ont des règles strictes quant à la récupération du matériel déposé dans leurs espaces.

-> Le second problème était de trouver des alimentations similaires. La mise en série d'alimentations non identiques peut provoquer des problèmes de surtension si elles n'ont pas des caractéristiques similaires en courant.

Suite à ces recherches, nous avons trouvé trois alimentations ayant pour caractéristiques: 12 volts 19 ampères. Nous avons essayé de brancher une seule alimentation sur un petit "moteur test" identique à ceux préexistants sur le robot. Nous avons appris que les alimentations d'ordinateur fixe nécessitent d'être en court-circuit au niveau de certaines pattes afin d'obtenir les 12V en sortie. Ce court-circuit est un avantage car il permet de pourvoir créer facilement un bouton de démarrage commun aux alimentations.

Une fois cette alimentation reliée au moteur, nous avons constaté qu'il tournait correctement. Cependant, la vitesse et le couple délivrées par ce dernier n'étaient pas suffisants. Nous avons donc essayé de mettre en série les alimentations afin d'augmenter la tension à ces bornes. Nous avons alors branché la masse d'une alimentation au 12 V d'une autre. La sécurité d'une des alimentations se déclenchaient et l'empêchait de sortir la tension globale de 24 V désirée. En désaccordant la terre de chaque boîtier d'alimentation, il est possible de les mettre en série. Comme l'opération était dangereuse, nous avons donc décidé de chercher une autre méthode.

Nous avons par la suite essayé de déclencher les freins du moteur qui nécessitaient du 24V via les sorties 12V et -12V d'une même alimentation. Le courant maximum pouvant passer dans la sortie -12V de notre alimentation n'était que de 0,8 ampère. Il n'était pas assez important, nous ne sommes pas parvenus à désactiver les freins du plus gros moteur présent sur le PUMA 500.

Nous nous sommes rendus compte qu'il nous fallait beaucoup d'alimentations pour satisfaire nos contraintes et qu'il s'agirait d'une alimentation encombrante



Par la rencontre de ces nombreux problèmes incontournables notamment en matière de sécurité, nous avons décidé d'acheter des alimentations disponibles sur le marché.

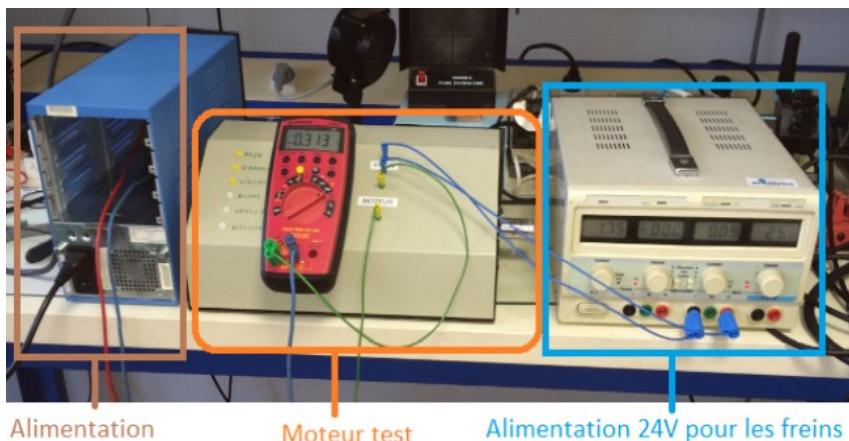
4. Choix de l'alimentation

Les recherches nous ont amenés à acheter trois alimentations industrielles AC/DC de 350 Watts fournissant en sortie 36 Volts et 9,7 Ampères chacune. Nous avons choisi ces alimentations car leur tension de sortie est proche de la tension nominale des moteurs et l'ampérage de sortie est proche du courant maximal que peut absorber un moteur. Afin d'éviter tout accident avec le robot, nous n'en utiliserons qu'une.



Les bornes de l'alimentation sont facilement accessibles. C'est pourquoi nous avons dû placer cette alimentation dans un ancien boîtier métallique qui était auparavant utilisé pour une alimentation du PUMA 500. Rappelons que cette alimentation dépasse les valeurs critiques pouvant causer de graves dommages en cas de contact direct avec un être humain. Afin de respecter les r

ègles de sécurité, nous avons pris soin de relier le boîtier à la terre ainsi que de sécuriser l'accès aux bornes de cette alimentation à l'aide de colle et de caches plastiques. De plus, nous avons placé des bornes femelles afin de s'y repiquer en toute sécurité avec des câble eux même normalisés.



Une fois la réalisation terminée, nous avons testé cette alimentation à l'aide d'un moteur test provenant d'un robot PUMA.

b) Les freins électro-mécaniques

1. Introduction de la partie

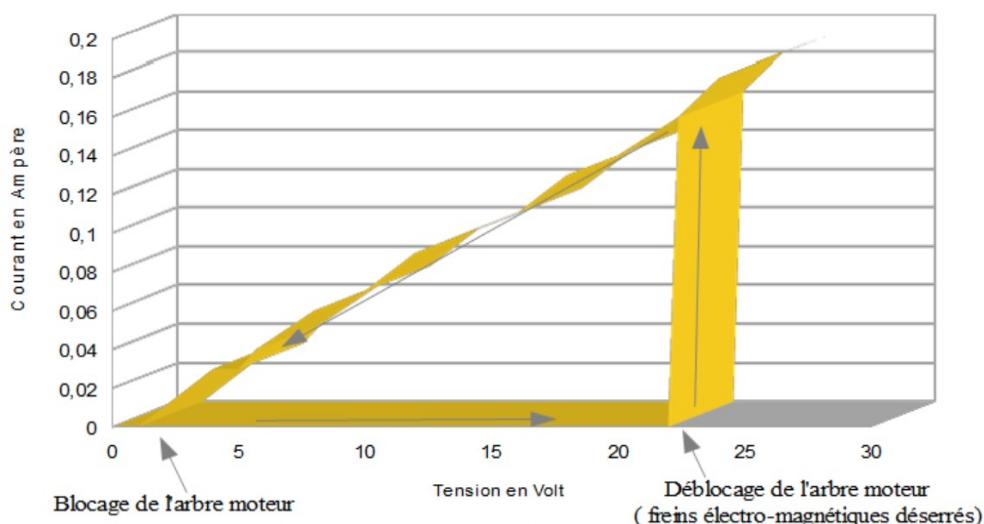
En début de projet, nous avions pour but de contrôler le robot en déplaçant légèrement chaque moteur l'un après l'autre afin que le mouvement soit le plus fluide possible. Cette solution envisageait donc de contrôler chaque moteur un par un en débloquant puis en bloquant chaque frein aussi rapidement que possible.

2. Étude du système de freins électro-mécaniques

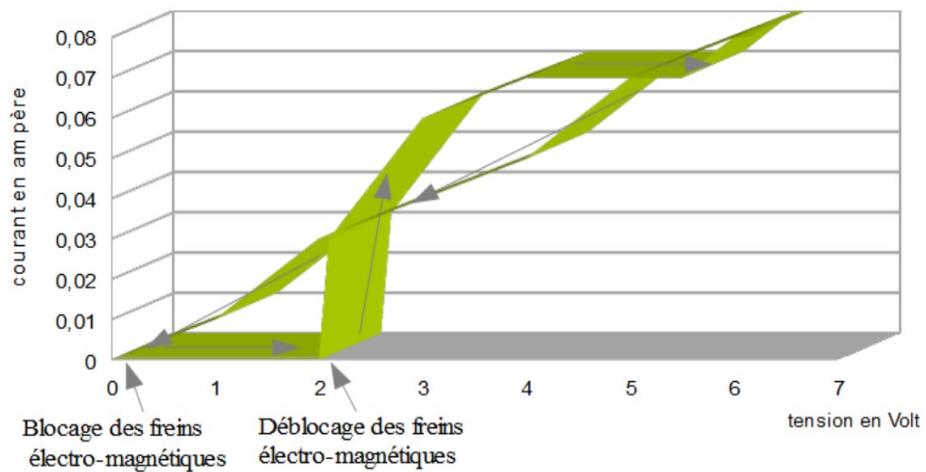
Sur le robot PUMA 500, chaque moteur possède un frein électro-mécanique. Ces freins électro-mécaniques sont des freins qui, par l'action d'un courant électrique circulant dans une bobine, produisent un champ magnétique qui permet le libre déplacement des moteurs présents sur le bras articulé.

Nous avons décidé d'étudier les caractéristiques des freins et des moteurs afin de pouvoir choisir le matériel adéquat pour les contrôler. Pour cela, nous avons alimenté les freins via une alimentation de tension stabilisée réglable. Nous avons relevé le courant en fonction de la tension imposée. Nous avons obtenu les deux courbes suivantes :

Caractéristiques tension/courant des freins électro-magnétiques des gros moteurs du PUMA 500



Caractéristiques tension/courant des petits freins électro-magnétiques



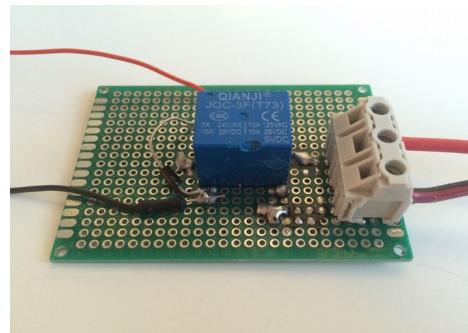
Nous tenons à signaler que la courbe précédente n'est pas représentative des résultats attendus et obtenus précédemment. Le petit frein testé ci-dessus était au niveau d'un petit moteur test qui, comme les techniciens nous l'avaient indiqué, n'était pas en très bon état. Par conséquent, le cycle d'hystérésis est quelque peu désordonné.

Nous sommes forcés de constater que les freins suivent un cycle d'hystérésis. Cela est dû à la disparition de l'entrefer lorsque les freins sont activés. Il convient de savoir que les freins électro-mécaniques fonctionnent d'une façon similaire à des relais. Avant la fermeture du circuit ferromagnétique, il y a la présence d'un entrefer. Ce dernier demande une force magnétique (et donc un courant électrique) plus importante. Une fois que les freins sont désactivés, le circuit magnétique est modifié. L'entrefer n'est plus présent dans le circuit ferromagnétique. Il faut alors beaucoup moins d'énergie magnétique et donc électrique pour que le circuit s'ouvre de nouveau. Cela explique le cycle d'hystérésis. Comme nous pouvons le voir également, les freins ne demandent que très peu de courant.

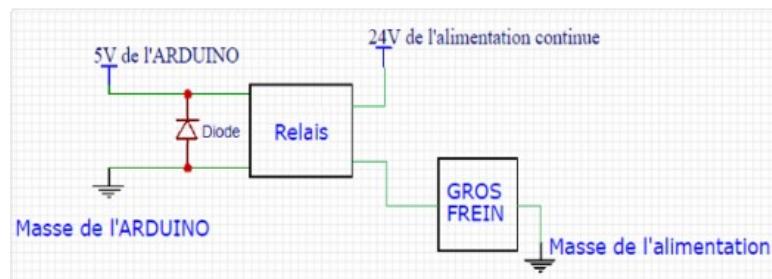
3. Test de contrôle des freins via un module relais

En début de projet, nous pensions pouvoir contrôler rapidement chaque moteur un par un en désactivant le frein concerné pendant la période d'alimentation des moteurs. Pour cela, nous avons étudié ce qui était préférable pour contrôler chaque frein. La première idée était de les contrôler via des relais. Nous avons fait ce choix pour sa facilité et sa rapidité de mise en œuvre. De plus, il existe des cartes composées de plusieurs relais sur le marché. Nous devons donc connaître à quelle fréquence maximale nous pouvons les commuter (activer et désactiver les freins) sans que l'inertie nous en empêche. C'est pourquoi nous avons réalisé les tests sur un gros frein où la masse et l'inertie sont plus importantes (le cas le plus défavorable). Nous avons donc réalisé la carte nécessaire pour effectuer les tests.

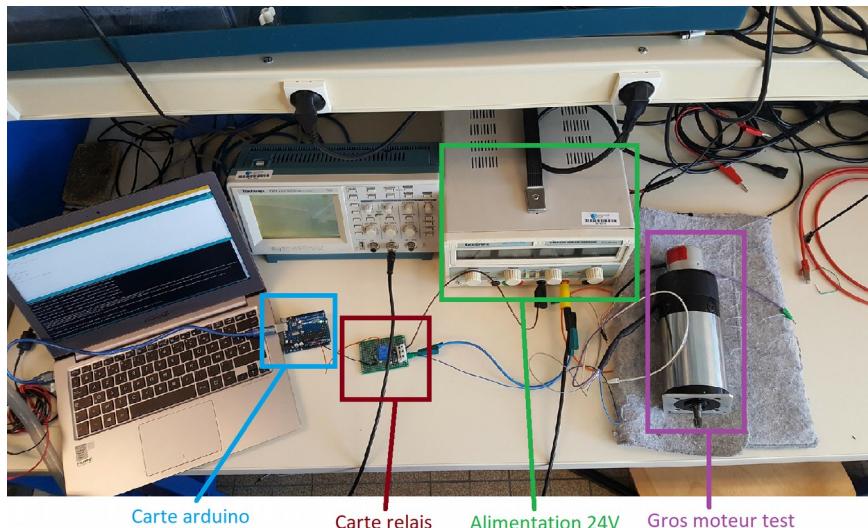
Voici la carte réalisée :



En voici un schéma électrique explicatif:



Nous avons placé une diode au niveau de l'Arduino afin d'éviter d'avoir un retour de courant trop important sur ce dernier à cause de la partie inductive présente dans le relais.



Nous avons donc réalisé le programme ci-dessous permettant d'envoyer des impulsions de tension au niveau de la broche A0 de l'Arduino où est branché le relais. Cela permet d'activer puis de désactiver le frein rapidement. Nous avons augmenté la fréquence jusqu'à sa limite qui est celle du programme ci-dessous :

```

void setup()
{
    pinMode(A0, OUTPUT);
}

void loop()
{
    digitalWrite(A0, HIGH);
    delay(38);
    digitalWrite(A0, LOW);
    delay(38);
}

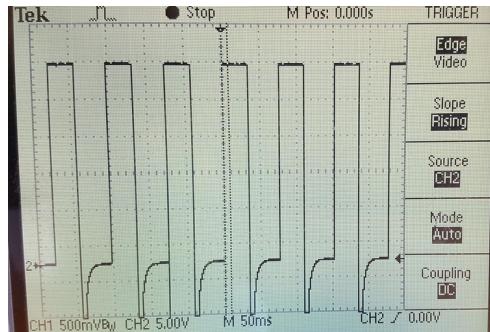
```

Nous avons déduit de ce programme que nous pouvions commuter les freins au maximum à :

$$f = \frac{1}{T} = \frac{1}{(0,038\text{s} \times 2)} = 13\text{ Hz}$$

Au-delà de cette limite, les fréquences sont trop élevées pour que le frein puisse commuter. Nous avons pu constater son bon fonctionnement par le bruit provoqué par les commutations et la liberté de rotation de l'axe moteur par intermittence.

Voici ce que nous obtenons à l'oscilloscope :

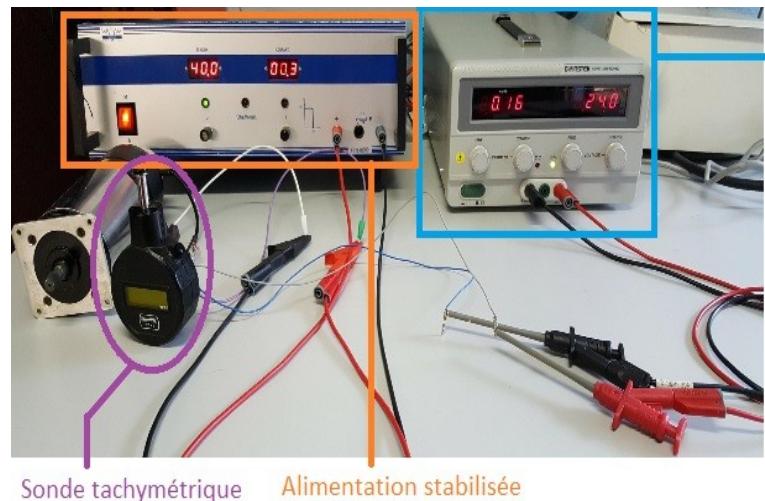


La fréquence est plutôt faible. De plus, les 6 relais feraient beaucoup de bruits sonores de commutations.

Notre seconde option pour commander les freins était de réaliser le même système que celui des relais mais avec des transistors. Ces derniers ont des vitesses de commutation plus importantes que celles des relais. De plus, ils ne font pas de bruit sonore de commutation.

Comme nous pouvons le voir ci-dessus, les gros moteurs tournent moins vite que les petits pour une même tension.

Les moteurs sont de faibles consommateurs de courant lorsqu'ils fonctionnent à vide (de l'ordre de 0,3A). Cependant, il convient de savoir qu'ils sont conçus pour tirer au maximum 10 A. Ce courant devient important dès lors que le moteur entraîne une charge.



Alimentation
des freins

Sonde tachymétrique Alimentation stabilisée

Voici, ci-dessus, deux photos prises pendant les essais.

On peut y distinguer la sonde tachymétrique tenue en main, une alimentation de tension stabilisée pour maintenir les freins électro-magnétiques désactivés ainsi qu'une seconde pour l'alimentation du moteur.

c) Étude du couple des moteur

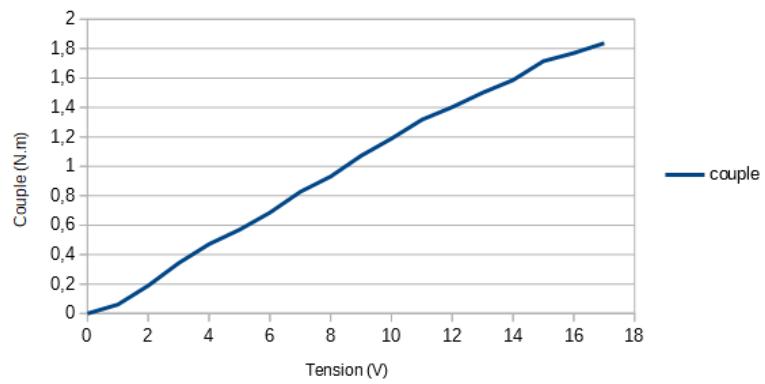
Afin de vérifier le bon fonctionnement des moteurs, nous avons pris une alimentation stabilisée pouvant fournir 52 V et 30 A. Le moteur est placé sous la barre de fixation du banc de manipulation. Une corde est fixée sur l'axe de rotation du moteur d'un côté et à un pèse-bagage de l'autre. Elle s'enroule autour d'un cercle de rayon 1,25 centimètres positionné sur l'axe de rotation du moteur. L'alimentation en bas à droite nous permet de débloquer les freins du moteur.

Une fois le moteur libéré, nous réglons l'alimentation du moteur pour ne pas dépasser 10 A et nous augmentons la tension petit à petit afin de relever le courant ainsi que la force exercée sur le pèse-bagages. Le couple nous sera donné par la formule : $C = F \times d$, où C est le couple, F la force qu'exerce le moteur sur le pèse-bagages et d le rayon du cercle autour duquel la corde s'enroule.

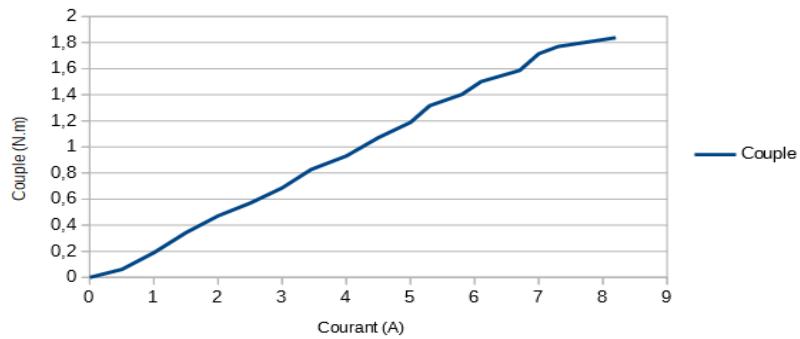
Voici un tableau regroupant les données relevées ainsi que le couple calculé :

Tension (Volts)	Courant (Ampères)	Poids (kilogrammes)	Couple (Newton-mètres)
0	0	0	0
1	0,5	0,5	0,06125
2	1	1,55	0,189875
3	1,5	2,8	0,343
4	2	3,85	0,471625
5	2,5	4,65	0,569625
6	3	5,6	0,686
7	3,45	6,75	0,826875
8	4	7,6	0,931
9	4,5	8,75	1,071875
10	5	9,7	1,18825
11	5,3	10,75	1,316875
12	5,8	11,45	1,402625
13	6,1	12,25	1,500625
14	6,7	12,95	1,586375
15	7	14	1,715
16	7,3	14,45	1,770125
17	8,2	15	1,8375

Tracé du couple en fonction de la Tension



Tracé du couple en fonction du courant



Montage ayant permis d'obtenir les résultats précédents :

Les mesures ne vont pas au-delà de 17 V en tension afin d'éviter toute surchauffe des moteurs ainsi qu'une éventuelle détérioration. Si nous poussons légèrement la tension, nous pouvons entendre un léger grincement du moteur et nous observons une perte de couple à tension constante. Ces petits problèmes viennent sans doute du fait que le robot date de 1987 et que les moteurs soient un peu usés. C'est d'ailleurs pour cette raison que nous avons décidé de refaire ces mesures afin d'obtenir des résultats représentatifs de nos moteurs.



Illustration 10: Montage de la mesure du couple.

d) Les cartes drivers

1. Introduction de la partie

Dans cette partie, nous allons expliquer comment nous avons procédé pour réaliser le boîtier driver. Ce boîtier permet le contrôle de l'ensemble des moteurs via un microcontrôleur.

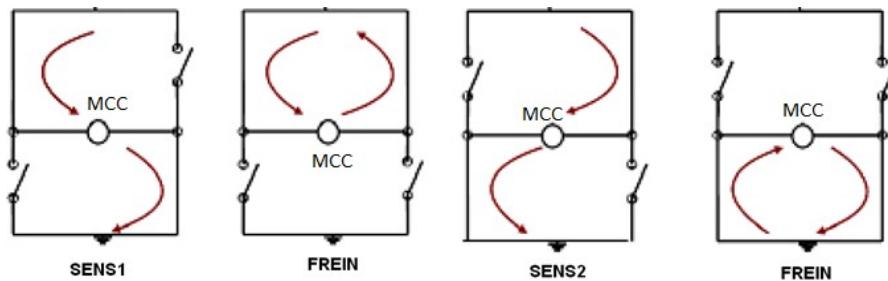
2. Principe de fonctionnement d'un driver

Le pont en H :

Le pont dit "en H "est une structure électronique ou électrique qui consiste à mettre en forme de H quatre interrupteurs qui sont commandés à l'ouverture ainsi qu'à la fermeture. En fonction de la commande de ces derniers, le moteur à courant continu placé au centre de ce pont a la possibilité de tourner dans les deux sens mais a également la possibilité de freiner en court-circuitant les bornes de ce dernier.

Ce principe du pont en H est utilisé par exemple dans le fonctionnement des cabines d'ascenseur pour récupérer une partie de l'énergie lors du freinage. Ce système est dit « récupératif ».

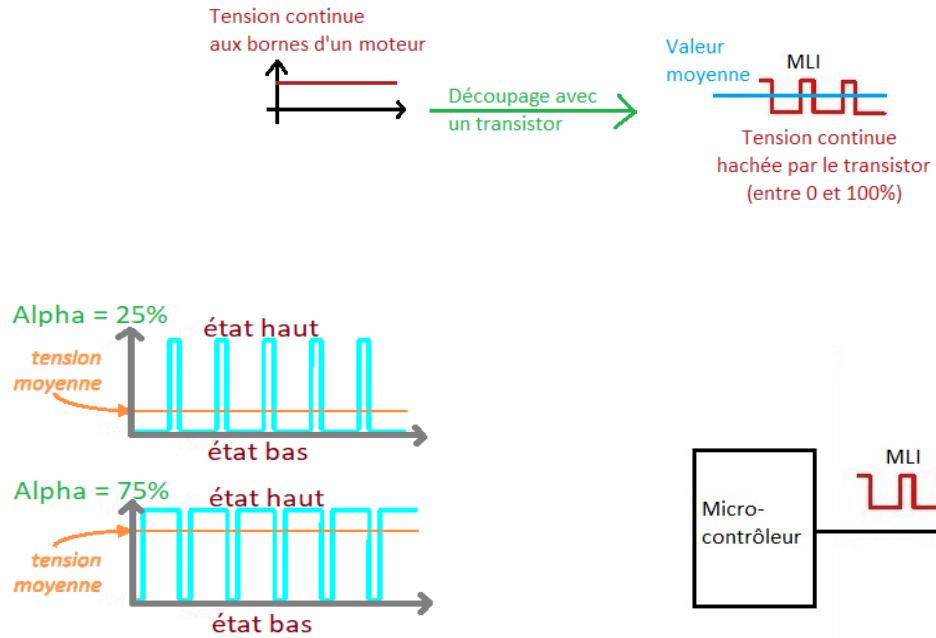
Voici toutes les combinaisons possibles pour commander le moteur à courant continu placé au centre:



3. Commande des interrupteurs:

Les interrupteurs utilisés dans les ponts en H sont principalement des relais ou des transistors selon le type d'utilisation. Ces derniers sont commandés par une partie commande qui fournit à chacun d'entre eux un signal MLI (Modulation en Largeur d'Impulsion) ou le terme anglo-saxon PWM (Pulse Width Modulation).

La MLI est une technique qui permet d'obtenir à partir d'une tension continue constante une tension modulable. Cette tension modulable possède un rapport cyclique Alpha compris entre 0%, signal constamment à l'état bas, et 100% signal constamment à l'état haut. La tension modulable créée est hachée si rapidement qu'elle est en réalité lue comme une tension continue qui est la valeur moyenne de cette dernière.



4. Choix des drivers et validation de leur bon fonctionnement :

Le choix des drivers que nous avons décidé d'utiliser s'est fait en fonction de l'alimentation. Nous avons donc cherché des drivers supportant une tension de 36 V et un courant de 9,7 A. De plus, le choix du sens de rotation et le PWM doivent pouvoir être commandés avec des tensions de l'ordre de 3,3 V ou 5 V car il s'agit des niveaux logiques utilisés par le microcontrôleur central. Nous avons trouvé et choisi ce driver:



Voici les caractéristiques:

Alimentation: 12 à 36 Vcc

Sorties: 1 x 12 A (1 x 15 A avec refroidisseur)

Entrées/sorties:

- niveau haut: 2 à 5,5 Vcc

- niveau bas: 0 à 0,8 Vcc

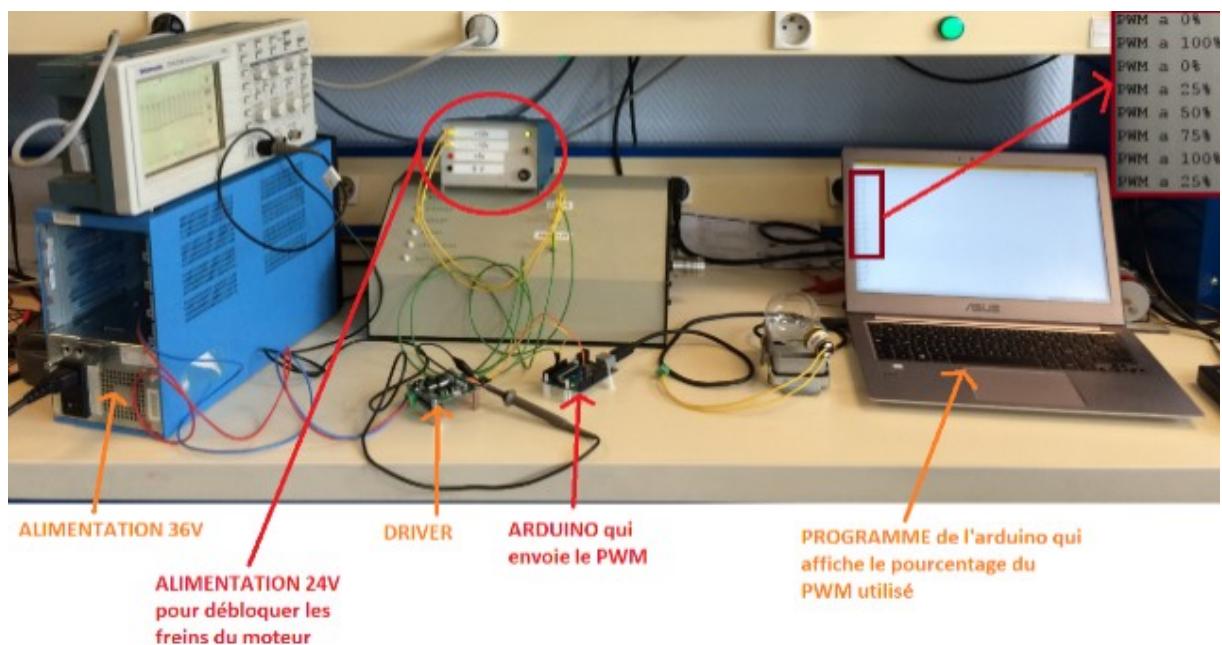
Signal de commande: PWM

Température de service: -25 à +85 °C

Poids: 42 gr

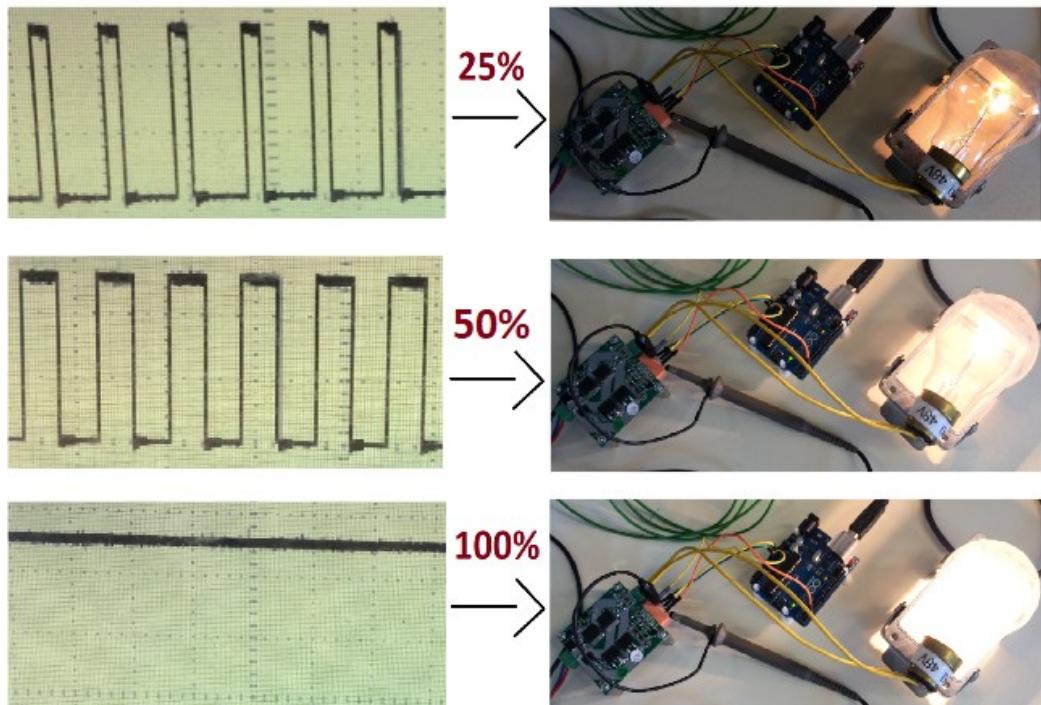
Dimensions: 55 x 55 x 20 mm

En plus de respecter les conditions précédentes, il dispose d'une sortie 5 V d'une grande utilité. En effet, les microcontrôleurs et les divers éléments permettant la commande du robot comme les encodeurs demandent d'être alimentés en 5 V ou en une tension légèrement inférieure. Une fois ces drivers reçus, nous avons décidé de les tester pour s'assurer de leur bon fonctionnement. De plus, nous avons décidé de vérifier que la température des transistors ne devenait pas trop élevée. Pour ce faire, nous avons généré un signal PWM à l'aide d'une carte Arduino (programme en annexe). Nous avons réalisé les tests avec une charge peu inductive afin d'éviter d'abîmer le matériel. Nous avons donc relié la sortie du driver à une ampoule 48 V.

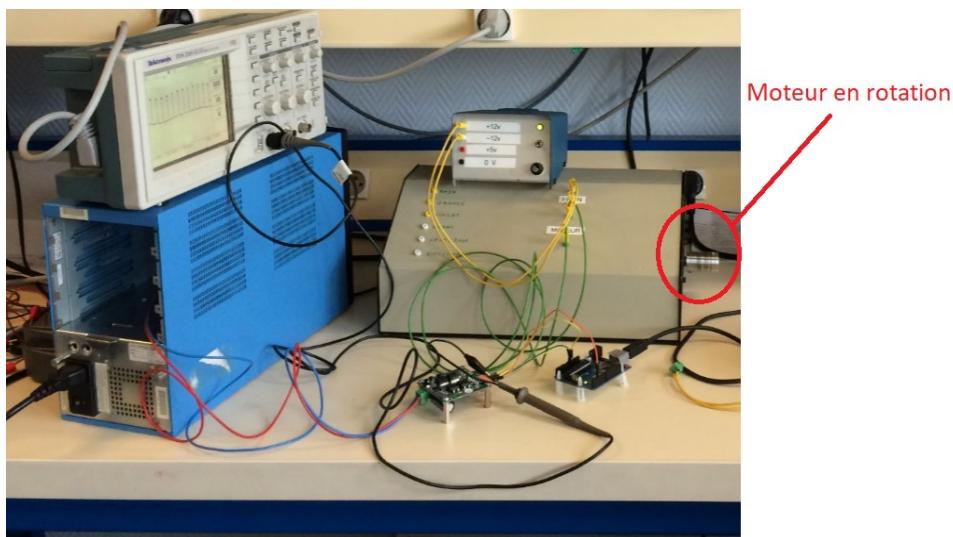


Comme il est possible de le voir sur les photos ci-dessous, plus le PWM est élevé, plus la valeur moyenne de la tension est importante et donc, plus la luminosité générée par la lampe est importante.

PWM EN POURCENTAGE:

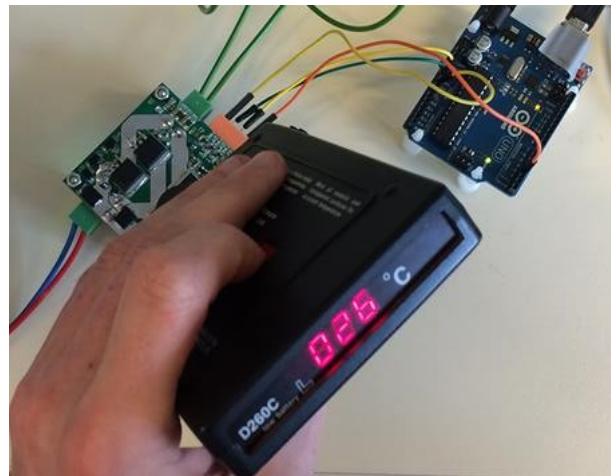


Une fois ces tests réussis, nous avons décidé d'essayer cette méthode un moteur du PUMA 500. Sur le même principe que la lampe, plus le PWM est élevé, plus la vitesse du moteur est importante.



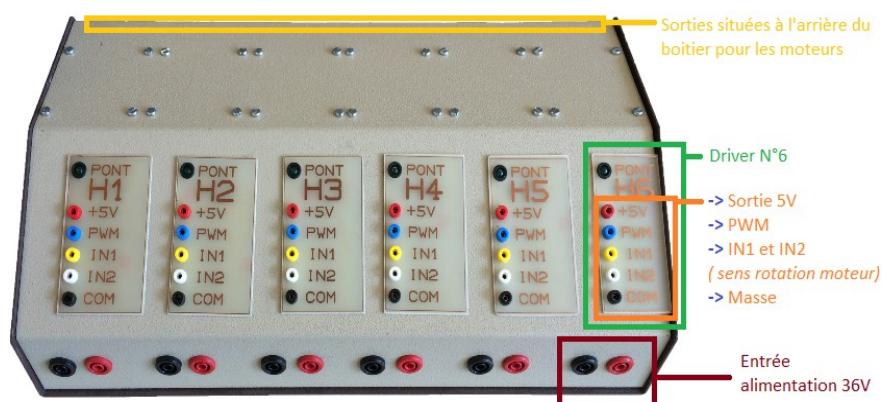
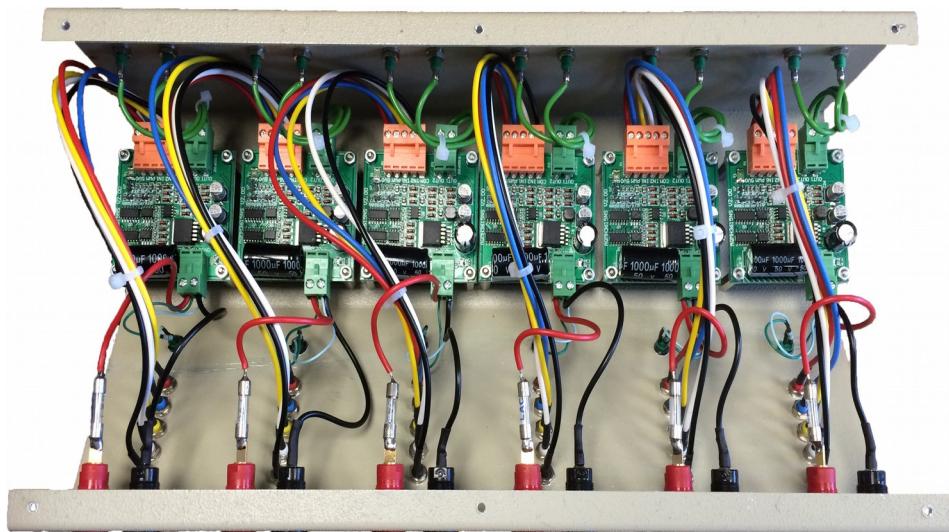
Les transistors sont connus pour être fortement sollicités durant leur fonctionnement. En effet, les transistors ne sont pas parfaits, ils possèdent une légère résistance. Cependant, le courant qui circule dans ces derniers est très important et peut monter jusque 9,7 A pour un moteur. Nous décidons maintenant de reproduire l'expérience sur un moteur avec un PWM à 100%. Nous maintenons l'arbre moteur pour réaliser une augmentation de courant significative. Afin de connaître l'augmentation de la température des transistors, nous avons utilisé un thermomètre infrarouge qui permet de mesurer la température à distance, sans le mettre en contact avec la carte. Nous avons choisi ce type de thermomètre afin d'éviter de créer malencontreusement un court-circuit sur la carte.

Comme il est possible de le voir sur la photo ci-contre, nous sommes restés bien loin des limites de la plage de températures annoncées par le constructeur. Cela nous a donc permis de savoir qu'il n'est pas nécessaire d'installer un ventilateur ou autre système de refroidissement (module Peltier, ...) au niveau de ces transistors.



5. Conception du boîtier des drivers

Une fois ces essais terminés, nous sommes passés à la conception du boîtier des drivers. Nous avons pour cela récupéré une boîte métallique dans laquelle nous avons placé les six drivers côte à côté. De plus, nous avons ajouté des fusibles au niveau de l'entrée de chaque alimentation des drivers pour s'assurer de ne pas les endommager durant les futurs essais. Afin de signaler qu'un driver est alimenté, nous avons également ajouté des leds témoins à l'entrée de chacun d'entre eux. Voici l'intérieur et l'extérieur de la boîte à l'état final :



Une fois cette boîte finie, nous avons réalisé des tests afin de vérifier son bon fonctionnement. Nous avons donc vérifié que tous les drivers fonctionnaient correctement. Puis nous avons contrôlé deux moteurs simultanément. Nous avons réalisé pour cela un programme Arduino (voir annexe) qui permet de contrôler les deux moteurs indépendamment via l'envoi de commande par la liaison série qui sépare l'Arduino et l'ordinateur. Voici l'ensemble des commandes que nous avons réalisées.

```

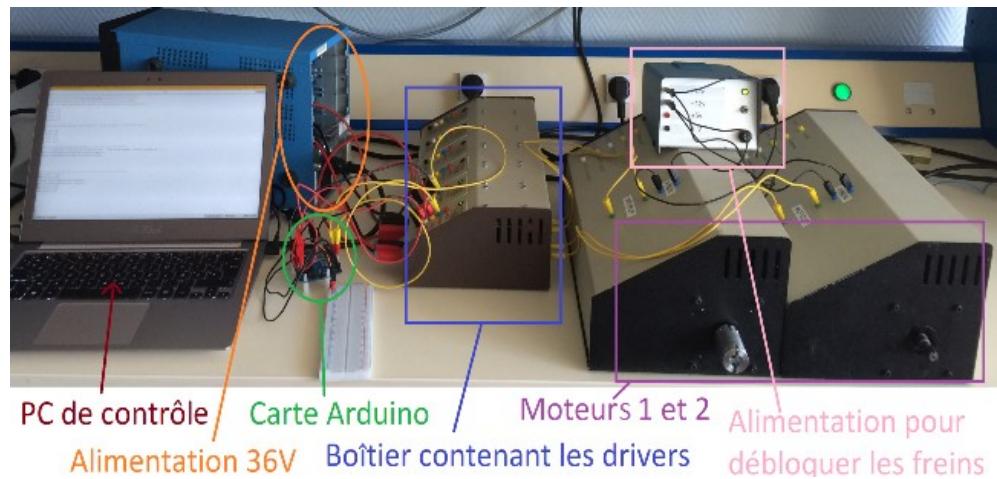
// TABLE DES FONCTIONS //
Entrez les valeurs suivantes pour modifier la vitesse du moteur 1 en sortie du driver 1:
1 pour 0 %
2 pour 25%
3 pour 50 %
4 pour 75 %
5 pour 100 %

Entrez les valeurs suivantes pour modifier la vitesse du moteur 2 en sortie du driver 2:
A pour 0 %
Z pour 25%
E pour 50 %
R pour 75 %
T pour 100 %

Entrez les valeurs suivante pour modifier le sens de rotation des moteurs
( ( !!! Attention, la modification du sens du moteur arrête automatiquement le moteur en question !!! ) )
Entrez Q pour modifier le sens de rotation du moteur 1
Entrez S pour modifier le sens de rotation du moteur 2

```

Voici une photo prise pendant les essais :



e) Conception de la manette de commande

1. Introduction de la partie

Dans cette partie, nous étudierons la conception de la manette de commande qui permettra d'envoyer les informations nécessaires pour commander le robot.

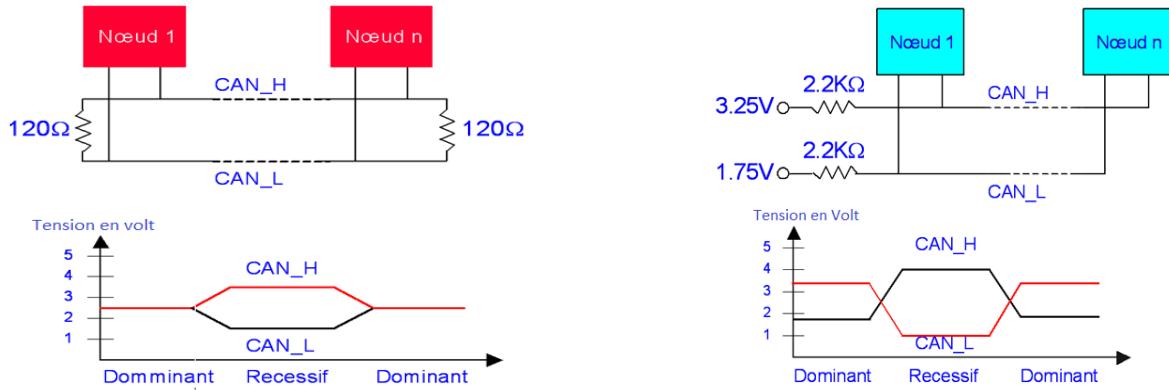
2. Principe de fonctionnement du BUS CAN

Le CAN BUS est un réseau multiplexé sur lequel de nombreux modules sont reliés afin qu'ils puissent communiquer ensemble. Le CAN BUS est relié à plusieurs équipements appelés "nœuds". Chaque nœud peut communiquer avec les autres via le bus. Un nœud est soit maître, c'est-à-dire qu'il peut donner des ordres, soit esclave, c'est-à-dire qu'il reçoit des ordres. Cette communication se fait via des messages transmis avec un protocole très précis.

Par ces protocoles, les nœuds maîtres peuvent réaliser des trames de requête qui consiste à poser une question et des trames de données qui consiste à envoyer des données. Ces deux types de trames sont destinées à un ou plusieurs équipements disposés sur le bus. Les nœuds esclaves peuvent quant à eux uniquement envoyer des trames de données.

Le bus se compose de deux lignes qui sont le « CAN High » et le « CAN Low ». Afin de faire fonctionner le CANBUS, il faut également une masse commune pour ne pas être en "flottant", cela permet d'avoir une référence commune. Dans de nombreux secteurs tel que celui de l'automobile, cette masse commune est prise directement sur le châssis des véhicules ce qui limite le nombre de fils à installer.

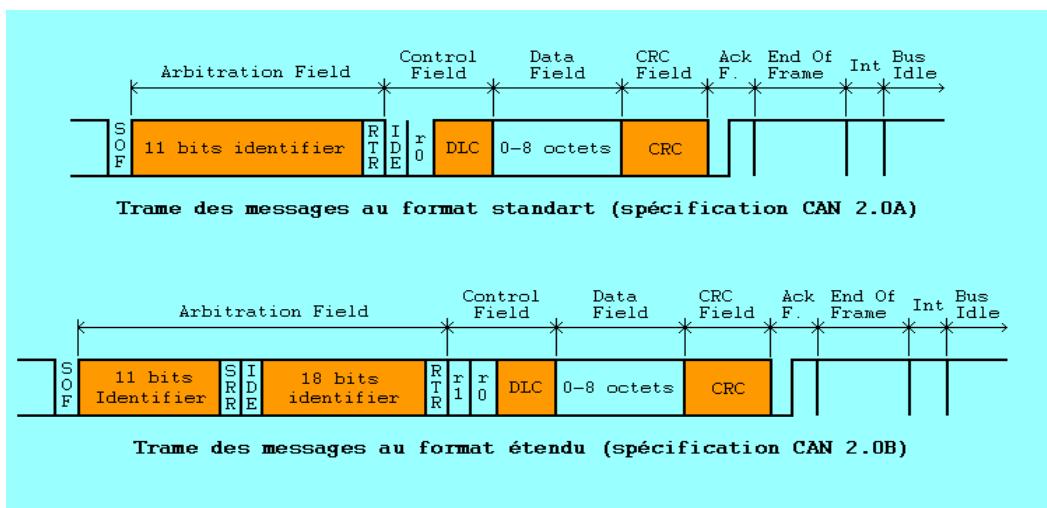
Différentes normes existent concernant le CAN au niveau des connexions et des tensions présentent sur le bus. On peut voir ci dessous la norme ISO11519-2 sur le CAN BUS basse vitesse (vitesse inférieure à 125Kbps) ainsi que la norme ISO11898 qui est la norme du CAN BUS en haute vitesse (vitesse est comprise entre 125Kbps et 1 Mbps).



Lors de notre étude sur le CANBUS, nous nous sommes uniquement servis de la norme ISO11898

(High Speed).

3. Forme des trames



Start of frame (SOF):

Il s'agit d'un bit dominant qui marque le début d'une trame de donnée (Data Frame) ou une trame de requête (Remote Frame). Ce bit a pour but également de synchroniser tous les nœuds avec celui émetteur du message.

Zone d'arbitrage (Arbitration field):

Il s'agit de 11 bits de l'identificateur ainsi que d'un bit de demande de transmission à distance (connu aussi sous l'acronyme RTR (remote transmission request)). Le RTR indique s'il s'agit d'une trame de requête ou de donnée. (RTR est un bit dominant pour une trame de donnée).

Champ de contrôle (Control field):

Ce champ est composé de 6 bits. Les deux premiers bit sont réservés, les 4 autres déterminent la longueur en octet de la trame de donnée (Data length Code (DLC)).

Champ de données (Data Field):

Il s'agit des données transmises. Elles sont comprises entre zéro et huit octets. Il convient de savoir que chaque octet est transmis avec le bit de poids fort en premier.

Code de redondance cyclique (Cyclic Redundancy Code (CRC Field)):

Cette partie de trame permet de repérer des erreurs. Le CRC Field est constitué des bits qui sont recalculés à la réception pour les comparer aux bits reçus. Si une différence est repérée, une erreur dite CRC est déclarée.

Zone d'acquittement (ACK (Aknowledge) Field):

La zone d'acquittement se constitue de deux bit. Le premier est le ACK Slot (emplacement de l'acquittement). Le second est l' ACK Délimiteur (délimiteur de l'acquittement) qui correspond au bit récessif.

Le nœud qui transmet l'information envoie un ACK Slot, si le noeud récepteur reçoit toutes les informations, alors il émet un ACK Delimiter.

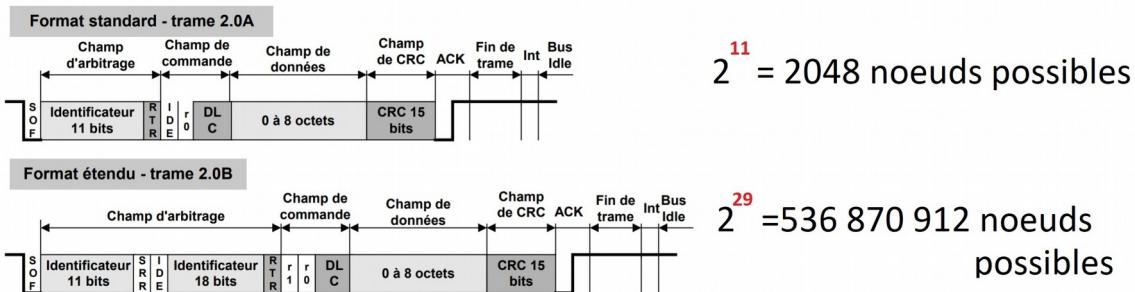
Fin de trame (End of Frame (EOF)):

Il s'agit de 7 bits récessif envoyés pour indiquer qu'il s'agit de la fin de la trame.

Stuff bit:

Pendant la construction des trames, si 5 bits consécutif ont la même valeur, alors un bit d'état opposé est ajouté à la suite. Cela permet de faire la différence avec une EOF.

Différence importante à signaler entre ces deux normes :

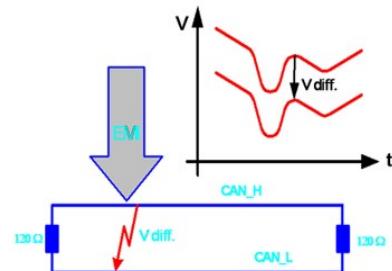


4. Intérêt du BUS CAN

Nous avons décidé d'utiliser ce type de communication car il présente de nombreux avantages :

- Réduction de la quantité de câbles et donc moins d'entretien.
- Travail sur de longues distances (contrairement à la communication I2C).
- Peu coûteux à mettre en place.
- Fonctionnement en multi-maître asynchrone (avec contrôle par priorité).
- Détection d'erreur incluse au protocole.
- Acquittement des messages par tous les nœuds.
- Gestion autonome de toutes erreurs temporaires rencontrées.
- Haute sécurité.

Le bus CAN fonctionne sur une différence de potentiel. Cela permet de rejeter au maximum les perturbations électro-magnétiques. Comme on peut le voir ci-dessous, malgré la perturbation, la différence de potentiel entre les deux lignes reste la même. L'information n'est donc pas perdue.



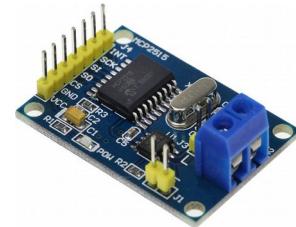
En conclusion le BUS CAN très réputé pour sa fiabilité, sa sécurité ainsi que sa robustesse.

5. Réalisation d'une communication BUS CAN via arduino

Afin d'apprendre à développer un BUS CAN, nous nous sommes décidés à commencer sur Arduino. C'est une méthode rapide et pédagogique permettant de se familiariser avec le BUS CAN.

Nous nous sommes confrontés à de nombreux problèmes notamment au niveau des librairies qui sont différentes au niveau de ce type de communication en fonction des quartz utilisés, du type d'Arduino et aussi des shields pré-existants.

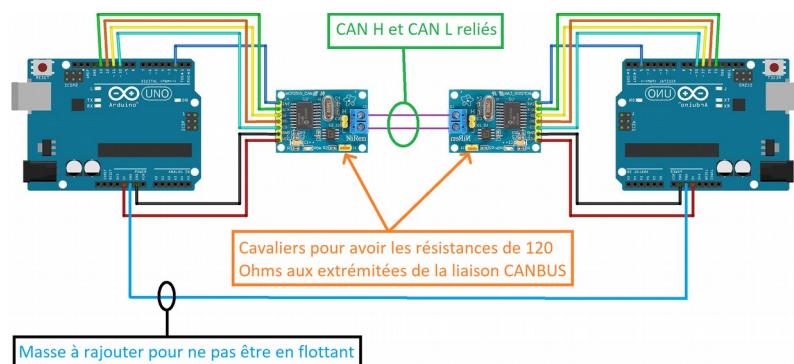
Nous disposons de deux modules MCP2515 en sortie des cartes Arduino.



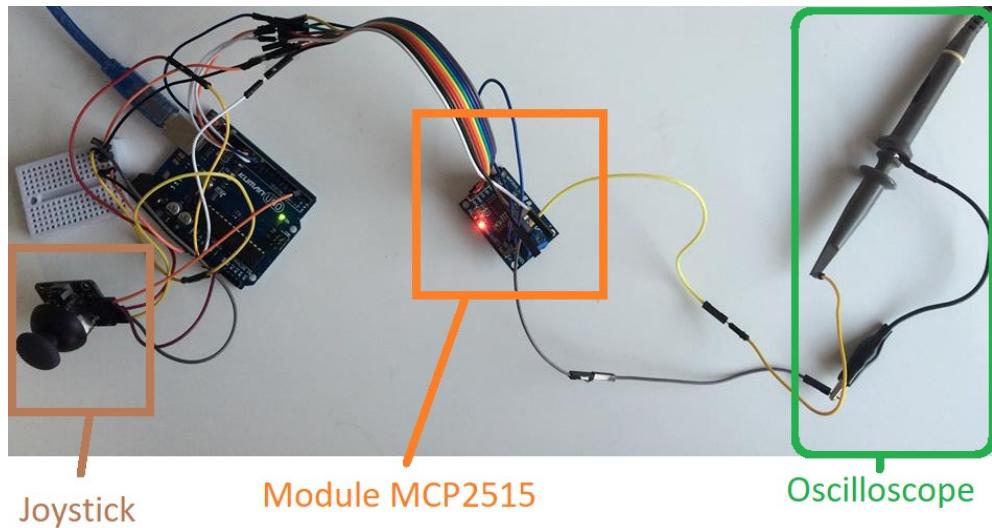
Ce module est un contrôleur CANBUS et comprend par conséquent le protocole employé par celui-ci.

La plupart des librairies que nous avons trouvées concernaient les modules MCP2515 associés avec un quartz de 16 MHz. Nous avons donc remplacé le quartz de 8 MHz pré-existent sur le MCP2515 par un quartz de 16MHz. Nous avons incorporé un programme émetteur qui émet sans cesse un même message sur le réseau, puis, un programme récepteur dans le second Arduino. Cependant, les données que nous envoyions n'étaient pas reçues par le second Arduino. Le problème venait d'un mauvais contact sur le bornier présent sur le MCP2515 mais également d'une masse commune manquante entre les deux cartes. Sans cette masse commune, nous étions en flottant et aucune transmission n'est possible dans ce cas.

Voici notre montage :



Voici une photo réalisée pendant les tests :

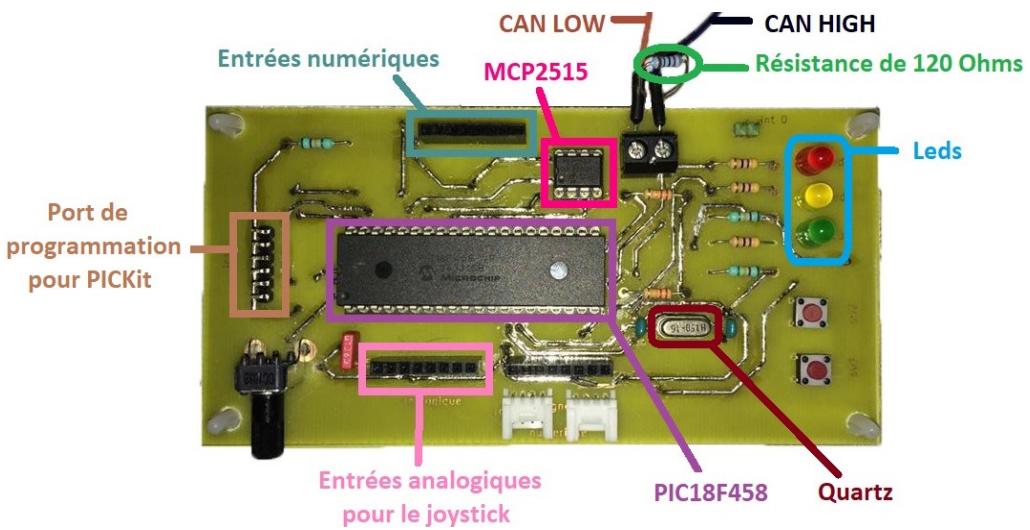


Voici une trame du type CAN BUS visualisée à l'oscilloscope :



6. Réalisation d'une communication CAN BUS via un microcontrôleur PIC

Après avoir réalisé les premiers essais du CAN BUS à l'aide d'un Arduino, nous avons décidé de réaliser cette communication via un PIC18F458 qui est un microcontrôleur de la marque Microchip. Ce microcontrôleur était monté sur une carte qui était déjà réalisée par les techniciens.



Comme nous pouvons le voir, la carte possède déjà le composant MCP2515 qui est le même contrôleur de communication CAN utilisé sur le contrôleur CAN BUS de la marque Arduino.

Ce microcontrôleur est utilisable dans notre application car il possède:

- deux pattes destinées pour le CAN BUS
- suffisamment d'entrées analogiques pour les futurs joysticks
- un système de Conversion Analogique Numérique
- suffisamment d'entrées numériques pour les futurs boutons poussoirs
- il se programme à l'aide d'un PIKit 3 ce qui est disponibles en bâtiment P2

Communication d'un PIC vers un autre :

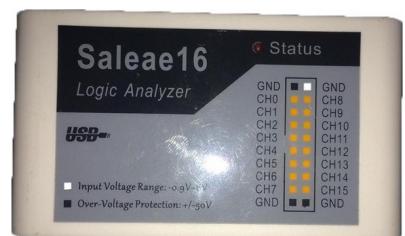
Après de nombreuses recherches et essais, nous avons trouvé une librairie CAN.H et CAN.C avec des fonctions déjà réalisées. Nous avons alors essayé de concevoir un programme simple pour envoyer un message d'un PIC à un autre sans oublier de mettre les résistances de 120 Ohms de part et d'autre du CAN BUS. Cependant nous ne parvenions pas à recevoir les messages envoyés. Grâce à l'oscilloscope, nous nous sommes aperçus qu'une des cartes ne fonctionnait pas à cause du microcontrôleur ou du MCP2515.

Communication d'un PIC vers un Arduino :

Nous avons alors essayé de transmettre des trames directement à l'Arduino programmé en réception tout en vérifiant que nous avions bien un signal de type CAN à l'oscilloscope. Cependant, aucune trame n'a été reçue par l'Arduino.

En comparant les trames d'envoi effectuées par le microcontrôleur PIC et l'Arduino, nous avons remarqué que le signal sortant du microcontrôleur Microchip n'avait pas la bonne fréquence. Cela explique que nous ne recevions pas les messages. Nous avons cherché dans la librairie, mais le bon quartz à utiliser n'était pas indiqué.

Afin de ne pas rester bloqués, nous avons utilisé un analyseur logique "SALEAE 16". Celui-ci nous permet de voir directement la trame qui circule sur le réseau en indiquant s'il s'agit bien d'une trame de type CAN BUS et permet également de connaître la fréquence du signal.



Nous avons tout d'abord cherché à changer de quartz pour tomber sur la même fréquence. Cependant nous n'y sommes pas parvenus. C'est pourquoi nous avons décidé de changer, en plus des quartz mis à notre disposition, les bits qui modifient le prescaler afin de modifier plus légèrement la fréquence du signal envoyé par notre programme. Un prescaler est un circuit électronique qui réduit la fréquence du quartz. Cette fréquence peut être plus ou moins modifiée par l'utilisateur en modifiant certains bits de configuration. Dans notre cas, nous avons modifié les bits du registre BRGCON1 (page 218 de la datasheet).

REGISTER 19-29: BRGCON1: BAUD RATE CONTROL REGISTER 1

| R/W-0 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| SJW1 | SJW0 | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 |
| bit 7 | | | | | | | bit 0 |

bit 7-6	SJW1:SJW0: Synchronized Jump Width bits 11 = Synchronization Jump Width Time = 4 x T _Q 10 = Synchronization Jump Width Time = 3 x T _Q 01 = Synchronization Jump Width Time = 2 x T _Q 00 = Synchronization Jump Width Time = 1 x T _Q
bit 5-0	BRP5:BRP0: Baud Rate Prescaler bits 111111 = T _Q = (2 x 64)/Fosc 111110 = T _Q = (2 x 63)/Fosc : : 000001 = T _Q = (2 x 2)/Fosc 000000 = T _Q = (2 x 1)/Fosc

Illustration 34: Extrait de la datasheet du PIC18F458

Une fois la bonne fréquence trouvée, nous avons observé sur la trame qu'elle était en format étendu. Dans notre cas, ce format étendu n'est pas utile. Pour ce faire, nous avons mis à zéro le bit EXIDEN (page 215 de la datasheet) afin de passer en format standard avec 11 bits d'identification.

```
// registre trame standard
TXB0SIDL = 0b00000011; // 2 bits identifier + standart frame (bit n°4) + extender identifier bit (n°1 et 2)
TXB0SIDH = 0b00001110; // 8 premiers bits identifier
// ici on a l'adresse 0xAA 0x00 = 1010 1010 0000 XXXX = 0x550
// si on met par exemple 0xAA 0xF0 on a 0x557 = 0101 0101 0111
```

Illustration 35: Extrait du programme réalisé.

Une fois ces étapes accomplies, nous sommes parvenu à envoyer des données via des trames CAN BUS à l'Arduino.

Récupération des valeurs du joystick sur un microcontrôleur PIC :

Un joystick est un module qui se compose de deux potentiomètres permettant d'acquérir, en fonction de la résistance, la position x et y du joystick. Afin de récupérer les valeurs de ce dernier, il est nécessaire de faire une Conversion Analogique Numérique ("CAN", à ne pas confondre avec Controller Area Network). Nous avons tout d'abord réalisé la fonction *initADC()* qui réalise l'initialisation de notre convertisseur :

```
void initADC()
{
    //paramètre convertisseur analogique numérique
    ADCON0 = 0;
    ADCON1 = 0;
    ADCON0bits.ADCS1 = 1; // on ajuste le temps de conversion par rapport a la vitesse du processeur
    ADCON1bits.PCFG = 0x0100; // an0 et 1 en analogique
    ADCON0bits.ADON = 1; // on active le convertisseur
}
```

Illustration 36: Extrait du programme réalisé sur la fonction initADC().

Ensuite, nous avons créé une fonction *getADC()* qui, selon son paramètre d'entrée, convertit la valeur de l'axe x ou y du joystick.

```

//Fonction qui reçoit 0 ou 1 en fonction de la voie à tester (X ou Y)
void getADC(int pos)
{
    unsigned int result;
    ADCON0bits.ADON = 0; // on désactive le convertisseur
    ADCON0bits.CHS = pos; // on récupère la valeur de l'appui bouton
    ADCON0bits.ADON = 1; // on active le convertisseur

    for(result = 0; result < 1000; result++)
    {
        //On laisse au minimum un cycle de conversion
        Nop(); // indication de la datasheet, il ne faut pas faire l'activation du convertisseur et la lecture à la suite
    }
    ADCON0bits.GO = 1; // On lance la conversion

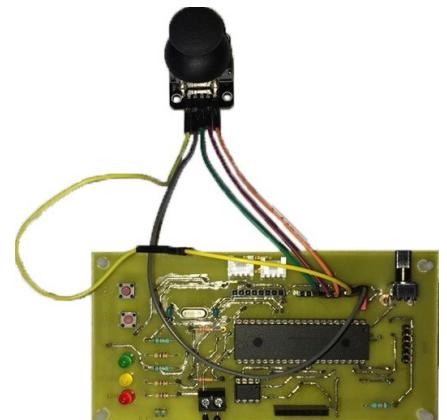
    while(ADCON0bits.DONE == 1); // On attend que le bit DONE passe à 1, cela indique la fin de la conversion
}

```

Illustration 37: Extrait du programme réalisé sur la fonction getADC().

Pour vérifier son bon fonctionnement, nous avons réalisé un programme simple pour allumer les leds de la carte en fonction de la position x puis y du joystick (photo des tests en annexe).

Quant au bouton poussoir du joystick, nous lisons la donnée avant de la stocker dans un buffer. Un buffer est une mémoire où l'on stocke les valeurs avant de les envoyer.



Nous lui avons également rajouté une résistance de pull-up qui n'est pas installée d'origine.

```

//on stocke la valeur du bouton poussoir
buffCANBUS[4] = !PORTAbits.RA4; // Valeur du bouton poussoir du joystick
//(l'inversion ("!") sert à avoir une logique d'appuyé 1 appuyé, 0 non appuyé)

```

Illustration 39: Extrait de programme sur le stockage de la valeur du bouton tout ou rien du joystick.

Envoi des données du joystick sur le réseau CANBUS :

Pour conclure, une fois ces parties maîtrisées, nous avons stocké les variables dans un tableau de buffers afin de les envoyer dans une trame sur le réseau CANBUS. Voici ci-dessous les trames obtenues grâce au logiciel de l'analyseur logique :

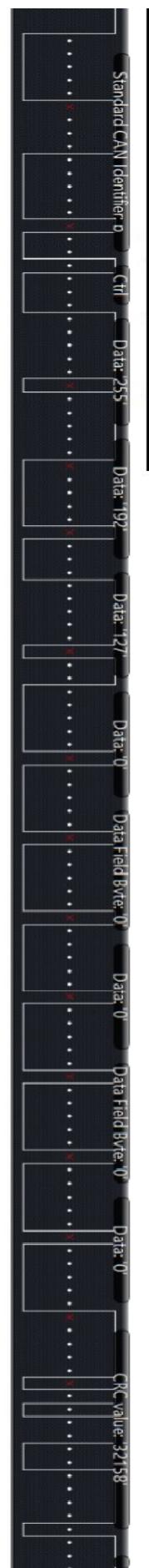
Trame obtenue avec le joystick placé vers le bas :



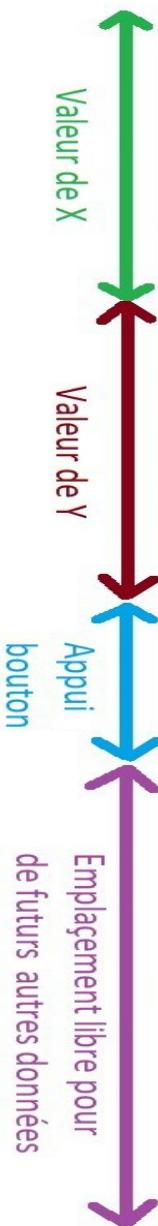
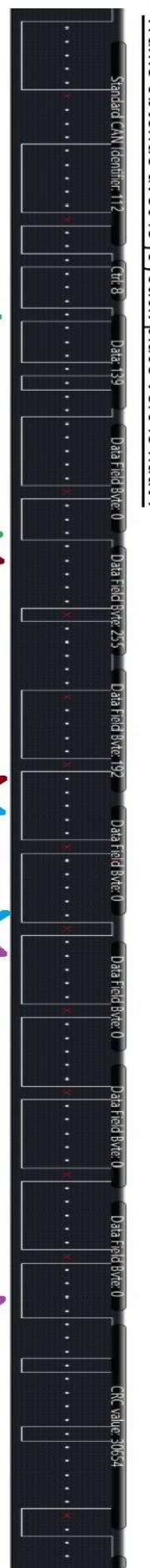
Trame obtenue avec le joystick placé au centre et enfoncé :



Frame obtenue avec le joystick placé à droite :



Trame obtenue avec le joystick placé à gauche :



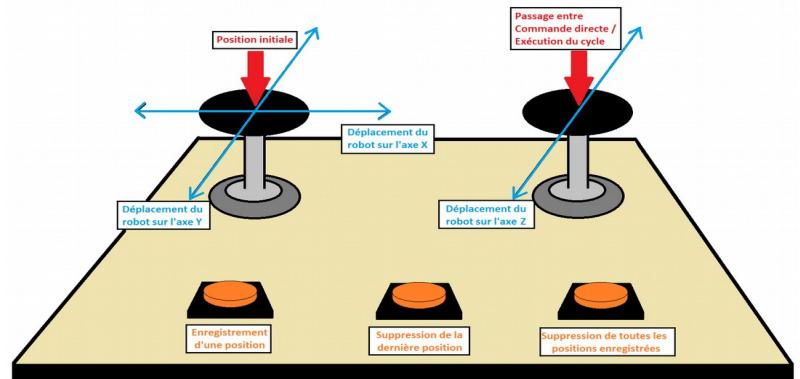
Comme nous pouvons le voir ci-dessus, la trame se compose tout d'abord de deux octets contenant la valeur de la position x, de deux octets contenant la valeur de la position y, puis de un octet pour l'appui du bouton poussoir du joystick.

Différents problèmes rencontrés durant cette partie :

Nous avons dû prendre en main le logiciel MPLAB afin de programmer correctement le microcontrôleur. De plus, l'une des deux cartes avait un problème au niveau du PIC ou du MCP2515. Pour finir, nous avons rencontré de nombreux problèmes de faux contact au niveau des fils entre la carte Arduino et le contrôleur CAN BUS.

7. Idée de conception de la manette

Nous avons pensé à la conception d'une manette permettant le contrôle du robot. Le programme contiendra un mode où l'utilisateur peut contrôler directement le robot en temps réel et enregistrer plusieurs positions. Et un second mode où le robot exécutera en boucle les positions précédemment enregistrées. Voici les éléments que la manette pourrait contenir :



Joystick numéro 1 :

- Déplacement du robot sur les positions x/y
- Bouton de pression du joystick pour mettre le robot en position initial
(bras dirigé vers le haut)

Joystick numéro 2 :

- Déplacement du robot sur la position z
- Passage en commande directe/exécution du cycle

Boutons poussoirs :

- Bouton enregistrement d'une position
- Bouton suppression de la dernière position enregistrée
- Bouton suppression de toutes les positions enregistrées

PARTIE IV : ASSERVISSEMENT ET PILOTAGE GÉNÉRAL

Rédigé par Guo Kaiqi & Boudejelthia Abdelfettah

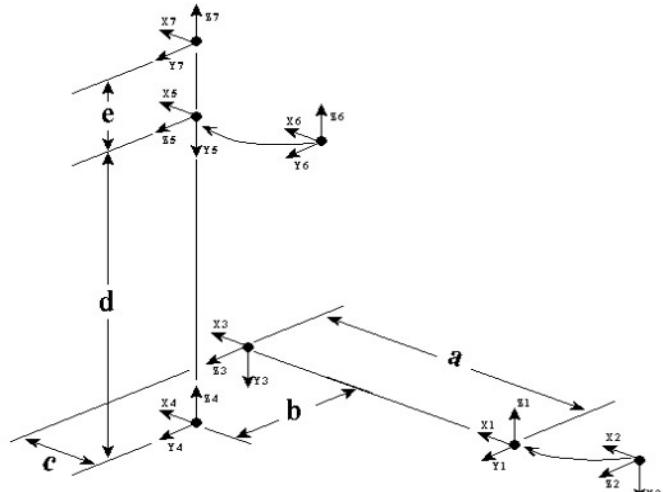
a) Établissement des coordonnées des articulations du robot Puma

Le manipulateur robotique Puma 500/560 est un bras motorisé composé de 6 axes de rotation contrôlés par microcontrôleur [1].

Afin de pouvoir contrôler et commander le robot puma 500 par une console (un circuit imprimé avec micro-processeur), on est obligé de passer par l'étape de modélisation du système en question ce qui est l'objectif de cette partie du projet.

Schématisation du robot Puma 500 :

La figure ci-contre (Figure N°1) montre vivement la représentation des articulations de notre système en question (bras de robot manipulateur Puma 500) selon le professeur Luc Baron.



Après avoir adopté cette représentation, l'étape qui suivre consiste à rendre ce système mécanique sous format mathématique et pour se faire, on propose la méthode de Denavit-Hartenberg, où cette dernière nous permet de bien déterminer les angles nécessaires pour amener l'effecteur a des cordonnées cartésiennes souhaités et vice versa dans la mesure d'effectuer une tache bien précise. Cette méthode qu'on vient de la citer, on va la détailler dans le point suivant (par la suite), ou on montrera le modèle mathématique qui décrive notre système en question.

- **Modélisation du Robot Puma 500 avec la méthode de Denavit-Hartenberg (D-H)**

Comme on l'a déjà précisé dans le point précédent, pour cette partie on va expliquer le modèle mathématique qui régit le robot puma 500.

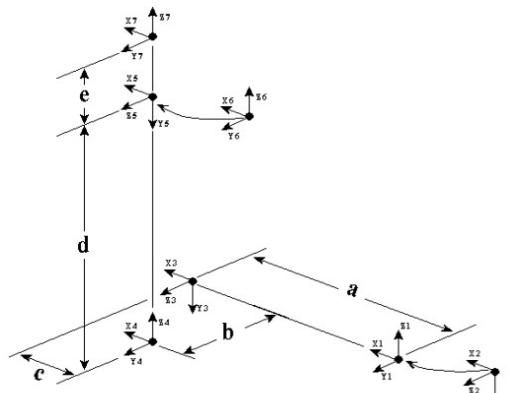
Le modèle mathématique était construit avec la méthode de Denavit-Hartenberg, où cette dernière comporte 4 grandes parties essentielles :

- f) L'attribution à chaque articulation un repère (x_i, y_i, z_i) .
- g) L'Etablissement (la construction) de la table des paramètres de Denavit-Hartenberg (D-H).
- h) Le calcul du Modèle Géométrique Direct (MGD).
- i) Le calcul du Modèle Géométrique Inverse (MGI).

On s'est servis du travail du professeur *Luc Baron*. On détaillera par la suite chaque étape de la méthode D-H :

- Etape N°1 : L'attribution à chaque articulation un repère (x_i, y_i, z_i) :

Pour cette représentation le repère associé à l'extrémité de l'effecteur (le point entre les pinces) est confondu à celui de l'effecteur (le dernier moteur du robot puma 500).



- Etape N°2 : L'établissement (la construction) de la table des paramètres de Denavit-Hartenberg (D-H) :

Le tableau des paramètres de Denavit-Hartenberg selon l'approche de Pr. Baron [1]:

i	Thêta θ	a_i (translation suivant l'axe x_i)	b_i (translation suivant l'axe z_i)	α_i (en °)
1	θ_1	0	0	-90
2	θ_2	a	0	0
3	θ_3	c	b	+90
4	θ_4	0	d	-90
5	θ_5	0	0	+90
6	θ_6	0	E	0

Où les valeurs des segments sont comme suit :

$$a = 431.81 \text{ mm}, \quad b = 149.10 \text{ mm}, \quad c = -20.31 \text{ mm}, \quad d = 433.04 \text{ mm}, \quad e = 56.23 + 97.5 \text{ mm}$$

- Etape N°3 : Le calcul du Modèle Géométrique Direct (MGD) :

Le modèle géométrique direct d'un robot manipulateur est la fonction f qui permet d'exprimer la situation de l'effecteur terminal du robot manipulateur par rapport à un repère de référence en fonction des variables articulaires.

Le Modèle Géométrique Direct (MGD) décrit la transformation des coordonnées du repère universel $R_0(x_0, y_0, z_0)$ au repère de l'effecteur $R_6(x_6, y_6, z_6)$, d'une autre façon, c'est le mouvement que fait l'effecteur par rapport au repère universel, l'algorithme pour avoir le Modèle Géométrique Direct est comme suit :

- I. Associer à chaque articulation un repère (x_i, y_i, z_i) , où i varie de 1 à n « n : nombre des articulations » ou dans ce cas $n=6$.
 - Axe Z_i est défini par: la même direction de axe de rotation
 - Axe X_i est défini par: la ligne normale commune de la Z_i Z_{i-1} , pointant à l'opposé de l'axe Z_{i-1} .

c. axe Yi est définir par: selon la règle du système de coordonnées rectangulaires de droite.

II. Établir les matrices de passage $A_i = T_i^{i-1}$ qui décrit le passage des cordonnées du repère R_{i-1} au repère R_i .

III. Calculer la matrice finale $T_6^1 = \prod_{i=1}^6 A_i$.

$$T_6^1 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \vec{R}_6^1 & \vec{P}_6^1 \\ \vec{0} & 1 \end{bmatrix}^{[2]}$$

$$\vec{R}_6^1 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad \vec{P}_6^1 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}, \quad \vec{0} = [0 \ 0 \ 0]$$

$$r_{11} = C_1 [C_{23}(C_4 C_5 C_6 - S_4 S_6) - S_{23} S_5 C_6] - S_1 (S_4 C_5 C_6 + C_4 S_6).$$

$$r_{21} = S_1 [C_{23}(C_4 C_5 C_6 - S_4 S_6) - S_{23} S_5 C_6] - C_1 (S_4 C_5 C_6 + C_4 S_6).$$

$$r_{31} = -S_{23}(C_4 C_5 C_6 - S_4 S_6) - C_{23} S_5 C_6.$$

$$r_{12} = C_1 [C_{23}(-C_4 C_5 S_6 - S_4 C_6) - S_{23} S_5 S_6] - S_1 (-S_4 C_5 S_6 + C_4 C_6).$$

$$r_{22} = S_1 [C_{23}(-C_4 C_5 S_6 - S_4 C_6) - S_{23} S_5 S_6] + C_1 (-S_4 C_5 S_6 + C_4 C_6).$$

$$r_{32} = -S_{23}(-C_4 C_5 S_6 - S_4 C_6) - C_{23} S_5 S_6.$$

$$r_{13} = C_1 (C_{23} C_4 S_5 + S_{23} C_5) - S_1 S_4 S_5.$$

$$r_{13} = S_1 (C_{23} C_4 S_5 + S_{23} C_5) + C_1 S_4 S_5.$$

$$r_{13} = -S_{23} C_4 S_5 - C_{23} C_5.$$

$$p_x = f_1(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$$

$$\textcolor{red}{i} a C_1 C_2 - b S_1 + c C_1 C_{23} + d C_1 S_{23} + e (C_1 (C_{23} C_4 S_5 + S_{23} C_5) - S_1 S_4 S_5)$$

$$p_y = f_2(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$$

$$\textcolor{red}{i} a S_1 C_2 + b C_1 + c S_1 C_{23} + d S_1 S_{23} + e (S_1 (C_{23} C_4 S_5 + S_{23} C_5) + C_1 S_4 S_5)$$

$$p_z = f_3(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6)$$

$$\textcolor{red}{i} -a S_2 - c S_{23} + d C_{23} + e (-S_{23} C_4 S_5 + C_{23} C_5).$$

IV. Après avoir obtenir la matrice globale T_6^0 , les coordonnées ($x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}$) seront les éléments de la matrice T_6^0 :

$$a. \quad x_{\text{effecteur}} = T_6^0(1,4) = p_x = f_1(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6).$$

$$b. \quad y_{\text{effecteur}} = T_6^0(2,4) = p_y = \textcolor{red}{i} f_2(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6).$$

$$c. \quad z_{\text{effecteur}} = T_6^0(3,4) = p_z = \textcolor{red}{i} f_3(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6).$$

d. Etape N°4 : Le calcul du Modèle Géométrique Inverse (MGI) :

Le modèle géométrique inverse permet de déterminer le vecteur des variables articulaires à partir du vecteur de coordonnées opérationnelles.

Après avoir calculé le Modèle Géométrique Direct (MGD), on va faire le chemin inverse et on tire les expressions des angles de rotation des 6 moteurs en fonction de $x_{\text{effecteur}}$, $y_{\text{effecteur}}$ et $z_{\text{effecteur}}$:

$$\begin{aligned}\theta_1 &= f_{\theta_1}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}). \\ \theta_2 &= f_{\theta_2}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}). \\ \theta_3 &= f_{\theta_3}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}). \\ \theta_4 &= f_{\theta_4}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}). \\ \theta_5 &= f_{\theta_5}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}). \\ \theta_6 &= f_{\theta_6}(x_{\text{effecteur}}, y_{\text{effecteur}}, z_{\text{effecteur}}).\end{aligned}$$

Après avoir établir le MGD et MGI, on pourra passer à l'étape de programmation, ou cette partie se fait sous MATLAB.

Pour le programme complet de Matlab, voir **Annexe – Matlab**

b) Asservissement

- **Le problème - Contrôle de la position du servo moteur .**

Avant de commencer à concevoir un contrôleur PID, nous devons comprendre le problème. Dans cet exemple, nous voulons déplacer l'arbre du moteur de sa position actuelle à la position cible.

Nous voulons déplacer l'arbre de sortie du moteur de la position actuelle à la position cible

Il existe quelques termes couramment utilisés pour décrire les boucles de contrôle PID, tels que:

Variable de contrôle (CV) - C'est la sortie de la boucle de contrôle. Dans ce cas, le CV est le cycle de service du signal PWM qui entraîne le moteur.

Variable de procédé (PV) - Il s'agit de la valeur de retour renvoyée par le système au contrôleur. Dans cet exemple, le PV est l'angle actuel de l'arbre du moteur.

Point de consigne (SP) - Le point de consigne est la valeur que nous souhaitons pour le système. Dans notre cas, le SP est la position cible de l'arbre du moteur en angle.

Erreur (E) - L'erreur fait référence à la différence entre le point de consigne et la variable de processus. En d'autres termes, cela signifie à quelle distance la position actuelle de l'arbre du moteur à partir de la position cible.

- **La mise en œuvre du contrôleur PID**

Le contrôleur PID, tout comme son nom, comprend une partie proportionnelle (P), une partie intégrale (I) et une partie dérivée (D). Les parties du contrôleur sont introduites dans les sections suivantes individuellement et en fonctionnement combiné.

Lorsque la position actuelle de l'arbre du moteur est encore loin de la position cible, nous voulons appliquer plus de puissance pour amener le moteur vers la position cible afin que nous puissions y arriver plus rapidement. Lorsque l'arbre se rapproche de la position cible, nous réduisons la puissance pour le ralentir. Au moment où l'arbre atteint la position cible, le moteur doit être arrêté. Si la position de l'arbre a dépassé, nous devons appliquer une puissance négative au moteur (inverser le moteur) pour le ramener à la position cible.

En bref, cela s'appelle un régulateur proportionnel car la puissance que nous appliquons au moteur est proportionnelle à l'erreur du système.

Le schéma de principe du contrôleur proportionnel

À partir du schéma de principe du régulateur proportionnel, nous pouvons voir que le cycle de service PWM (sortie) est le résultat de la multiplication de l'erreur par une constante, K_p .

Erreur = Définir le point - Variable de processus

Variable de contrôle = $K_p * \text{Erreur}$

Pour mieux visualiser les différents résultats, ici on trouve un modèle pour simuler le correcteur sur matlab

(J)	moment of inertia of the rotor	3.2284E-6 kg.m ²
(b)	motor viscous friction constant	3.5077E-6 Nms
(Ke)	electromotive force constant	0.0274 V/rad/sec
(Kt)	motor torque constant	0.0274 Nm/Amp
(R)	electric resistance	4 ohm
(L)	electric inductance	2.75E-6H

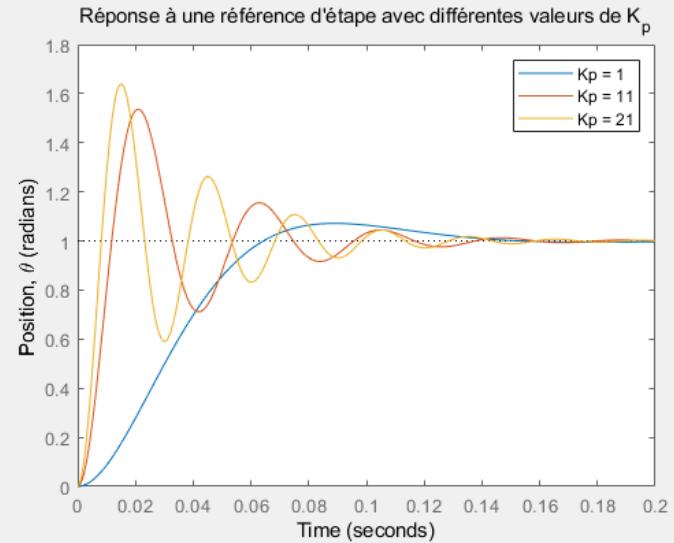
- Contrôleur Proportionnel

Code matlab

```
J = 3.2284E-6;
b = 3.5077E-6;
K = 0.0274;
R = 4;
L = 2.75E-6;
s = tf('s');
P_motor = K/(s*((J*s+b)*(L*s+R)+K^2));
```

```
Kp = 1;
for i = 1:3
    C(:,:,i) = pid(Kp);
    Kp = Kp + 10;
end
sys_cl = feedback(C*P_motor,1);

t = 0:0.001:0.2;
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title("Réponse à une référence d'étape avec différentes valeurs de K_p")
legend('Kp = 1', 'Kp = 11', 'Kp = 21')
```

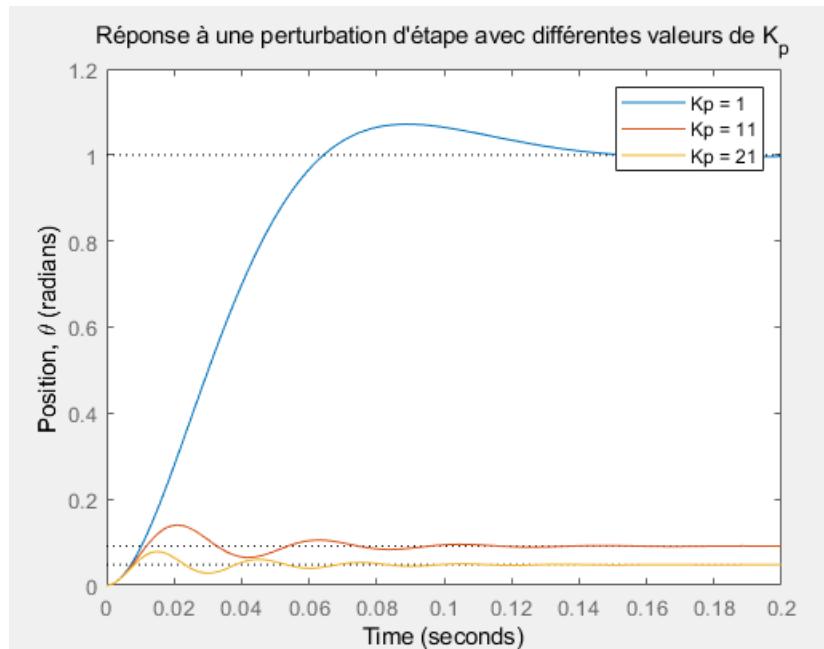


Considérons également la réponse du système à une perturbation d'échelon. Dans ce cas, nous supposerons une référence de zéro et verrons comment le système répond lui-même à la perturbation. La commande de retour peut encore être utilisée pour générer la fonction de transfert en boucle fermée où il y a encore une rétroaction négative, mais maintenant seule la fonction de transfert de l'installation P (s) est dans le chemin aller et le contrôleur C (s) est considéré comme dans le chemin de retour.

```

pertu_cl = feedback(P_motor,C);
step(pertu_cl(:,:,1), pertu_cl(:,:,2),pertu_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title("Réponse à une perturbation d'échelon avec différentes valeurs de K_p")
legend('Kp = 1', 'Kp = 11','Kp = 21')

```



Les graphiques ci-dessus montrent que le système n'a pas d'erreur en régime permanent en réponse à la référence d'échelon par elle-même, quel que soit le choix du gain proportionnel K_p . Cela est dû au fait que l'installation a un intégrateur. Cependant, le système présente une erreur significative en régime permanent lorsque la perturbation est ajoutée. Plus précisément, la réponse due à la référence et à la perturbation appliquée simultanément est égale à la somme des deux graphiques présentés ci-dessus. Cela résulte de la propriété de superposition qui vaut pour les systèmes linéaires. Par conséquent, pour avoir une erreur de zéro en régime permanent en présence d'une perturbation, il faut que la réponse de perturbation soit nulle. Plus la valeur de K_p est grande, plus l'erreur en régime permanent est faible, mais elle n'atteint jamais zéro. En outre, l'utilisation de valeurs de plus en plus grandes de K_p a pour effet négatif d'augmenter le temps de dépassement et de stabilisation, comme le montre le tracé de référence de l'échelon.

- Contrôleur PI

En ajoute un contrôleur PI pour se débarrasser de l'erreur d'état stable due à la perturbation. Nous allons définir $K_p = 21$ et tester les gains intégraux K_i allant de 100 à 500.

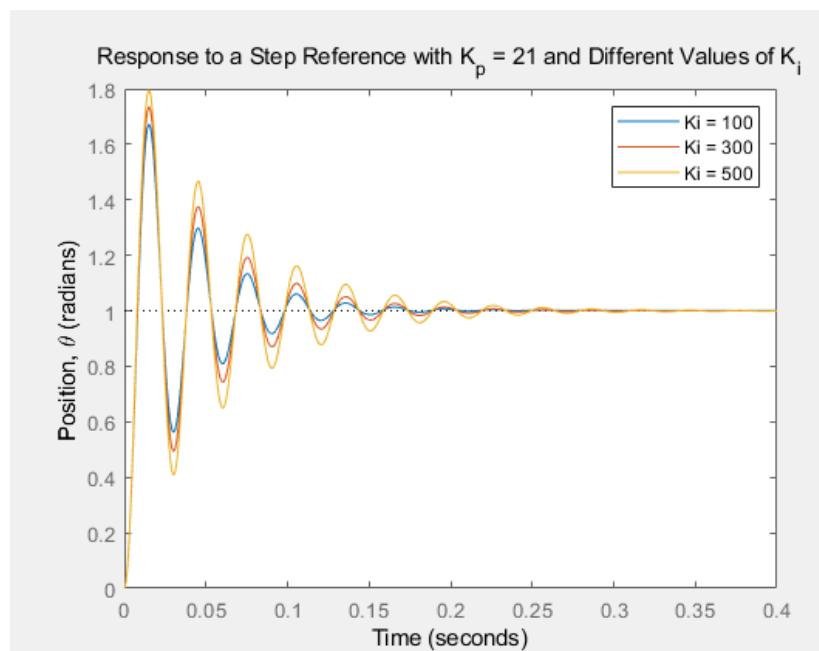
```

Kp = 21;
Ki = 100;

for i = 1:5
    C(:,:,i) = pid(Kp,Ki);
    Ki = Ki + 200;
end

sys_cl = feedback(C*P_motor,1);
t = 0:0.001:0.4;
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title("Réponse à une référence d'échelon avec K_p = 21 et différentes valeurs de K_i")
legend('Ki = 100', 'Ki = 300', 'Ki = 500')

```



Et aussi la réponse de perturbation de l'échelon

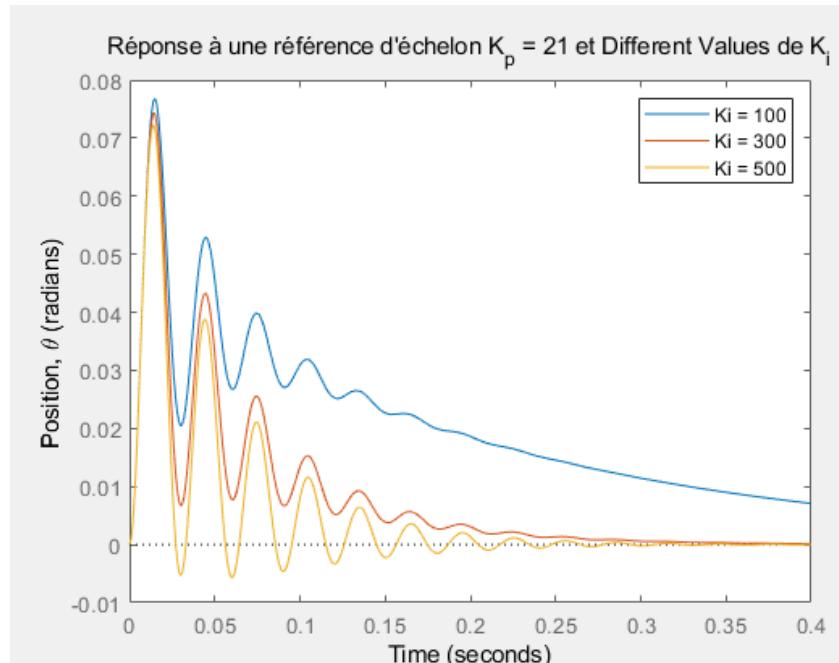
```

pertu_cl = feedback(P_motor,C);
step(pertu_cl(:,:,1), pertu_cl(:,:,2), pertu_cl(:,:,3), t)

ylabel('Position, \theta (radians)')

title("Réponse à une référence d'échelon  $K_p = 21$  et Different Values de  $K_i$ ")
legend('Ki = 100', 'Ki = 300', 'Ki = 500')

```



La commande intégrale a ramené l'erreur de régime permanent à zéro, même en présence d'une perturbation progressive; c'était l'objectif d'ajouter le terme intégral. Pour la réponse à la référence d'étape, toutes les réponses se ressemblent avec la quantité d'oscillation augmentant légèrement lorsque K_i est rendu plus grand. Cependant, la réponse due à la perturbation change de manière significative lorsque le gain intégral K_i est modifié. Plus précisément, plus la valeur de K_i utilisée est grande, plus l'erreur diminue rapidement. Nous choisirons $K_i = 500$ car l'erreur due à la perturbation décroît rapidement, même si la réponse à la référence a un temps de stabilisation plus long et plus de dépassement. Nous allons essayer de réduire le temps de stabilisation et de dépassement en ajoutant un terme dérivé au contrôleur.

- Contrôleur PID

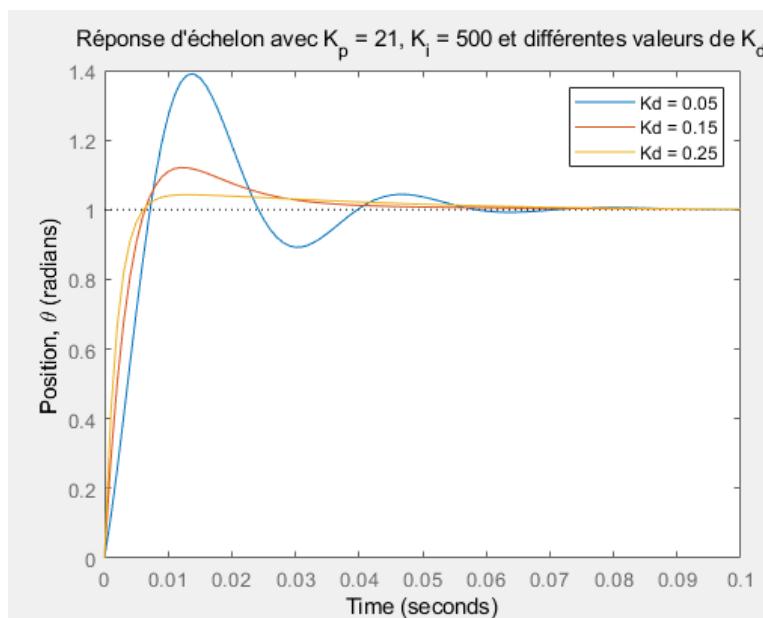
L'ajout d'un terme dérivé au contrôleur signifie que nous avons maintenant les trois termes du contrôleur PID. Nous étudierons les gains dérivés K_d allant de 0,05 à 0,25.

```

Kp = 21;
Ki = 500;
Kd = 0.05;
for i = 1:3
    C(:,:,i) = pid(Kp,Ki,Kd);
    Kd = Kd + 0.1;
end

sys_cl = feedback(C*P_motor,1);
t = 0:0.001:0.1;
step(sys_cl(:,:,1), sys_cl(:,:,2), sys_cl(:,:,3), t)
ylabel('Position, \theta (radians)')
title("Réponse d'échelon avec K_p = 21, K_i = 500 et différentes valeurs de K_d")
legend('Kd = 0.05', 'Kd = 0.15', 'Kd = 0.25')

```



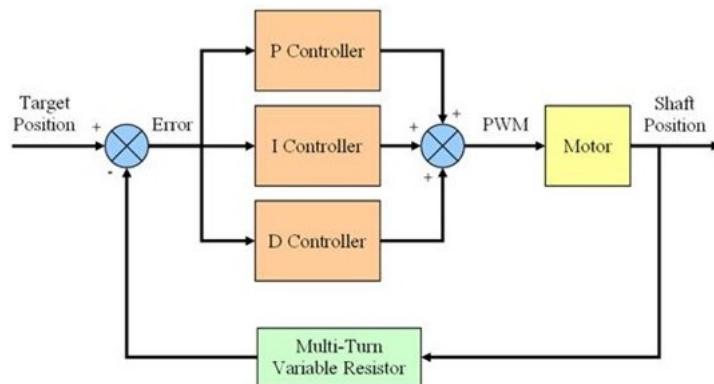
$kd=0.05$	$kd=0.15$	$kd=0.25$
<code>stepinfo(sys_cl(:,:,1))</code>	<code>stepinfo(sys_cl(:,:,2))</code>	<code>stepinfo(sys_cl(:,:,3))</code>
RiseTime: 0.0055	RiseTime: 0.0046	RiseTime: 0.0037
SettlingTime: 0.0537	SettlingTime: 0.0338	SettlingTime: 0.0435
SettlingMin: 0.8920	SettlingMin: 0.9103	SettlingMin: 0.9103
SettlingMax: 1.3904	SettlingMax: 1.1212	SettlingMax: 1.0431
Overshoot: 39.0372	Overshoot: 12.1175	Overshoot: 4.3065
Undershoot: 0	Undershoot: 0	Undershoot: 0
Peak: 1.3904	Peak: 1.1212	Peak: 1.0431
PeakTime: 0.0139	PeakTime: 0.0122	PeakTime: 0.0131

De ce qui précède, nous voyons que la réponse à une référence d'étape a un temps de stabilisation d'environ 34 ms (<40 ms), un dépassement de 12% (<16%) et aucune erreur d'état stable. De plus, la réponse de perturbation d'étape n'a pas non plus d'erreur en régime permanent. Nous savons maintenant que si nous utilisons un contrôleur PID avec

$$K_p = 21, K_i = 500 \text{ et } K_d = 0,15$$

- Conclusion

Le contrôleur PID est un système de contrôle simple mais efficace largement utilisé dans l'industrie. Cependant, mettre en œuvre le contrôleur PID est simple, mais pas le réglage. Le processus de réglage des paramètres PID (K_p , K_i et K_d) est un processus continu d'essais et d'erreurs. Il n'y a pas de méthode exacte pour calculer la valeur des paramètres à moins que l'ensemble du système ne soit mathématiquement modélisé et simulé. L'expérience est un facteur important pour obtenir les paramètres PID optimaux en fonction de l'observation du comportement du système pendant le processus de réglage.



A partir du schéma de principe du contrôleur PID, nous pouvons voir que la sortie de la boucle est simplement la somme de la sortie des contrôleurs P, I et D. Les équations pour la boucle PID sont illustrées ci-dessous:

Last Error = Error
Error = Set Point – Process Variable
Integral = Integral + Error
Derivative = Error – Last Error
Control Variable = $(K_p * Error) + (K_i * Integral) + (K_d * Derivative)$

PID en c code

```
while(1)
{
    // obtient la position actuelle
    current_position =read_current_position();
    // calcule l'erreur
    error=target_position-current_position;
    // calcule l'intégrale
    integral=integral+error;
    // calcule le dérivé
    derivative=error-last_error;
    // calcule la variable de contrôle
    pwm=(kp*error)+(ki*integral)+(kd*derivative);
    // limite la variable de contrôle à + -255
    if(pwm>255)
        pwm=255;
    else if(pwm<0)

        // si la variable de contrôle est positive, faire tourner le moteur positive
        if(pwm>0)
            motor_cw(pwm);
        // si la variable de contrôle est positive, faire tourner le moteur inverse
        else if(pwm<0)
            motor_ccw(-pwm);
        // si la variable de contrôle est zéro, arrêtez le moteur
        else motor_stop();

    // enregistre l'erreur actuelle comme dernière erreur pour l'itération suivante
    last_error=error;
}
```

CONCLUSION, BIBLIOGRAPHIES ET ANNEXES

CONCLUSION

À terme, nous avons vu, tout au long de ce projet, des possibilités viables pour la rénovation d'un robot PUMA 500. Nous avons étudié un système obsolète, émis un avis critique sur ce dernier et élaboré un tout nouveau système usant de technologies récentes pour ainsi remettre au goût du jour ce robot qu'est le PUMA 500.

Nous avons, dans un premier temps, étudié les différents moyens pour régler la question de l'alimentation des moteurs constituant l'automate d'où notre choix d'utiliser des cartes « driver » dont le rôle est de réguler l'apport énergétique des moteurs en fonction de la demande.

Dans un second temps, nous avons choisi le microcontrôleur le plus adéquat pour le pilotage de l'automate et avons confectionné une carte de contrôle munie de ce même microcontrôleur, le PIC32MK dont le rôle est d'interpréter les informations renvoyées par les encodeurs des servomoteurs ainsi que de piloter les cartes « drivers ». Étant donné que le PIC32MK est un microcontrôleur récemment mis sur le marché, de même que son outil logiciel de développement, Mplab Harmony, il nous a donc fallu, avant toute chose, nous familiariser avec ces outils pour pouvoir concrétiser le projet. Nous avons pu alors constater, durant l'expérimentation de cette partie du projet, que l'utilisation du PIC32MK fut une idée judicieuse mais représente aussi un certain challenge étant donnée la rareté actuelle de son utilisation dans le domaine public.

Une élaboration d'une manette de commande fut, ensuite, mise en œuvre pour communiquer avec la carte de contrôle et ainsi rendre possible le pilotage du robot avec précision et rapidité à l'aide d'un joystick et de boutons. Pour ce faire, nous avons opté pour une communication entre la manette et la carte de contrôle via un protocole dit « BUS CAN » qui assure la transmission des données de communication et préserve contre d'éventuelles pertes d'informations en court de route dues aux interférences. Le BUS CAN étant un protocole, initialement, plus complexe que nécessaire dans le cadre de ce projet, nous avons donc décidé de simplifier le protocole et de le personnaliser pour l'optimiser.

Pour conclure, nous avons étudié l'asservissement des moteurs, leurs comportements au freinage et à l'accélération pour ainsi, assurer la bonne trajectoire du bras mécanique. De même, nous avons mis en évidence les différentes équations et donc, les différentes positions respectives de chaque moteur dans le cas d'un pilotage général de l'automate dans un repère cartésien à trois dimensions.

En général, cette aventure nous a permis de bénéficier d'une première expérience pour mener à bien un projet tout en faisant face aux obstacles pouvant mettre à mal le bon déroulement des missions ainsi que le respect des délais. Cela nous aussi donné un premier aperçu du travail en équipe.

Bien évidemment, il reste encore fort à faire pour que le robot PUMA 500 soit pleinement opérationnel, voir même, pour qu'il puisse enfin rivaliser avec les bras mécaniques à la pointe de la technologie actuelle

BIBLIOGRAPHIES

PARTIE II : CARTE DE CONTRÔLE

[1] : Advanced Monolithic Systems, Inc. *Datasheet ASM1117*,

<http://www.advanced-monolithic.com/pdf/ds1117.pdf>, consulté le 18 février 2018

[2] : *Bihn Daniel et Hsia Steve. Universal Six-Joint Robot Controller. Février 1988*

[3] : *Corke Peter I. . THE UNIMATION PUMA SERVO SYSTEM. Juillet 1994. CSIRO Division of Manufacturing Technology Australia.*

[4] : Farooq M. , Wang Dao-bo. *Implementation of a new PC based controller for a PUMA robot.* 13 Juin 2007. Université aéronautique et astronautique d'ingénierie d'automatisation, Nanjing, Chine.

[5] : Jones Frederick J. . *Modelling, Simulation and Identification of an Industrial Manipulator.*. Septembre 1990. Université de Dublin, école d'ingénierie électronique, unité de recherche des technologies de contrôle, cursus master d'ingénieur.

[6] : Microchip Technology Inc. 2017. *Datasheet PIC32MK*

<http://ww1.microchip.com/downloads/en/DeviceDoc/60001402D.pdf> , consulté le 23 Mars 2018

[7] : Microchip Technology Inc. *MPLAB® Harmony Help,*

http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB%20Harmony%20Help_v202.pdf, consulté le 3 mai 2018

[8] : Oeuvre collective. *Interfacing a PUMA 500 Robot with a PC-based Controller, , 4 Février 2009. ECE4007 Senior Design Project, Section L01, Yellow PUMA Team*

[9] : *Rutherford Jerry. USING THE PUMA 560 ROBOT, 2012. PUMA Unofficial User's Guide*

[10] : Wyeth Grodon, James Kennedy et Jared Lillywhite. *Distributed Digital Control of a Robot Arm.* Université des sciences informatiques et de l'ingénierie électrique de Queensland, Australie

[11] : https://moodle.insa-toulouse.fr/pluginfile.php/88116/mod_resource/content/3/TP_ADC_v5.pdf consulté le 4 mai 2018

PARTIE III : ALIMENTATION, CARTE DRIVER ET MANETTE DE COMMANDE

Liens pour l'achat du driver : <https://www.gotronic.fr/art-commande-de-moteur-cc-12a-dri0042-25558.htm>

DELAUNAY, Eric.

« LE BUS CAN » http://meteosat.pessac.free.fr/Cd_elect/perso.libertysurf.fr/edelaunay/buscan.htm

Dominique, Jean-Luc. « Mise en œuvre du Bus CAN entre modules Arduino ». Le 2 novembre 2015. Adresse: <http://www.locoduino.org/spip.php?article130>

LERY, Charles. « MISE EN ŒUVRE D'UNE COMMUNICATION PAR BUS CAN ». [en ligne]. 2008/2009.

Adresse : https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwivkfqR0J7bAhUEQJoKHVwMD6MQFaggMAA&url=http%3A%2F%2Fdirac.epucfe.eu%2Fprojets%2Fwakka.php%3Fwiki%3DP08AB12index%2Fdownload%26file%3Dnote_application_com_pic_mppt.pdf&usg=AOvVaw2Tdm4U3p-9R3Wm8FMxQUXk.

Pont en H : ROUSSEAU, Jean-Noël, « Le moteur à courant continu (partie 2) : le pont en H et les circuits intégrés ». <https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-moteur-a-courant-continu-partie-2-le-pont-en-h-et-les-circuits-integres>.

Illustration 11 : Technologuepro, « Variateur de Vitesse Pont H max 10A ». 13 Février 2013.
<http://www.technologuepro.com/montages-electroniques/Pont-H-10A100V-20.html>

Illustration 13 : « Conversion Numérique/Analogique – PWM ».

<http://arduino.blaisepascal.fr/index.php/2015/07/29/conversion-numeriqueanalogique-pwm/>

Illustration 14 : « Commande d'un moteur à courant continu ». http://www.train35.fr/commande_moteur.html

Illustration 26 : DELAUNAY, Eric, « LE BUS CAN ». <http://edelaunay.chez-alice.fr/buscan.htm>

Illustration 27 : « LE BUS CAN 2,0B ».
<http://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/pedagogiques/494/494-3-buscan-20b.pdf>

Illustration 28 : « Cours Systèmes Embarqués:Le Bus CAN ».

<http://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-CAN.htm>

Illustration 29 : <https://fr.aliexpress.com/item/Free-shipping-MCP2515-CAN-Bus-Module-TJA1050-Receiver-SPI-Module-For-arduino/32709839532.html>

Illustration 30 : julianpe. « Arduino MCP2515 connecting to Volkswagen ».

<http://www.instructables.com/topics/Arduino-MCP2515-connecting-to-Volkswagen/>

PARET, Dominique. « Le Bus CAN Applications ». DUNOD, 1er avril 1998. 340 pages.

PARTIE IV : ASSERVISSEMENT ET PILOTAGE GÉNÉRAL

[1] : <http://www.professeurs.polymtl.ca/luc.baron/index.php?id=210&lg=f>

[2] : Hamdi H, (Nov 2012-2016), *Modélisation et commande*, cours ROBOTIQUE, université des frères mentouri, Constantine – ALGERIE

[3]: <http://petercorke.com/wordpress/>

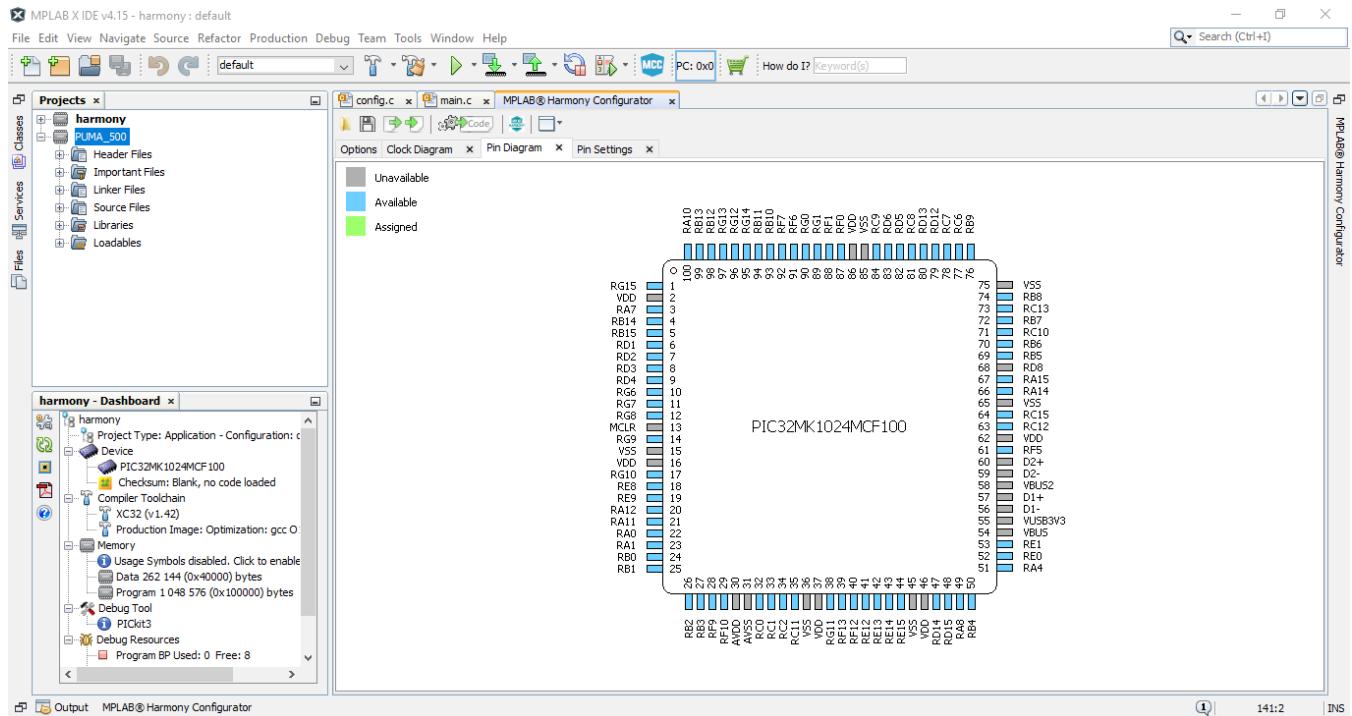
[4]: http://en.wikipedia.org/wiki/PID_controller

[5]: <https://tutorial.cytron.io/2012/06/22/pid-for-embedded-design/>

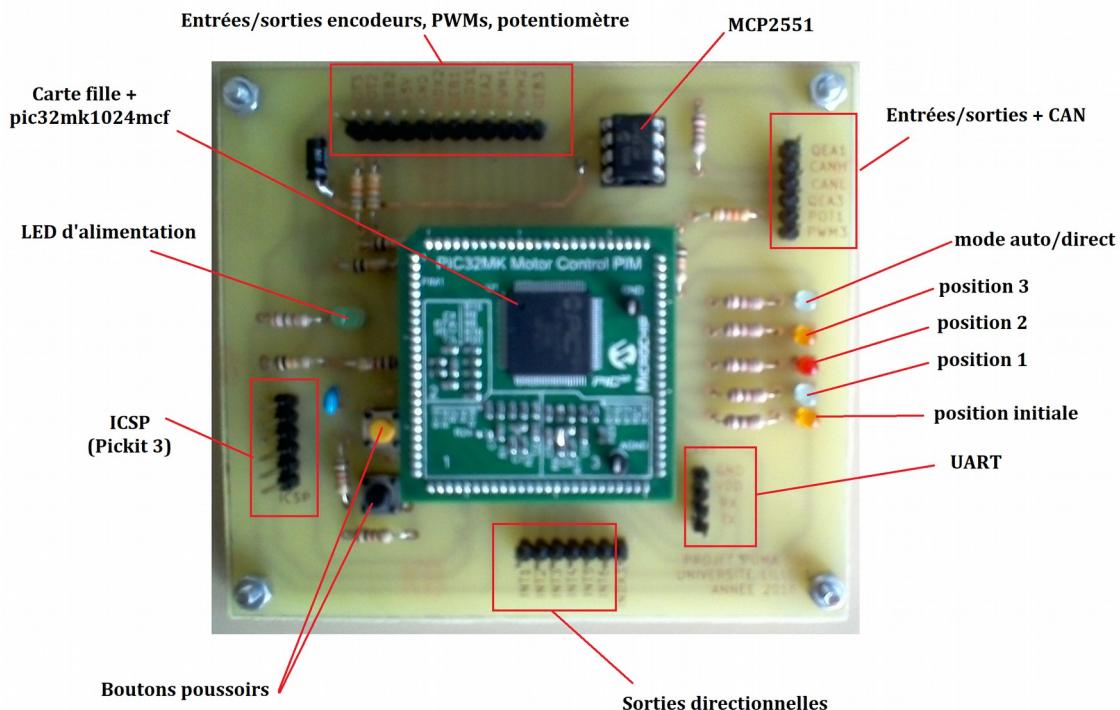
ANNEXES

PARTIE II : CARTE DE CONTRÔLE

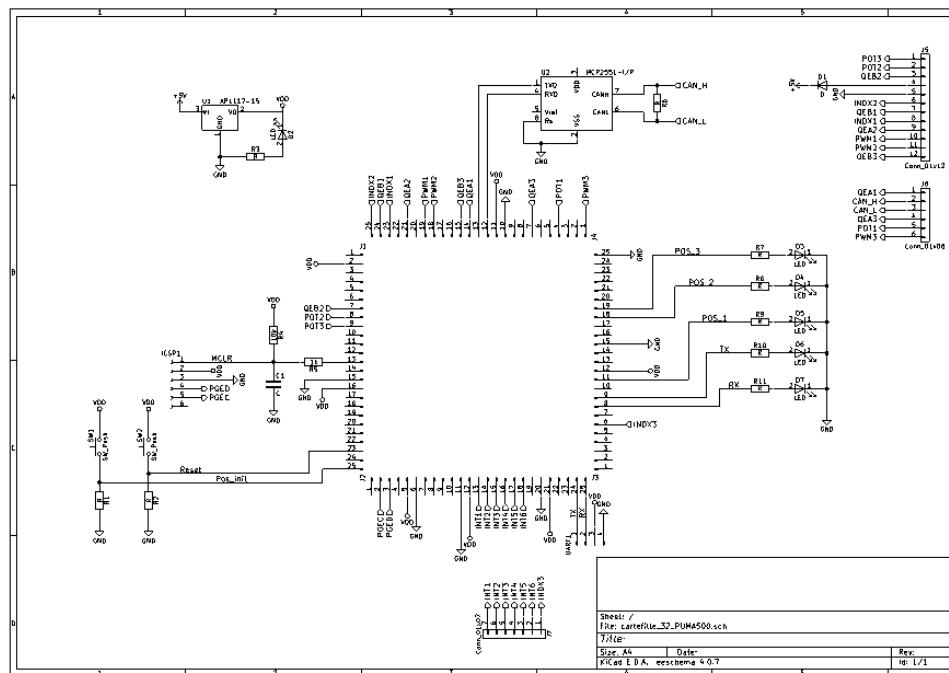
Annexe – Interface du firmware Mplab Harmony



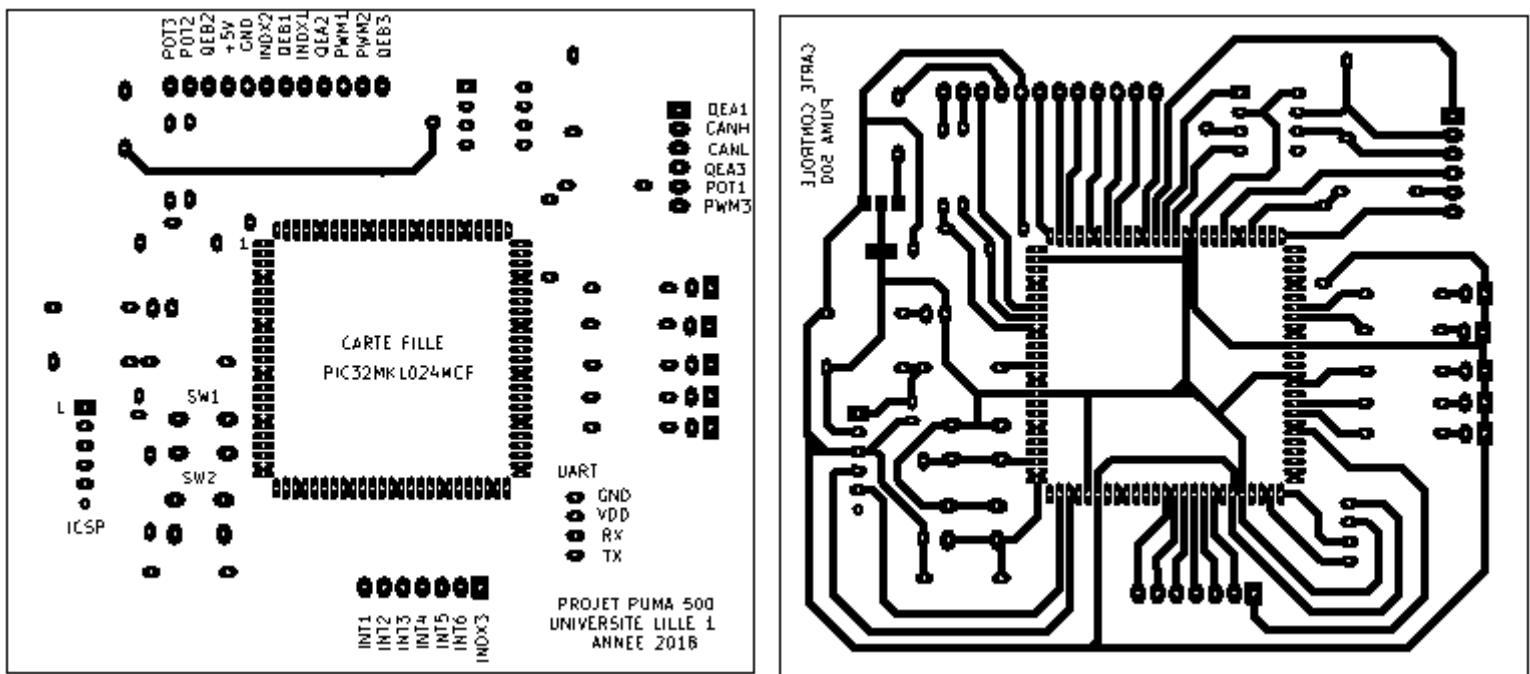
Annexe – Carte de contrôle avec carte fille



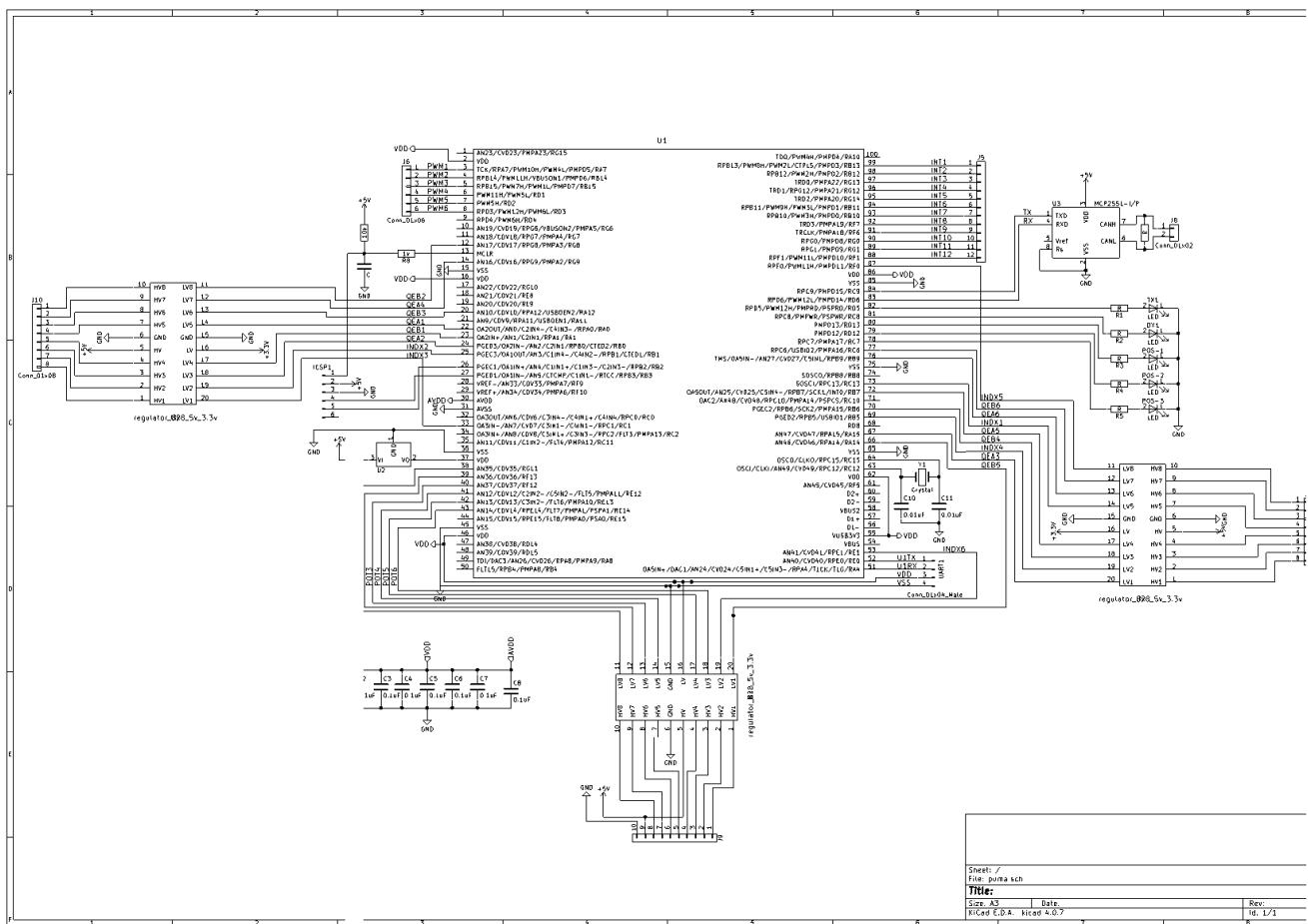
Annexe – schéma électrique de la carte de contrôle avec carte fille



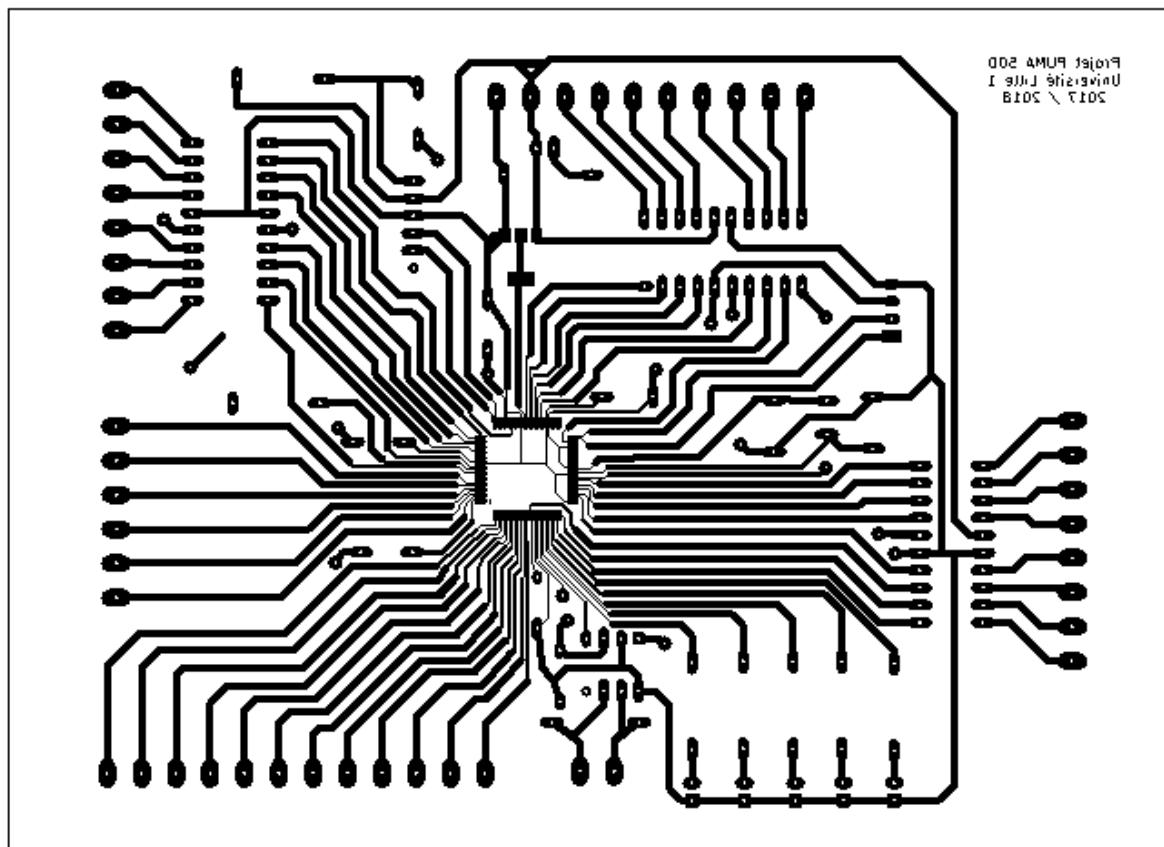
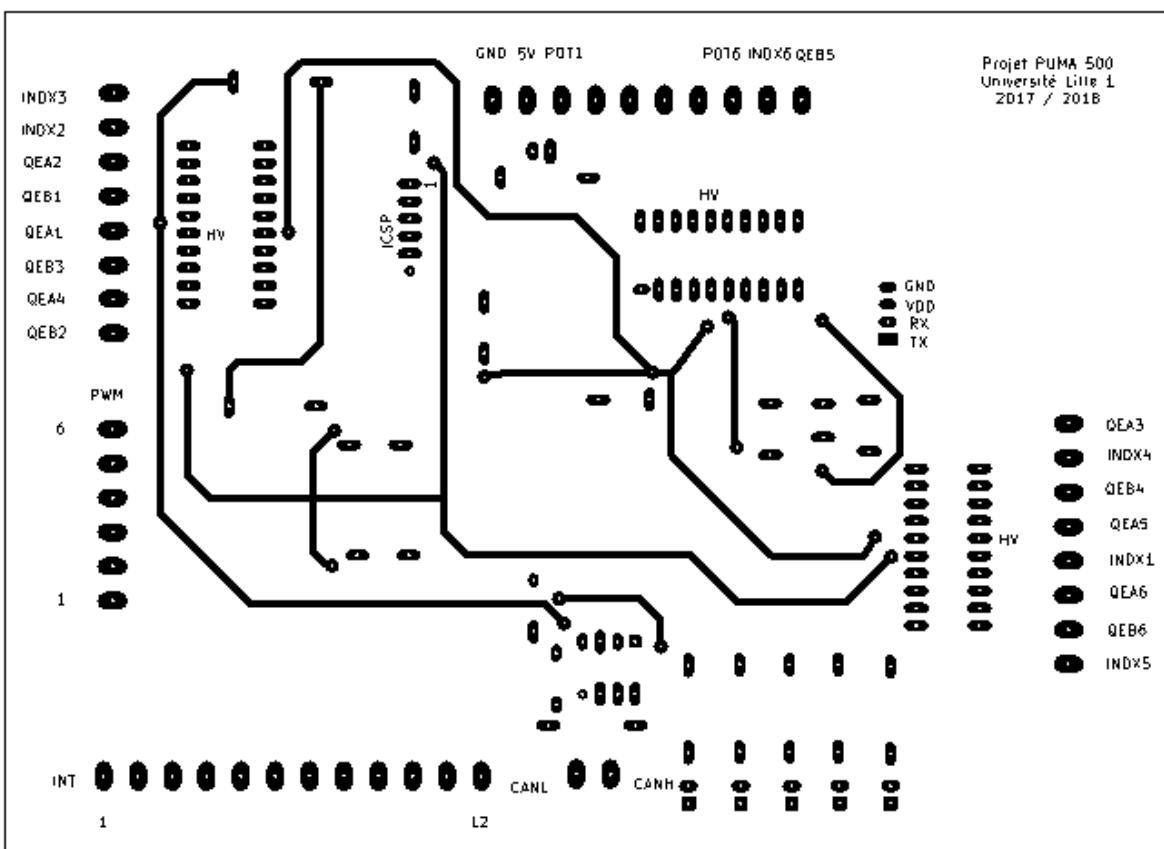
Annexe – Typon carte de contrôle avec carte fille (côté composant à gauche et côté cuivre à droite)



Annexe – schéma électrique de la carte de contrôle avec pic32mk1024mcf

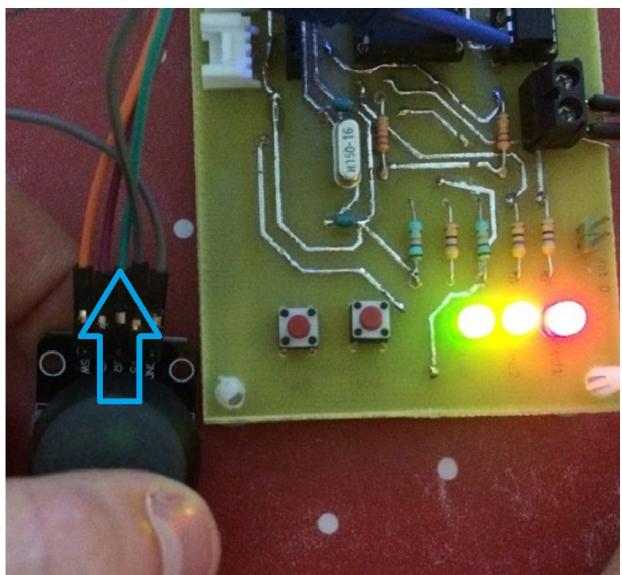
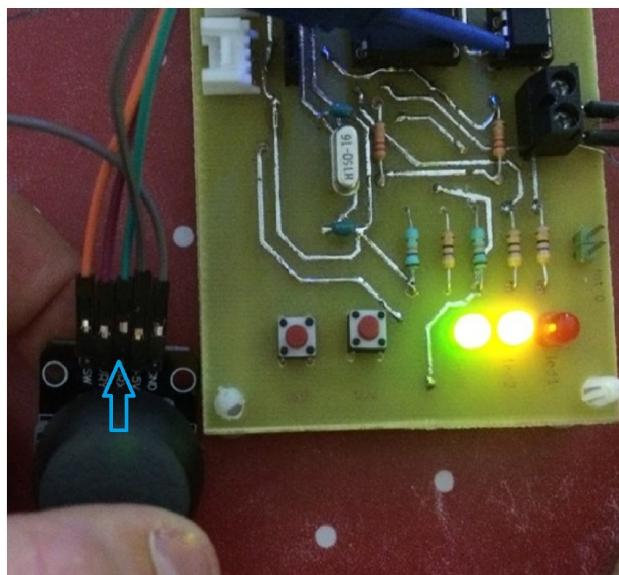
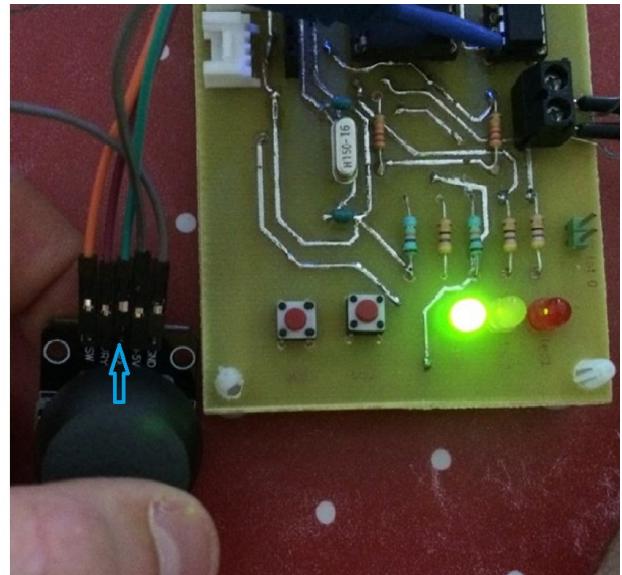


Annexe – Typon carte de contrôle avec pic32mk1024mcf (côté composant et côté cuivre)



PARTIE III : ALIMENTATION, CARTE DRIVER ET MANETTE DE COMMANDE

Tests du joystick sur un PIC18F458 :



Programme arduino pour le contrôle des deux moteurs via les signaux MLI:

```
//Déclaration des sorties moteur ( sorties PWM )
int sortie1 = 3;
int sortie2 = 5;

//Sorties qui permettent de contrôler le sens de rotation du moteur 1
byte SA0=14;
byte SA1=15;

//Sorties qui permettent de contrôler le sens de rotation du moteur 2
byte SA4=18;
byte SA5=19;

// Mémorisation des vitesses initialement demandées par l'utilisateur pour réaliser le changement de sens de rotation
int pwm1;
int pwm2;

void setup()
{
    Serial.begin(9600);

    //On met en sortie les broches pour le PWM
    pinMode(3, OUTPUT);
    pinMode(5,OUTPUT);

    //On met en sortie les broches pour gérer le sens de rotation des moteurs
    pinMode(SA0,OUTPUT);
    pinMode(SA1,OUTPUT);
    pinMode(SA4,OUTPUT);
    pinMode(SA5,OUTPUT);

    // On défini le sens de rotation de départ pour les deux moteurs
    digitalWrite(SA0,LOW);
    digitalWrite(SA1,HIGH);
    digitalWrite(SA4,LOW);
    digitalWrite(SA5,HIGH);
```

```

//On initialise les PWM à zéro pourcent
analogWrite(sortie1, 0); //initialisation du PWM du driver 1 à zéro
analogWrite(sortie2, 0); //initialisation du PWM du driver 2 à zéro

//Présentation des différents branchements à réaliser avant la manipulation
Serial.println("/// BRANCHEMENTS : /// \n");
Serial.println("/// DRIVER 1 :");
Serial.println("// IN1 sur A1");
Serial.println("// IN2 sur A0");
Serial.println("//PWM sur la broche 5");

Serial.println("\n");

Serial.println("/// DRIVER 2 :");
Serial.println("// IN1 sur A5");
Serial.println("// IN2 sur A4");
Serial.println("//PWM sur la broche 3");

Serial.println(" fin des branchements \n");

///////////
//////AFFICHAGE TABLE DES FONCTIONS /////
/////////

// On affiche ici toutes les commandes que l'on peut envoyer pour contrôler les deux moteurs
Serial.println(" //////////////////////////////\n ////////////////////////////// TABLE DES\n FONCTIONS //////////////////////////////\n ////////////////////////////// ");
Serial.println("Entrez les valeurs suivantes pour modifier la vitesse du moteur 1 en sortie du driver 1:");
Serial.println("\n 1 pour 0 % \n 2 pour 25% \n 3 pour 50 % \n 4 pour 75 % \n 5 pour 100 % \n");

Serial.println("Entrez les valeurs suivantes pour modifier la vitesse du moteur 2 en sortie du driver 2:");
Serial.println("\n A pour 0 % \n Z pour 25% \n E pour 50 % \n R pour 75 % \n T pour 100 % \n");

Serial.println("Entrez les valeurs suivante pour modifier le sens de rotation des moteurs");
Serial.println(" ( ( !!! Attention, la modification du sens du moteur arrête automatiquement le moteur en question !!! ) ) ");
Serial.println("Entrez Q pour modifier le sens de rotation du moteur 1");
Serial.println("Entrez S pour modifier le sens de rotation du moteur 2");

Serial.println("\n ");

```

```

Serial.println("///////////");
}

void loop()
{
    /////////////
    // configuration du premier moteur: sortie1 ///
    /////////////

    while(Serial.available()>0)
    {
        char c = Serial.read();

        //Si le caractère envoyé de l'ordinateur est 1, on met la sortie du PWM du moteur 1 à 0 pour mettre le PWM à 0%
        if(c=='1')
        {
            analogWrite(sortie1, 0);
            Serial.println("PWM sortie 1 a 0%"); // pas de caractères accentués !!
            pwm1=0;
        }

        //Si le caractère envoyé de l'ordinateur est 2, on met la sortie du PWM du moteur 1 à 63 pour mettre le PWM à 0%
        if(c=='2')
        {
            analogWrite(sortie1, 63);
            Serial.println("PWM sortie 1 a 25%"); // pas de caractères accentués !!
            pwm1=63;
        }

        if(c=='3')
        {
            analogWrite(sortie1, 127);
            Serial.println("PWM sortie 1 a 50%"); // pas de caractères accentués !!
            pwm1=127;
        }

        if(c=='4')
        {
    
```

```

analogWrite(sortie1, 190);
Serial.println("PWM sortie 1 a 75%"); // pas de caractères accentues !!
pwm1=190;
}
if(c=='5')
{
analogWrite(sortie1, 255);
Serial.println("PWM sortie 1 a 100%"); // pas de caractères accentues !!
pwm1=255;
}

///////////////////////////////
// configuration du second moteur: sortie2 ///
///////////////////////////////

//Si le caractère envoyé de l'ordinateur est A, on met la sortie du PWM du moteur 2 à 0 pour mettre le PWM à 0%
if(c=='A')
{
analogWrite(sortie2, 0);
Serial.println("PWM sortie 2 a 0%"); // pas de caractères accentues !!
pwm2=0;
}

//Si le caractère envoyé de l'ordinateur est Z, on met la sortie du PWM du moteur 2 à 63 pour mettre le PWM à 25%
if(c=='Z')
{
analogWrite(sortie2, 63);
Serial.println("PWM sortie 2 a 25%"); // pas de caractères accentues !!
pwm2=63;
}

if(c=='E')
{
analogWrite(sortie2, 127);
Serial.println("PWM sortie 2 a 50%"); // pas de caractères accentues !!
pwm2=127;
}

if(c=='R')
{

```

```

analogWrite(sortie2, 190);
Serial.println("PWM sortie 2 a 75%"); // pas de caractères accentues !!
pwm2=190;
}

if(c=='T')
{
analogWrite(sortie2, 255);
Serial.println("PWM sortie 2 a 100%"); // pas de caractères accentues !!
pwm2=255;
}

///////////////////////////////
/// dispositif de changement de sens des moteurs ///
///////////////////////////////

//Moteur 1 avec IN1 et IN2 à brancher sur A0 et A1
// Si le caractère envoyé sur la liaison série de l'arduino est Q
if(c=='Q')
{
//on met le pwm du moteur à zéro
analogWrite(sortie1, 0);
//on inverse les valeurs de IN1 et IN2 pour modifier le sens de rotation du moteur 1
digitalWrite(SA4, !digitalRead(SA4));
digitalWrite(SA5, !digitalRead(SA5));

Serial.println("sens de rotation modifie pour le moteur 1"); // pas de caractères accentues !!

// on attends un peu afin que l'arrêt du moteur total
//( l'inertie de l'arbre de rotation ne permet pas au moteur de s'arrêter immédiatement )
delay(300);

//on relance le moteur à la même vitesse que précédemment
analogWrite(sortie1,pwm1);
}

//Moteur 2 avec IN1 et IN2 à brancher sur A4 et A5
if(c=='S')

```

```

{

//on met le pwm du moteur à zéro
analogWrite(sortie2, 0);

//on inverse les valeurs de IN1 et IN2 pour modifier le sens de rotation du moteur 2
digitalWrite(SA0, !digitalRead(SA0));
digitalWrite(SA1, !digitalRead(SA1));

Serial.println("sens de rotation modifie pour le moteur 2"); // pas de caractères accentues !!

// on attend un peu afin que l'arrêt du moteur total
//( l'inertie de l'arbre de rotation ne permet pas au moteur de s'arrêter immédiatement )
delay(300);

// on attend l'arrêt total du moteur
analogWrite(sortie2,pwm2);
}

}

}

```

Programme du CANBUS microcontrôleur PIC :

```

/*
 * File: main.c
 * Author: marc
 *
 * Created on 22 mars 2018, 19:08
 */

```

```
// PIC18F458 Configuration Bit Settings
```

```
// 'C' source line config statements
```

```
#include <p18F458.h>
```

```

#include "ecan.h"
#include "CAN.H"

//CONFIGURATION GENERAL DU MICROCONTROLEUR

// CONFIG1H
#pragma config OSC = HS      // Oscillator Selection bits (HS oscillator)
#pragma config OSCS = OFF    // Oscillator System Clock Switch Enable bit (Oscillator system clock switch option is disabled (main oscillator is source))

// CONFIG2L
#pragma config PWRT = ON     // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOR = ON      // Brown-out Reset Enable bit (Brown-out Reset enabled)
#pragma config BORV = 27     // Brown-out Reset Voltage bits (VBOR set to 2.7V)

// CONFIG2H
#pragma config WDT = OFF     // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))
#pragma config WDTPS = 128    // Watchdog Timer Postscale Select bits (1:128)

// CONFIG4L
#pragma config STVR = ON     // Stack Full/Underflow Reset Enable bit (Stack Full/Underflow will cause Reset)
#pragma config LVP = ON       // Low-Voltage ICSP Enable bit (Low-Voltage ICSP enabled)

// CONFIG5L
#pragma config CP0 = OFF     // Code Protection bit (Block 0 (000200-001FFFh) not code protected)
#pragma config CP1 = OFF     // Code Protection bit (Block 1 (002000-003FFFh) not code protected)
#pragma config CP2 = OFF     // Code Protection bit (Block 2 (004000-005FFFh) not code protected)
#pragma config CP3 = OFF     // Code Protection bit (Block 3 (006000-007FFFh) not code protected)

// CONFIG5H
#pragma config CPB = OFF     // Boot Block Code Protection bit (Boot Block (000000-0001FFh) not code protected)
#pragma config CPD = OFF     // Data EEPROM Code Protection bit (Data EEPROM not code protected)

// CONFIG6L
#pragma config WRT0 = OFF    // Write Protection bit (Block 0 (000200-001FFFh) not write protected)
#pragma config WRT1 = OFF    // Write Protection bit (Block 1 (002000-003FFFh) not write protected)
#pragma config WRT2 = OFF    // Write Protection bit (Block 2 (004000-005FFFh) not write protected)

```

```

#pragma config WRT3 = OFF // Write Protection bit (Block 3 (006000-007FFFh) not write protected)

// CONFIG6H

#pragma config WRTC = OFF // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write
protected)

#pragma config WRTB = OFF // Boot Block Write Protection bit (Boot Block (000000-0001FFh) not write protected)

#pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM not write protected)

// CONFIG7L

#pragma config EBTR0 = OFF // Table Read Protection bit (Block 0 (000200-001FFFh) not protected from Table Reads executed in other
blocks)

#pragma config EBTR1 = OFF // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from Table Reads executed in other
blocks)

#pragma config EBTR2 = OFF // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from Table Reads executed in other
blocks)

#pragma config EBTR3 = OFF // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from Table Reads executed in other
blocks)

// CONFIG7H

#pragma config EBTRB = OFF // Boot Block Table Read Protection bit (Boot Block (000000-0001FFh) not protected from Table Reads
executed in other blocks)

```

//DECLARATION DES FONCTIONS

void EnvoieCanBus(void); //déclaration de la fonction d'envoie sur le canbus

void RecoitCanBus(void); // déclaration de la fonction de réception du canbus

```

int r;

void initADC();

void getADC(int pos);

char buffCANBUS[8] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x00, 0x00, 0x00}; //données à envoyer 8 octets au maximum

```

```

void main(void)
{

```

```

    int val;
    int resx;

```

```

int resy;

TRISDbits.TRISD0 = 0; // en sortie
TRISDbits.TRISD1 = 0; // en sortie
TRISDbits.TRISD2 = 0; // en sortie
TRISAbits.TRISA0 = 1; // en entree
TRISAbits.TRISA1 = 1; //en entree
TRISAbits.TRISA4 = 1; // le bouton du joystick
LATDbits.LATD0 = 0; // allumer la led rouge
LATDbits.LATD1 = 0; // allumer la led jaune
LATDbits.LATD2 = 0; // allumer la led verte

initADC(); // on initialise le convertisseur analogique numerique

while(1)
{
    getADC(0); // on lit la valeur convertie de l'axe x
    //on stocke la valeur de x convertie. On va chercher directement dans les registre du convertisseur Analogique Numerique
    buffCANBUS[0] = ADRESH;
    buffCANBUS[1] = ADRESL;

    getADC(1); // on lit la valeur convertie de l'axe y
    //on stocke la valeur de y convertie. On va chercher directement dans les registre du convertisseur Analogique Numerique
    buffCANBUS[2] = ADRESH;
    buffCANBUS[3] = ADRESL;

    //on stocke la valeur du bouton poussoir
    buffCANBUS[4] = !PORTAbits.RA4; // Valeur du bouton poussoir du joystick

    EnvoieCanBus(); // appel de la fonction qui envoi les nouvelles valeurs sur le bus
}

void initADC()
{

```

```

//paramètre convertisseur analogique numérique

ADCON0 = 0;
ADCON1 = 0;

ADCON0bits.ADCS1 = 1; // on ajuste le temps de conversion par rapport a la vitesse du processeur

ADCON1bits.PCFG = 0x0100; // an0 et an1 en analogique

ADCON0bits.ADON = 1; // on active le convertisseur

}

//Fonction qui reçoit 0 ou 1 en fonction de la voie à tester (X ou Y)

void getADC(int pos)

{
    unsigned int result;

    ADCON0bits.ADON = 0; // on desactive le convertisseur

    ADCON0bits.CHS = pos;//on recupere la valeur de l'appuie bouton

    ADCON0bits.ADON = 1; // on active le convertisseur

    for(result = 0; result < 1000; result++)

    {
        //On laisse au minimum un cycle de conversion

        Nop(); // indication de la datasheet, il ne faut pas faire l'activation du convertisseur et la lecture à la suite

    }

    ADCON0bits.GO = 1;// On lance la conversion

    while(ADCON0bits.DONE == 1); // On attend que le bit DONE passe à 1, cela indique la fin de la conversion
}

void EnvoieCanBus(void)

{
    unsigned int txCID = 0x00;

    CANInit(); // initialisation du bus can

    CANSetBitRate(2); // communication à 250Kb/s

    CANOpen(); // ouverture du CANBUS
}

```

```

while(1)
{
    if(CANIsOpen() == TRUE)
    {
        if(CANIsTxRdy() == TRUE)
        {
            /*
            CANPutTxDataTyp0("marc");
            CANPutTxCID(1);
            CANPutTxCnt(5);
            CANSend(6);
            CANSend(txt);

            LATDbits.LATD1 ^= 1;
            LATDbits.LATD2 ^= 1;
            delay(500);
            */

            LATDbits.LATD2 = 1;// on allume la led verte
            // LATDbits.LATD1 = 0;//on eteint la led jaune
            // delay(100000);
            // LATDbits.LATD1 = 1; //on allume la led jaune
            CANPutTxCnt(8);//On envoi la taille du contenu à envoyer
            CANPutTxCID(txCID);//On envoie l'identificateur
            CANPutTxDataTyp0(buffCANBUS);//on envoie les données du joystick
            //CANSend(8);
            //CANPutTxDataTyp0(txt);
            //CANSend(txt);
            //LATDbits.LATD1 = 1; //on allume la led jaune
        }
    }
}

//CANClose(); // fermeture du bus CAN
}

}

/*
void RecoitCanBus(void)

```

```

{

    unsigned int ID;
    while(1)
    {
        CANInit();// initialisation du bus can
        CANOpen();//ouverture du CANBUS

        if(CANIsOpen()== TRUE)
        {
            CANSetBitRate(1); //communication à 250Kb/s
            LATDbits.LATD2 = 1;// on allume la led verte
            CANRead();

            if(CANIsRxRdy() == TRUE)
            {
                ID=CANGetRxCID();
                LATDbits.LATD1 = 1; //on allume la led jaune
            }
        }

    }

}

*/
/*
void delay(int val) // fonction d'attente en microseconde environ
{
    int i, x;

    for(i = 0; i < val; i++)
    {
        for(x = 0; x < 125; x++)
            Nop();
    }
}

```

```
 }  
}  
*/
```

Programme Arduino qui affiche les valeurs de la position x et y d'un joystick ainsi que son appui bouton :

```
/*  
 Utilisation Joystick  
 */  
  
int x = A0; // joystick mouvement vertical  
int y = A1; // joystick mouvement horizontal  
int valeur_x = 0;  
int valeur_y = 0;  
  
byte bouton = 3;  
  
void setup()  
{  
    Serial.begin(9600);  
    pinMode(x, INPUT);  
    pinMode(y, INPUT);  
    pinMode(bouton, INPUT_PULLUP);  
}  
  
void loop()  
{  
    // Lit l'entrée analogique  
    valeur_x=analogRead(x);  
    valeur_y=analogRead(y);  
  
    Serial.print("X= ");  
    Serial.print(valeur_x);  
    Serial.print(" Y= ");  
    Serial.println(valeur_y);  
  
    // Serial.println(digitalRead(bouton)); // apuye=0 non appuye=1
```

```

if(digitalRead(bouton) == LOW)
{
    Serial.println("bouton appuye");
}
else
{
    Serial.println("bouton non appuye");
}

delay(500);
}

```

Programme pour envoyer des données sur le réseau CAN BUS via Arduino :

```

// demo: CAN-BUS Shield, send data

#include <mcp_can.h>
#include <SPI.h>

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
const int ledHIGH  = 1;
const int ledLOW   = 0;

MCP_CAN CAN(SPI_CS_PIN);           // Set CS pin

void setup()
{
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_250KBPS))      // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");
        delay(100);
    }
}

```

```

Serial.println("CAN BUS Shield init ok!");

}

unsigned char stmp[8] = {0, 1, 2, 3, 4, 5, 6, 7};

void loop()
{
  Serial.println("In loop");

  // send data: id = 0x00, standard frame, data len = 8, stmp: data buf
  CAN.sendMsgBuf(0x70, 0, 8, stmp);

  delay(1000);          // send data once per second
}

//********************************************************************

END FILE
//********************************************************************/

```

Programme pour recevoir des données sur le réseau CAN BUS Arduino :

```

// demo: CAN-BUS Shield, receive data with check mode

// send data coming to fast, such as less than 10ms, you can use this way

// loovee, 2014-6-13

```

```

#include <SPI.h>
#include "mcp_can.h"

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 10;
const int LED      = 8;
boolean ledON     = 1;

MCP_CAN CAN(SPI_CS_PIN);           // Set CS pin

void setup()
{
  Serial.begin(115200);
  pinMode(LED,OUTPUT);

```

```

while (CAN_OK != CAN.begin(CAN_250KBPS))      // init can bus : baudrate = 500k
{
    Serial.println("CAN BUS Shield init fail");
    Serial.println("Init CAN BUS Shield again");
    delay(100);
}

Serial.println("CAN BUS Shield init ok!");

}

void loop()
{
    unsigned char len = 0;
    unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive())      // check if data coming
    {
        Serial.println("----- DONNEE ARRIVE -----");
        CAN.readMsgBuf(&len, buf);  // read data, len: data length, buf: data buf

        unsigned char canId = CAN.getCanId();

        Serial.println("-----");
        Serial.println("get data from ID: ");
        Serial.println(canId);

        for(int i = 0; i<len; i++) // print the data
        {
            Serial.print(buf[i]);
            Serial.print("\t");
            if(ledON && i==0)
            {

                digitalWrite(LED, buf[i]);
                ledON = 0;
                delay(500);
            }
            else if((!ledON) && i==4)
        }
    }
}

```

```

    {
        digitalWrite(LED, buf[i]);
        ledON = 1;
    }
}

Serial.println();
}

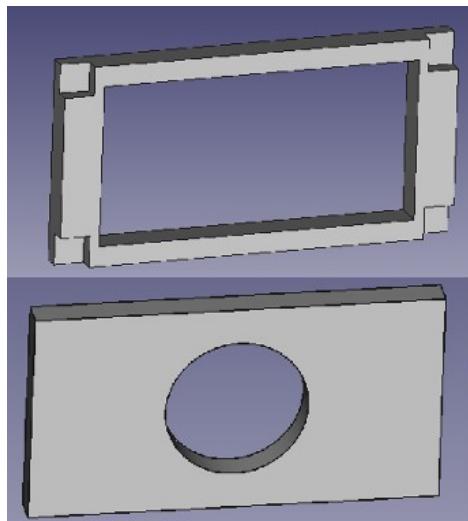
}

//END FILE

```

Conception d'une pièce 3D :

Nous avons réalisés une pièce 3D permettant de cacher les câbles accessibles et sans protection au niveau de la prise principale située sur la base du robot. Cette pièce sera réalisée par impression 3D. Les pièces représentées ci-dessous seront collées entre elles afin de former la pièce finale.



Modélisation via logiciel 3D
de la pièce cache-câbles

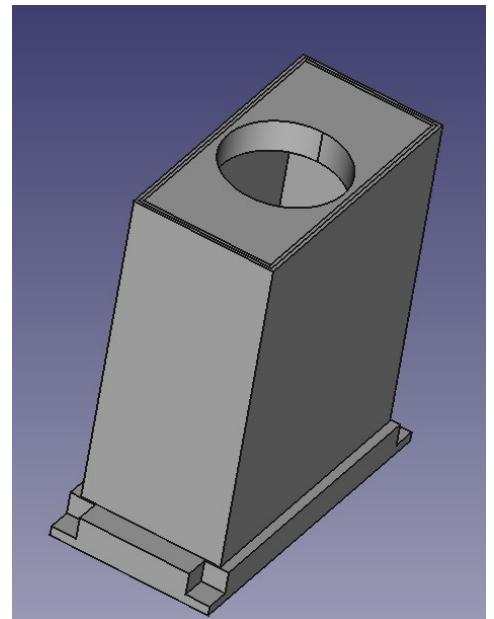
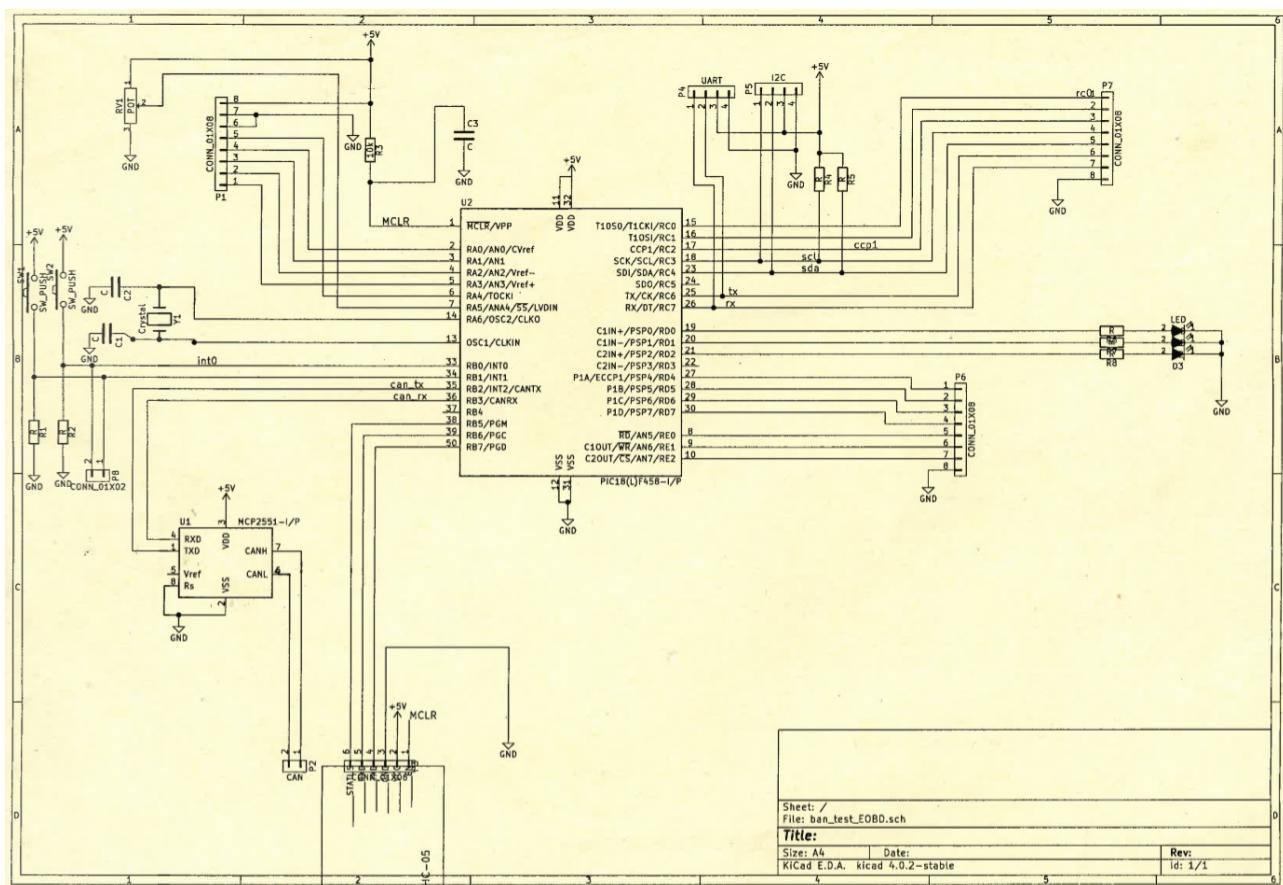


Schéma de la carte support du PIC18F458 réalisé par les techniciens :



PARTIE IV : ASSERVISSEMENT

ET PILOTAGE GÉNÉRAL

Annexe – Matlab

```
% ici c'est la fonction de la mod" "le g" ;om" ;trique inverse
% Cette fonction va convertir une position cart" ;ienne de l'effecteur terminal en
% un ensemble de variables communes de PUMA560 avec une table cin" ;matique connue et fixe

function joint_variables = inverse_Kinematics(cartesian_position_DH)

% =====Construction de la table cin" ;matique pour Puma560
% Table cin" ;matique pour Puma560
% d a alpha theta
KTable_PUMA560 = [ 0 0 -pi/2 0;
0 0.432 0 0;
0.149 -0.02 pi/2 0;
0.433 0 -pi/2 0;
0 0 pi/2 0;
0 0 0 0];
d3 = 0.149;
% d4 = 0.433;
% a2 = 0.432;
% a3 = -0.02;

% =====Calculer A1 "¤ A6 de PUMA560=====
A =zeros(4,4,6);
T =zeros(4,4,6);
for i = 1:1:6
A(:,:,i) = [cos(KTable_PUMA560(i,4)) -sin(KTable_PUMA560(i,4))*cos(KTable_PUMA560(i,3))
sin(KTable_PUMA560(i,4))*sin(KTable_PUMA560(i,3)) KTable_PUMA560(i,2)*cos(KTable_PUMA560(i,4));
sin(KTable_PUMA560(i,4)) cos(KTable_PUMA560(i,4))*cos(KTable_PUMA560(i,3)) -cos(KTable_PUMA560(i,4))*sin(KTable_PUMA560(i,3))
KTable_PUMA560(i,2)*sin(KTable_PUMA560(i,4));
0 sin(KTable_PUMA560(i,3)) cos(KTable_PUMA560(i,3)) KTable_PUMA560(i,1);
0 0 0 1];
for j = 1:i
if j == 1
T(:,:,:i) = A(:,:,:1);
end
if j>1
T(:,:,:i) = T(:,:,:i)*A(:,:,:j);
end
end
end

T6 = cartesian_position_DH;

px = T6(1,4);
py = T6(2,4);
pz = T6(3,4);

% T6_ref4 = A(:,:,5)*A(:,:,6);
% T6_ref3 = A(:,:,4)*A(:,:,5)*A(:,:,6);

% ===== Slove theta1 "¤ theta3=====
% Theta1 "¤ Theta3 est sur le point de fin effecteur du robot

%pour r" ;soudre deux configurations de theta1

theta1_1 = (atan2(T6(2,4),T6(1,4))-atan2(d3,(T6(1,4)^2+T6(2,4)^2-d3^2)^2^(1/2)));
theta1_2 = (atan2(T6(2,4),T6(1,4))-atan2(d3,-(T6(1,4)^2+T6(2,4)^2-d3^2)^2^(1/2)));

d3 = 0.149;
d4 = 0.433;
a2 = 0.432;
a3 = -0.02;
M = (px^2 + py^2 + pz^2 - a2^2 - a3^2 - d3^2 - d4^2) / (2*a2);
inner_sqrt = sqrt(a3^2+d4^2-M^2);
theta3_1 = atan2( M, inner_sqrt)- atan2(a3,d4) ;
theta3_2 = atan2( M, (-1)*inner_sqrt) - atan2(a3,d4);
```

```

%pour r";soudre quatre configurations de theta2
theta23_1 = atan2( d4+a2*sin(theta3_1) , a3 + a2*cos(theta3_1) ) - atan2 (pz, cos(theta1_1)*px + sin(theta1_1)*py );
theta2_1 = theta23_1 - theta3_1;

theta23_2 = atan2( d4+a2*sin(theta3_2) , a3 + a2*cos(theta3_2) ) - atan2 (pz, cos(theta1_1)*px + sin(theta1_1)*py );
theta2_2 = theta23_2 - theta3_2;

theta23_3 = atan2( d4+a2*sin(theta3_1) , a3 + a2*cos(theta3_1) ) - atan2 (pz, cos(theta1_2)*px + sin(theta1_2)*py );
theta2_3 = theta23_3 - theta3_1;

theta23_4 = atan2( d4+a2*sin(theta3_2) , a3 + a2*cos(theta3_2) ) - atan2 (pz, cos(theta1_2)*px + sin(theta1_2)*py );
theta2_4 = theta23_4 - theta3_2;

% ===== r";soudre th";ta 4 "a th";ta 6, pour les combinaisons de th";ta1 "a theta3 =====%
% ===== Calculer la dynamique pour chaque combinaison de th";ta1,2,3 ====
% lors de la r";solution de th";ta 4,5,6, toutes les combinaisons de th";ta 1,2,3 doivent "tre appliqu";es "a A1, A2, A3
%
A11 = [cos(theta1_1) -sin(theta1_1)*cos(KTable_PUMA560(1,3)) sin(theta1_1)*sin(KTable_PUMA560(1,3))
KTable_PUMA560(1,2)*cos(theta1_1);
sin(theta1_1) cos(theta1_1)*cos(KTable_PUMA560(1,3)) -cos(theta1_1)*sin(KTable_PUMA560(1,3))
KTable_PUMA560(1,2)*sin(theta1_1);
0 sin(KTable_PUMA560(1,3)) cos(KTable_PUMA560(1,3)) KTable_PUMA560(1,1);
0 0 0 1];
A12 = [cos(theta1_2) -sin(theta1_2)*cos(KTable_PUMA560(1,3)) sin(theta1_2)*sin(KTable_PUMA560(1,3))
KTable_PUMA560(1,2)*cos(theta1_2);
sin(theta1_2) cos(theta1_2)*cos(KTable_PUMA560(1,3)) -cos(theta1_2)*sin(KTable_PUMA560(1,3))
KTable_PUMA560(1,2)*sin(theta1_2);
0 sin(KTable_PUMA560(1,3)) cos(KTable_PUMA560(1,3)) KTable_PUMA560(1,1);
0 0 0 1];
A21 = [cos(theta2_1) -sin(theta2_1)*cos(KTable_PUMA560(2,3)) sin(theta2_1)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*cos(theta2_1);
sin(theta2_1) cos(theta2_1)*cos(KTable_PUMA560(2,3)) -cos(theta2_1)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*sin(theta2_1);
0 sin(KTable_PUMA560(2,3)) cos(KTable_PUMA560(2,3)) KTable_PUMA560(2,1);
0 0 0 1];
A22 = [cos(theta2_2) -sin(theta2_2)*cos(KTable_PUMA560(2,3)) sin(theta2_2)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*cos(theta2_2);
sin(theta2_2) cos(theta2_2)*cos(KTable_PUMA560(2,3)) -cos(theta2_2)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*sin(theta2_2);
0 sin(KTable_PUMA560(2,3)) cos(KTable_PUMA560(2,3)) KTable_PUMA560(2,1);
0 0 0 1];
A23 = [cos(theta2_3) -sin(theta2_3)*cos(KTable_PUMA560(2,3)) sin(theta2_3)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*cos(theta2_3);
sin(theta2_3) cos(theta2_3)*cos(KTable_PUMA560(2,3)) -cos(theta2_3)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*sin(theta2_3);
0 sin(KTable_PUMA560(2,3)) cos(KTable_PUMA560(2,3)) KTable_PUMA560(2,1);
0 0 0 1];
A24 = [cos(theta2_4) -sin(theta2_4)*cos(KTable_PUMA560(2,3)) sin(theta2_4)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*cos(theta2_4);
sin(theta2_4) cos(theta2_4)*cos(KTable_PUMA560(2,3)) -cos(theta2_4)*sin(KTable_PUMA560(2,3))
KTable_PUMA560(2,2)*sin(theta2_4);
0 sin(KTable_PUMA560(2,3)) cos(KTable_PUMA560(2,3)) KTable_PUMA560(2,1);
0 0 0 1];
A31 = [cos(theta3_1) -sin(theta3_1)*cos(KTable_PUMA560(3,3)) sin(theta3_1)*sin(KTable_PUMA560(3,3))
KTable_PUMA560(3,2)*cos(theta3_1);
sin(theta3_1) cos(theta3_1)*cos(KTable_PUMA560(3,3)) -cos(theta3_1)*sin(KTable_PUMA560(3,3))
KTable_PUMA560(3,2)*sin(theta3_1);
0 sin(KTable_PUMA560(3,3)) cos(KTable_PUMA560(3,3)) KTable_PUMA560(3,1);
0 0 0 1];
A32 = [cos(theta3_2) -sin(theta3_2)*cos(KTable_PUMA560(3,3)) sin(theta3_2)*sin(KTable_PUMA560(3,3))
KTable_PUMA560(3,2)*cos(theta3_2);
sin(theta3_2) cos(theta3_2)*cos(KTable_PUMA560(3,3)) -cos(theta3_2)*sin(KTable_PUMA560(3,3))
KTable_PUMA560(3,2)*sin(theta3_2);
0 sin(KTable_PUMA560(3,3)) cos(KTable_PUMA560(3,3)) KTable_PUMA560(3,1);
0 0 0 1];

% =====R";soudre l'articulation du poignet de l'articulation 4,5,6=====
% R";f";rence de la r";solution des angles d'Euler
%
EulerAngle_Invk = zeros(8,3);
T6_ref3_array=zeros(4,4,4);
T6_ref3_array(:,:,1) = inv(A31)*inv(A21)*inv(A11)*T6;
T6_ref3_array(:,:,2) = inv(A32)*inv(A22)*inv(A11)*T6;
T6_ref3_array(:,:,3) = inv(A31)*inv(A23)*inv(A12)*T6;
T6_ref3_array(:,:,4) = inv(A32)*inv(A24)*inv(A12)*T6;

j = 1;
for i = 1:1:4
T6_ref3 = T6_ref3_array(:,:,i);
if abs(T6_ref3(3,3)) == 1
% d";g";n";r", nous d";finissons psi comme 0
EulerAngle_Invk(j,3) = 0;
else
EulerAngle_Invk(j,2) = pi;
end
% en ";tat d";g";n";r", nous d";finissons psi comme 0
EulerAngle_Invk(j,2) = 0;
end

```

```

EulerAngle_Invk(j,1) = atan2(T6_ref3(1,2),T6_ref3(1,1));

EulerAngle_Invk(j+1,:) = EulerAngle_Invk(j,:);
EulerAngle_Invk(j+1,1) = EulerAngle_Invk(j+1,1) + pi;
else
% pas d'"g"\n"r"
end

% column 1 de EulerAngle est phy
EulerAngle_Invk(j,1) = atan2(T6_ref3(2,3),T6_ref3(1,3));
EulerAngle_Invk(j+1,1) = atan2(T6_ref3(2,3),T6_ref3(1,3))+pi;

% column 2 de EulerAngle est theta
EulerAngle_Invk(j,2) = atan2((cos(EulerAngle_Invk(j,1))*T6_ref3(1,3)+sin(EulerAngle_Invk(j,1))*T6_ref3(2,3)),T6_ref3(3,3));
EulerAngle_Invk(j+1,2) =
atan2(cos(EulerAngle_Invk(j+1,1))*T6_ref3(1,3)+sin(EulerAngle_Invk(j+1,1))*T6_ref3(2,3),T6_ref3(3,3));
% column 3 de EulerAngle psi
%EulerAngle_Invk(j,3) = atan2( (-sin(EulerAngle_Invk(j,1))*T6_ref3(1,1) + cos(EulerAngle_Invk(j,1))*T6_ref3(2,1)) ,
-sin(EulerAngle_Invk(j,1))*T6_ref3(1,2)+cos(EulerAngle_Invk(j,1))*T6_ref3(2,2) );
%EulerAngle_Invk(j+1,3) = atan2( -sin(EulerAngle_Invk(j+1,1))*T6_ref3(1,1) + cos(EulerAngle_Invk(j+1,1))*T6_ref3(2,1) ,
-sin(EulerAngle_Invk(j+1,1))*T6_ref3(1,2)+cos(EulerAngle_Invk(j+1,1))*T6_ref3(2,2) );
EulerAngle_Invk(j,3) = atan2( -sin(EulerAngle_Invk(j,1))*T6_ref3(1,1)+cos(EulerAngle_Invk(j,1))*T6_ref3(2,1) ,
-sin(EulerAngle_Invk(j,1))*T6_ref3(1,2)+cos(EulerAngle_Invk(j,1))*T6_ref3(2,2));
EulerAngle_Invk(j+1,3) = atan2( -sin(EulerAngle_Invk(j+1,1))*T6_ref3(1,1)+cos(EulerAngle_Invk(j+1,1))*T6_ref3(2,1) ,
-sin(EulerAngle_Invk(j+1,1))*T6_ref3(1,2)+cos(EulerAngle_Invk(j+1,1))*T6_ref3(2,2));
end
j = j+2;
end
% =====sortie et montrer le r'"sultat de la cin"matique inverse===== %

Result = zeros(8,6);

Result(1,:) = [ theta1_1, theta2_1, theta3_1, EulerAngle_Invk(1,1), EulerAngle_Invk(1,2), EulerAngle_Invk(1,3) ];
Result(2,:) = [ theta1_1, theta2_1, theta3_1, EulerAngle_Invk(2,1), EulerAngle_Invk(2,2), EulerAngle_Invk(2,3) ];
Result(3,:) = [ theta1_1, theta2_2, theta3_2, EulerAngle_Invk(3,1), EulerAngle_Invk(3,2), EulerAngle_Invk(3,3) ];
Result(4,:) = [ theta1_1, theta2_2, theta3_2, EulerAngle_Invk(4,1), EulerAngle_Invk(4,2), EulerAngle_Invk(4,3) ];
Result(5,:) = [ theta1_2, theta2_3, theta3_1, EulerAngle_Invk(5,1), EulerAngle_Invk(5,2), EulerAngle_Invk(5,3) ];
Result(6,:) = [ theta1_2, theta2_3, theta3_1, EulerAngle_Invk(6,1), EulerAngle_Invk(6,2), EulerAngle_Invk(6,3) ];
Result(7,:) = [ theta1_2, theta2_4, theta3_2, EulerAngle_Invk(7,1), EulerAngle_Invk(7,2), EulerAngle_Invk(7,3) ];
Result(8,:) = [ theta1_2, theta2_4, theta3_2, EulerAngle_Invk(8,1), EulerAngle_Invk(8,2), EulerAngle_Invk(8,3) ];

% Le r'"sultat est les variables communes montr'"es dans rads

% normaliser l'angle et le laisser entre 0 et 360
% en rads son 0 "a 2 * pi

for i = 1:1:8
for j = 1:1:6
if Result(i,j) > 2*pi
Result(i,j) = rem(i,2*pi);
end
if Result(i,j) > pi
Result(i,j) = Result(i,j)-2*pi;
end
end
end

% =====v'"ifier si le r'"sultat est hors limites===== %

OutofRange = zeros(8,6);

% avant de v'"ifier disponible, l'a chang'" de rad "a degre"
Result = Result*(180/pi);
for i = 1:1:8
if abs(Result(i,1))>160
OutofRange(i,1) = 1;
else
OutofRange(i,1) = 0;
end

if abs(Result(i,2))>125
OutofRange(i,2) = 1;
else
OutofRange(i,2) = 0;
end

if abs(Result(i,3))>135
OutofRange(i,3) = 1;
else
OutofRange(i,3) = 0;
end
if abs(Result(i,4))>140
OutofRange(i,4) = 1;
else
OutofRange(i,4) = 0;
end

if abs(Result(i,5))>100
OutofRange(i,5) = 1;

```

```

else
OutofRange(i,5) = 0;
end

if abs(Result(i,6))>260
OutofRange(i,6) = 1;
else
OutofRange(i,1) = 0;
end
end

%===== afficher resultat =====%
showResult = zeros(16,6);
for i = 1:1:8
showResult(2*i-1,:) = Result(i,:);
showResult(2*i,:)= OutofRange(i,:);
end

% =====Choisissez un ensemble comme r'';sultat ===== %
% joint_variables = R'';sultat

% s'il y a un ensemble pour tous pas hors de port'';e, montrez-le comme r'';sultat
% si plus d'un ensemble de r'';sultats, ce programme montre le premier
% s'il n'y a pas d'ensemble pour tous hors de port'';e, affichez-le comme [-1, -1, -1, -1, -1, -1]

% initialise les joint_variables comme [-1 -1 -1 -1 -1 -1]
% return [-1, -1, -1, -1, -1] pour montrer que ce cart'';sien n'est pas disponible
% pour ce manipulateur de robot
joint_variables = [-1 -1 -1 -1 -1 -1];

%s'il y a des variables conjointes l'';gales, il sera
for i = 1:1:8
OutofRange = sum(OutofRange(i,:),2);
if(sum(OutofRange(i,:),2)) == 0
% cet ensemble de r'';ponse est l'';gal
joint_variables = Result(i,:);
break;
end
end

```