Etudiant :

HART Tristan

Tuteurs :

ASTORI Rodolphe

VANTROYS Thomas

BOE Alexandre

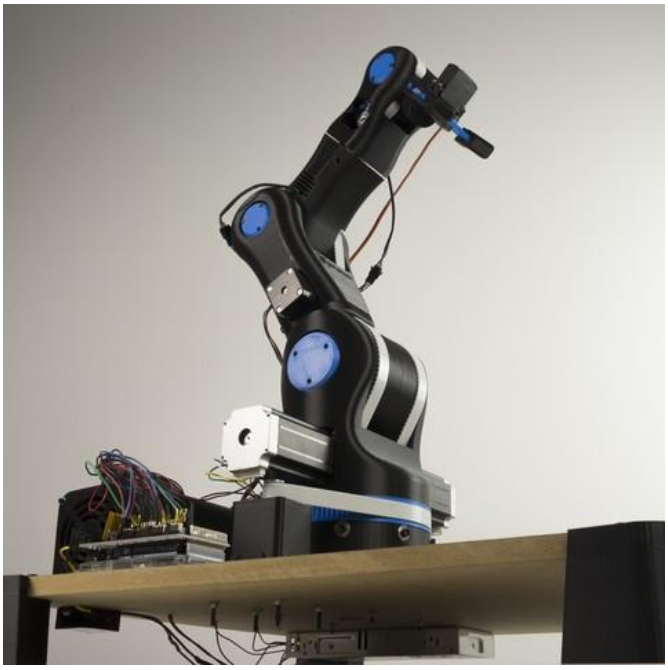
REDON Xavier

INGÉNIEUR POLYTECH LILLE

Département Informatique Microélectronique Automatique

Mémoire Final de PFE

DEVELOPPEMENT D'UN COBOT



Année universitaire 2017-2018

POLYTECH LILLE

Université de Lille

SOMMAIRE

INTRODUCTION

Ce projet se déroule dans le cadre de ma formation ingénieur Informatique Microélectronique Automatique à l'Ecole Polytechnique Universitaire de Lille. Lors de cette formation, il m'est demandé de travailler sur un projet de fin d'étude. J'ai choisi de travailler sur le développement d'un robot collaboratif, ou cobot, à partir d'un bras robotique open source intégralement imprimé à l'imprimante 3D. Mon choix s'est porté sur ce sujet pour plusieurs raisons. Tout d'abord, car la cobotique est un sujet qui me fascine, ce projet était donc pour moi une occasion d'assouvir ma curiosité à propos de domaine. De plus, la cobotique est en pleine croissance dernièrement, on peut dire que la cobotique est un des éléments clés de l'usine du futur. C'était donc pour moi l'occasion de me former sur ce sujet à la fois réaliste et innovateur.

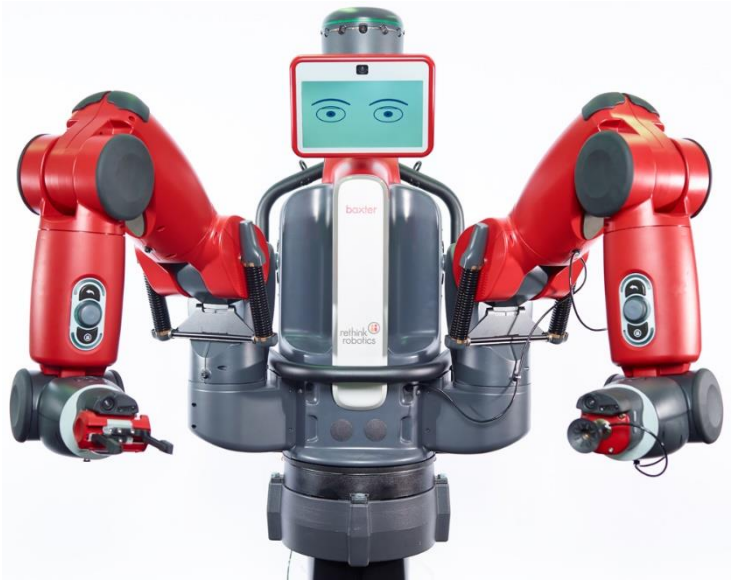
Ce projet va me permettre de mettre en pratique de nombreuses compétences vues au cours de ma formation notamment autour de l'informatique, de la conception et de l'électronique. La partie conception sera réalisée sous le logiciel Onshape et la programmation se fera sous Arduino.

Ce rapport apparait dans le cadre de ce projet de fin d'étude afin de rendre compte, au terme des 4 mois et demi alloués, du travail accompli. Ainsi, plusieurs points seront abordés dans ce rapport, comme le cahier des charges, le travail effectué, les difficultés rencontrées ou encore les perspectives d'amélioration. Bien entendu, ce rapport viendra se clore avec une conclusion faisant la synthèse de ce dossier.

I/ PRESENTATION DU PROJET

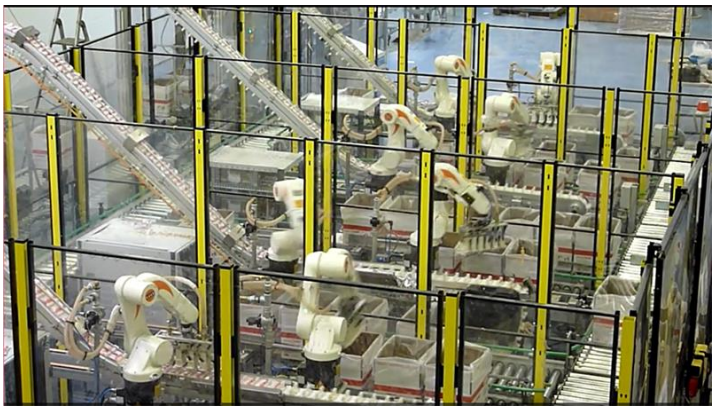
a) Qu'est-ce que la cobotique ?

Les robots industriels font partie intégrante de notre paysage industriel. Ils sont de véritables machines capables d'effectuer des tâches en suivant un programme de façon automatique. Ainsi, la **robotique** joue un rôle majeur au sein des **industries** en permettant d'automatiser des tâches pénibles, d'améliorer la productivité et la qualité de la production. Pourtant, la robotique industrielle « Traditionnelle » comporte certaines faiblesses : elle demande un investissement lourd, une mise en place complexe et est surtout peu flexible.



C'est dans ce contexte qu'intervient le robot collaboratif. Une des forces importantes du robot collaboratif est sa facilité de programmation. Le cobot est très simple à programmer et permet ainsi pour des fonctionnalités simples de programmer soi-même le robot. À titre d'exemple, le robot collaboratif est capable d'apprendre par le geste, l'opérateur effectuera les mouvements souhaités avec le bras du robot, qui se souviendra de ces gestes et pourra ainsi les répéter. Il peut être facilement déplacé, réinstallé et reprogrammé, peut alors être intégré à de multiples projets et non pas être simplement cantonné à une tâche unique.

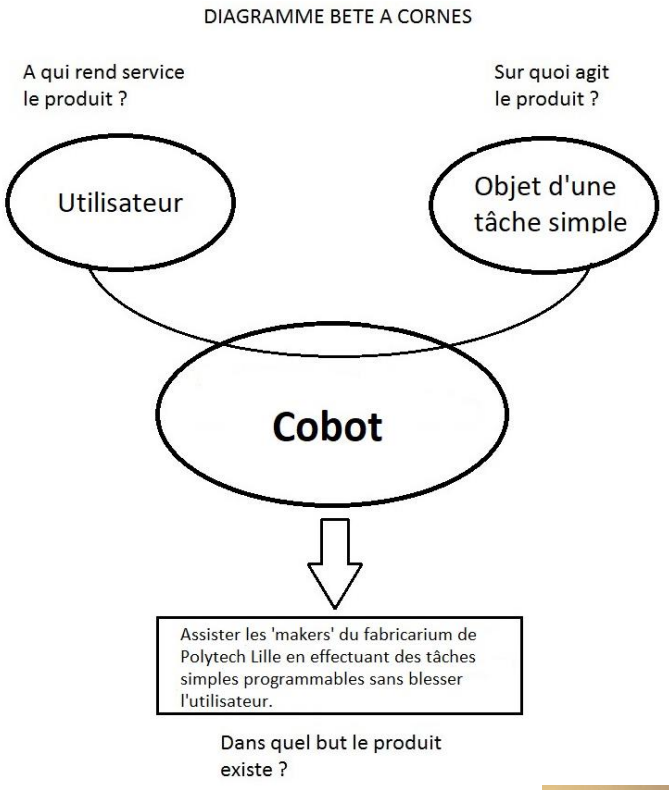
Le cobot est également plus économique. En effet, le robot traditionnel, pour des mesures de sécurité, doit être entouré de barrières, ce dispositif couteux n'est pas obligatoire dans les projets intégrant un robot collaboratif**.** Les cellules robotiques collaboratives permettent ainsi de gagner 30 à 40% de surface au sol par rapport à une cellule classique.



Enfin, majoritairement à cause de la panoplie de capteurs embarqués sur un cobot, il reste moins rapide en termes de vitesse de mouvement comparé à un robot classique. D'un point de vue conception, le cobot est également moins robuste que le robot industriel.

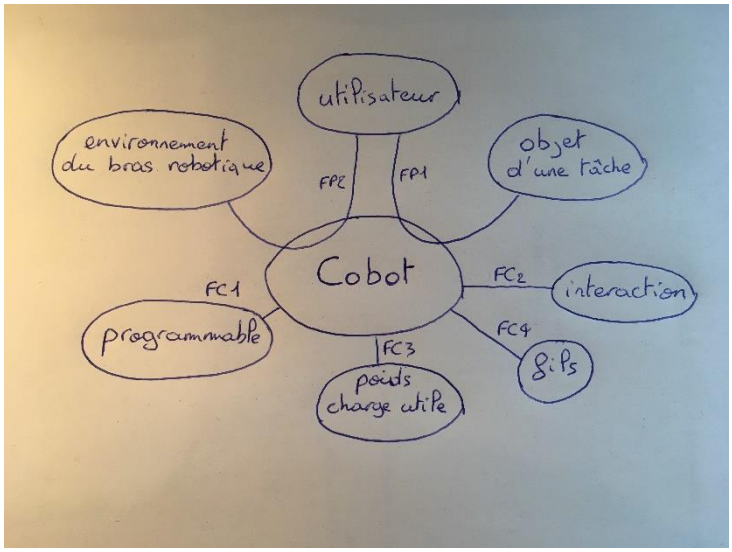
b) Cahier des charges

Comme pour tout projet, il faut définir un cahier des charges permettant de d'identifier le projet et toutes ses spécificités. Pour cela, je me sers de l'analyse fonctionnelle, une démarche qui consiste à rechercher et à caractériser les fonctions offertes par un produit pour satisfaire les besoins de son utilisateur. Pour le projet, je me sers de la 'bête à corne', du diagramme 'pieuvre', du diagramme 'FAST' et d'un diagramme de Gantt.



Le diagramme 'Bête à corne' ci-contre sert à mettre en évidence le but du projet, à bien définir les acteurs et les objectifs d'un produit.

Ci-dessous, le diagramme 'pieuvre' sert à identifier chaque fonction du projet et de les séparer en fonctions principales, qui sont les buts des relations créées par l'objet entre au moins deux éléments de son milieu extérieur, et fonctions contraintes, qui sont des exigences d'un élément contraignant du milieu extérieur.



Ici, nos fonctions sont :

FP1: Assister l'utilisateur dans la réalisation d'une tâche.

FP2: Préserver l'environnement du bras (humain ou matériel).

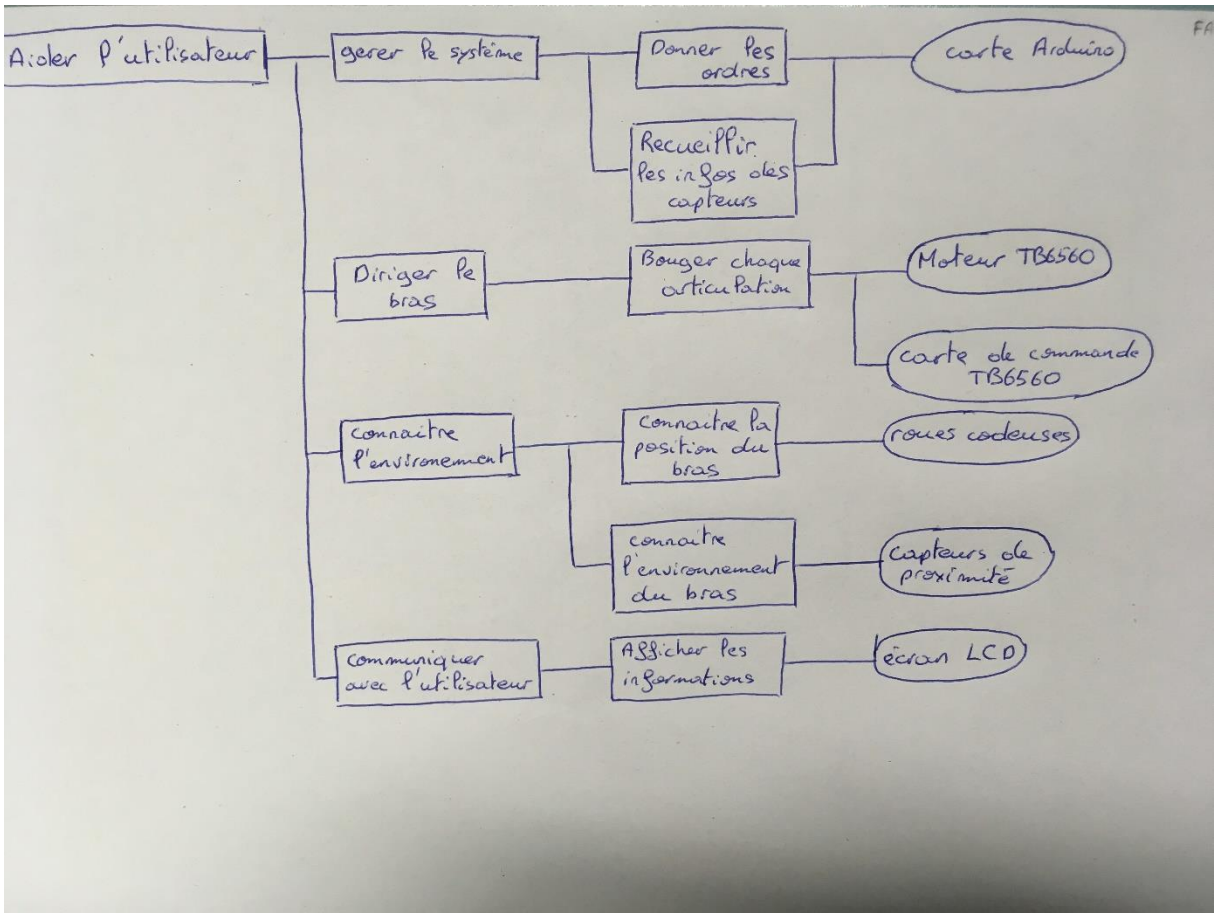
FC1: Être programmable.

FC2: Interagir avec l'utilisateur.

FC3: Doit savoir soutenir une charge.

FC4: Protéger le câblage apparent.

Le diagramme ‘FAST’ ci-dessous me sert à définir en amont le matériel qui sera nécessaire à la réalisation du projet. En partant d’un objectif du projet, on répond en se déplaçant sur la droite avec la question ‘comment ?’. On en vient donc à faire un choix préliminaire du matériel assurant chaque besoin.



Le matériel nécessaire que j’ai choisi avant le lancement réel du projet est donc le suivant :

- Le bras robotique open source Moveo BC3ND partiellement monté.
- Une carte Arduino MEGA 2560 afin de pouvoir modifier la position du robot.
- Un joystick shield arduino afin de pouvoir contrôler le robot et le faire bouger.
- Des moteurs pas-à-pas afin de motoriser le bras robotique de façon précise.
- Des cartes de commandes TB6560 afin de contrôler les moteurs par le biais de l'arduino.
- Des capteurs de types roues codeuses afin de pouvoir récupérer la position du robot.
- Un écran LCD pour arduino afin de renseigner l'utilisateur sur le mode de fonctionnement.
- Des capteurs de proximités afin de connaitre l’environnement du robot.
- Des sondes de courant, afin d'effectuer le monitoring du courant dans les moteurs et détecter une collision.
- Un accéléromètre afin de détecter une éventuelle détection.

Maintenant que le projet est bien compris, je peux créer un planning prévisionnel ayant pour but de définir comment les tâches vont se suivre afin de parvenir à l'aboutissement du projet. Dans mon cas, je dois commencer par terminer de concevoir mon bras robotique qui n'est que partiellement monté. Afin de bien maitriser sa programmation et sa dynamique, je le programmerai en chaine cinématique ouverte (axe par axe) avant d'implémenter la partie capteur relative au cobot. Puis j'implémenterai la partie capteur sur un premier axe, puis sur un deuxième etc… jusque à gérer tout le robot.

Les tâches de mon projet, avec leurs dates prévisionnelles de début et de fin sont résumées sur le tableau suivant :

Nom	Date de début	Date de fin	Durée
Tâches Préliminaires	25/09/17	20/10/17	20
Cahier des charges & Bibliographie	25/09/17	29/09/17	5
Terminer la conception du bras robotique.	02/10/17	06/10/17	5
Motorisation du bras robotique.	09/10/17	20/10/17	10
Partie 1: Base du Mouvement:	23/10/17	10/11/17	15
Mouvement du bras sur 1 axe en boucle ouver...	23/10/17	01/11/17	8
Mouvement du bras en chaine cinématique ou...	02/11/17	10/11/17	7
Partie 2: Mode collaboratif sur 1 axe:	13/11/17	15/12/17	25
Détection d'un obstacle sur le premier axe.	13/11/17	01/12/17	15
Reprise du mouvement après obstacle.	04/12/17	15/12/17	10
Partie 3: Mode collaboratif sur 2 axes:	08/01/18	26/01/18	15
Détection d'un obstacle sur le deuxième axe.	08/01/18	12/01/18	5
Reprise du mouvement après obstacle.	15/01/18	19/01/18	5
Fusion des programmes	22/01/18	26/01/18	5
Partie 4: Robot global:	29/01/18	16/02/18	15
Détection sur les autres axes.	29/01/18	02/02/18	5
Reprise du mouvement après obstacle.	05/02/18	09/02/18	5
* Fusion des programmmes	12/02/18	16/02/18	5

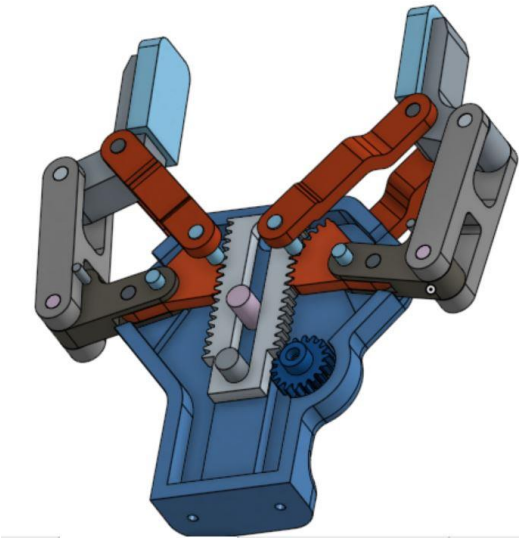
Maintenant que le projet est compris, que mes premiers choix sont validés par mes tuteurs et que ma trame de travail est définie, je peux commencer à réellement travailler sur mon robot en commençant par la fin de sa conception.

II/ TRAVAIL REALISE

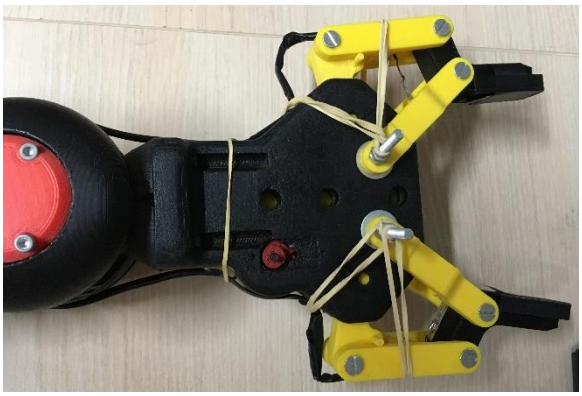
a) Conception du bras robotique

Afin de terminer la conception du bras robotique Moveo BCN3D présent au Fabricarium, il convient d'achever la création de plusieurs éléments du bras. Il faut entre autres créer le préhenseur venant se placer au bout du bras ainsi que la base sur laquelle viendra se fixer notre robot démonstrateur.

Pour le préhenseur du bras, je pensais réutiliser le préhenseur que j'ai dû designer lors des séances "d'initiation à la mécatronique" lors du dernier semestre. Ce préhenseur (présenté ci-dessous) à été réalisé sous le logiciel de CAD en ligne 'Onshape'. Ce logiciel a pour intérêt de sauvegarder chacune des créations sur un cloud, ce qui nous permet d'accéder à nos fichiers depuis n'importe quel PC.

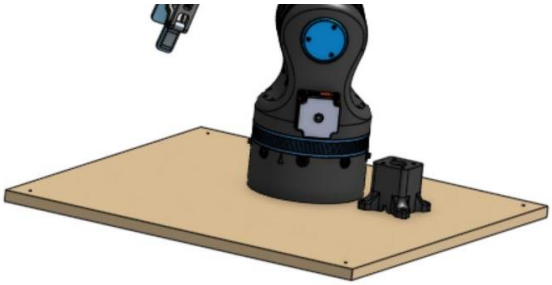


Le point fort de ce type de préhenseur est qu'il peut prendre des objets ronds ou plats grâce aux biellettes qui permettent à la pince de se courber pour s'adapter à la forme de l'objet. Néanmoins, il est nécessaire d'utiliser des élastiques sur la vraie pince afin d'éviter que la pince ne se mettent en position 'prise d'objet arrondi' à cause des frottements plastique-plastique.



J'ai rajouté des capteurs de force au bout des doigts de la pince (photo de droite). Leur intérêt est de mesurer la force appliquée à l'objet entre les doigts et donc de savoir si on exerce une trop grosse pression sur l'objet. Les 2 câbles de chaque capteur ont été collés ensemble avec du chatterton afin d'éviter leur dispersion et ainsi diminuer le risque qu'ils s'arrachent durant une tâche.

Pour ce projet, il faut également créer une base pour accueillir le bras. Cette base devra être assez lourde et assez solide afin de supporter le poids du robot ainsi que de sa charge. Cela devrait suivre l'exemple suivant :



Un plan de menuiserie a donc été réalisé et a été envoyé au menuisier de Polytech.

La planche fait 55cm x 50cm et est réalisée en contreplaquée 15 mm, ce qui la rend assez lourde pour empêcher le robot de basculer en cas de port de charge lourde.

Maintenant que le robot est fini et qu'il a été installé sur son support, il faut le doter de ses 6 moteurs afin qu'il puisse se mouvoir.

b) Motorisation du bras robotique

Le bras robotique Moveo BCN3D est doté de 6 emplacements moteurs prédéfinis. C'est un bras robotique avec 5 degrés de liberté. Chaque articulation est bougée via un système de poulies-courroies qui permettent un meilleur contrôle sur la position du robot.

Pronation / Supination :

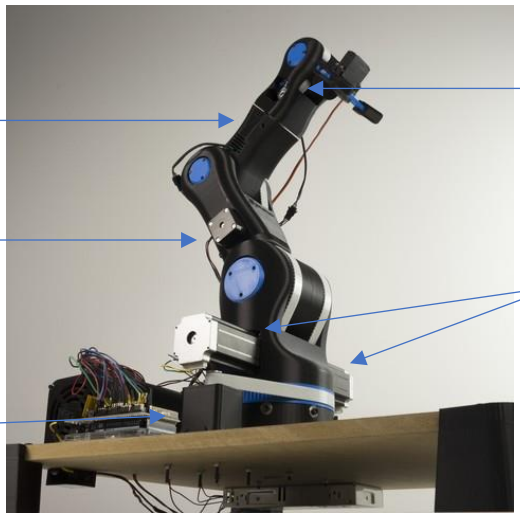
Moteur NEMA 17.

Articulation coude :

Moteur NEMA 17 avec réducteur 5:1.

Rotation base :

Moteur NEMA 17.



Articulation poignet :

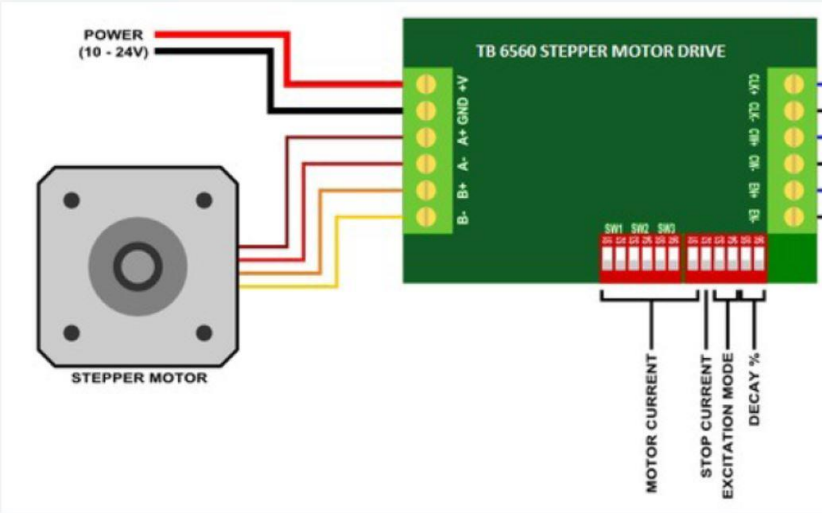
Moteur NEMA 14.

Articulation épaule :

2x Moteur NEMA 23.

Le schéma ci-dessus montre les types et les localisations des moteurs sur le bras robotique. Chacun de ces moteurs est contrôlé par un driver de moteur TB6560. Ce dernier sert à garder une commande de courant à injecter dans le moteur afin d'avoir un couple de maintien au niveau de l'articulation. Le robot garde donc sa position même si les moteurs ne sont pas utilisés.

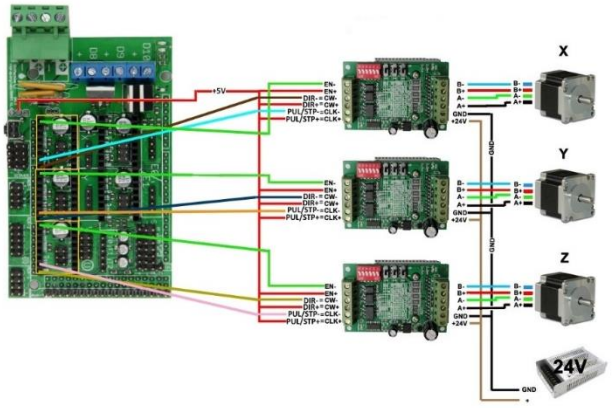
Les cartes TB6560 sont connectées selon le schéma suivant :

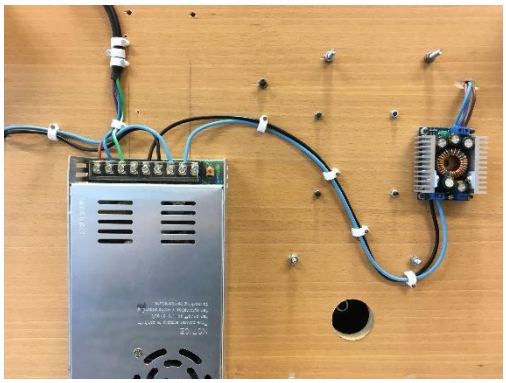


La tension d'alimentation de la carte est faite via un transformateur 24V se servant du réseau. La carte est quant à elle configurée pour des pas de 1/16', ce qui correspond à la segmentation de pas du moteur pas-à-pas.

Les 6 cartes TB6560 sont quant à elle commandées grâce à une carte Ramps V1.4. Son intérêt est qu'elle permet d'interfacer différents périphériques qu'on utilise souvent, comme des moteurs pas à pas, des dispositifs de puissances ou des contacts ou des servo-moteur. La carte est alimentée en 12V et est montée sur la carte Arduino Mega 2560. Le Ramps sert également à traduire les ordres de l'arduino en signal PWM compréhensible par les moteurs.

La carte Ramps et les drivers TB6560 sont cablés comme sur la photo en-dessous à gauche :

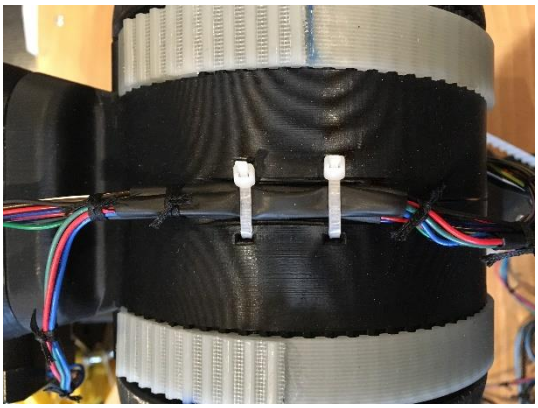




Pour une question d'habilitation électrique, la partie alimentation (au-dessus à droite) a été fixée sous la planche du robot de manière à ce que personne ne puisse mettre les doigts sur les transformateurs ou ne puisse arracher les câbles, ce qui représenterai un risque d'électrocution pour un opérateur ou une personne proche du robot.

Pour l'ensemble des moteurs, il a fallu agrandir la longueur des fils afin de pouvoir les relier jusqu'aux drivers. L'enjeux étant de pouvoir garder un jeu minimum pour les articulations tout en limitant les risques d'arrachages. Tous les moteurs possèdent 4 fils : un noir correspondant au A+, un vert correspondant au A-, un rouge correspondant au B+ et un bleu correspondant au B-. Sur les photos ci-dessous, vous pouvez voir comment ont été fixé les fils autour du robot.

La conception du robot incluait des creux qui permettent de faire passer les fils et de les fixer grâce à des colsons. Pour la base, les fils passent par le seul endroit sans courroie pour ne pas gêner la rotation. Enfin, le seul endroit où le câblage limite le mouvement du robot est sur la rotation du poignet. En effet, sans câblage, cette articulation pourrait tourner à l'infini, mais comme il y a des fils venant de l'articulation du poignet ou du servo-moteur au-dessus, les fils limitent le mouvement. En laissant suffisamment de jeu, l'articulation peut tout de même tourner d'environ 450°.

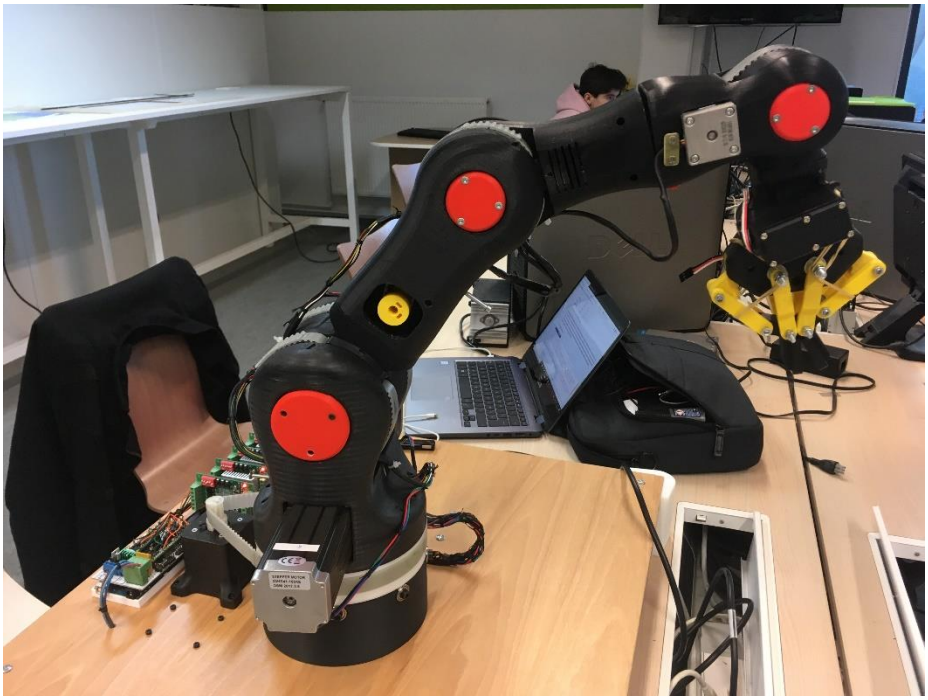


Mais pour deux moteurs, celui de la pronation/supination et le moteur du poignet, les fils étaient soit de couleur différente, soit plus nombreux. Il convient donc de faire un tableau d'équivalence des câbles pour signaler quel fil de couleur devient quelle couleur et à quelle phase ils correspondent :

Couleur du fil	Nouvelle couleur	Phase correspondante
Moteur Poignet:		
Rouge	Noir	A+
Jaune	Rouge	B+
Bleu	Vert	A-
Blanc	Bleu	B-

Couleur du fil	Nouvelle couleur	Phase correspondante
Moteur Pronation/Supination:		
Orange	Violet	A+
Orange & Blanc	blanc	A-
Rouge	Rouge	B+
Rouge & Blanc	Marron	B-
Noir	Noir	A-
Noir & Blanc	Bleu	A+
Jaune	Jaune	B-
Jaune & Blanc	Bleu	B+

Les moteurs sont donc installés et le câblage fait et protégé. Lorsque les cartes sont alimentées, le bras garde sa position, preuve que les moteurs sont alimentés et gardent leurs positions. Le seul problème est au niveau de la rotation du poignet où le moteur tourne dans le vide à cause de l'axe moteur qui tourne dans l'accouplement d'axe. Le bras est donc prêt à être utilisé de façon pérenne sans risque de dégradation matérielle. Il me faut maintenant commencer le codage du robot via ma carte arduino MEGA 2560.



c) Mouvement en cinématique directe

Maintenant que le moteur de la base est relié aux cartes de commandes, nous pouvons commencer à le faire bouger. Pour cela, nous nous servons d'une carte Arduino Mega 2560. Le programme sera implanté sur cette carte et commandera le Ramps 1.4 qui est un shield Arduino et qui vient donc se placer par-dessus la carte.

Pour faire bouger les moteurs pas-à-pas, une librairie 'stepper' est déjà présente dans les librairies par défaut Arduino, mais cette dernière n'est pas très adaptée pour la commande de plusieurs moteurs. C'est pour cela que je me sers de la librairie 'AccelStepper' qui simplifie le code pour ajuster plus précisément la vitesse du moteur, l'accélération ou effectuer des mouvements complexes.

Je commence donc par me familiariser avec les mouvements du robot et avec l'utilisation de la librairie en démarrant par le mouvement de la base.

```
void Homing()
{
  Serial.print("Début de la phase de mise en position ""HOME""...\n...En cours...\n"
  int HomeBase = -(NBR_PAS_BASE/2);
  int AU=0;

  while(AU != 49){
    stepperB.moveTo(HomeBase);  // Bouge jusque à la position initiale
    stepperB.run();

    posB = stepperB.currentPosition(); // récupération du nombre de pas effectués de
    AU = Serial.read(); // arrêt d'uragence
    if(posB==(HomeBase))
      break;
  }
  Serial.print("Posistion de la base = ");Serial.print(posB);Serial.print("\t");
  Serial.print("Arret d'urgence:");Serial.print(AU);Serial.print("\n");
  Serial.print("Homing effectué pour la base.\n");
}
```

Ce programme fait bouger la base du bras. L'opérateur le met dans une position initiale prédéfinie, puis le bras fait une rotation pour se mettre dans la position 'HOME'. Un arrêt d'urgence a été implémenté afin de stopper le Homing en cas de problèmes. Il suffira à l'utilisateur d'envoyer un 1 sur le port série d'Arduino. Ce dernier est utilisé afin de visualiser la valeur des variables ainsi que l'avancée du programme.

Le problème majeur du moteur pas-à-pas, est qu'il ne connait pas sa position au démarrage. Ainsi lorsque le moteur démarre, sa position est donc considérée comme la position 0. Le robot ne peut donc pas bouger pour se mettre en position HOME quelle que soit sa position initiale. En effet, si le robot commence en bout de course, il cherchera à continuer à tourner, forçant sur la poulie et la courroie pour continuer sa rotation.

Maintenant que le mouvement sur la base est maitrisé et que la bibliothèque AccelStepper est maitrisée. Je peux étendre mon programme à l'ensemble du bras robotique. L'idée est, comme pour la base, de créer un programme où l'ensemble du robot se met en position HOME. Puis l'utilisateur peut faire bouger chacun des axes à son envie et peux également le faire revenir à sa position de repos. Il faudra également gérer les sorties de la zone d'activité pour éviter de forcer sur les poulies/courroies ou donner l'impression d'avoir heurter un obstacle. Je commence donc par recenser, pour chaque articulation, le nombre de pas pour faire un l'ensemble de la zone d'activité ainsi que le nombre de degrés représentés :

Moteur	Code Switch TB6560	Nombre de pas	nombre de degrés	vitesse
Base	111 101100	7800	180°	2000
Épaule	100 001100	4360	180°	1000
Coude	101 001000	5946	225°	1000
Rotation poignet	101 001100	2100	450°	500
Poignet	011 001100	4300	235°	1000

Je décide ensuite de créer ma propre librairie arduino que je nomme 'Moveo'. Elle contiendra l'ensemble des fonctions dont j'aurai besoin pour faire bouger mon robot. Cette classe contient les fonctions suivantes

- Une fonction de récupération de l'axe demandé.
- Les fonctions de mouvement de chaque axe.
- Les fonctions de récupération des positions des moteurs.
- Les fonctions de retour en position HOME et de repos.
- La fonction de calcul des zones impossible à atteindre pour les moteurs.

On peut donc appeler ces fonctions en écrivant par exemple 'moveo.PositionP()' pour récupérer la position du poignet.

L'utilisateur communique donc par le biais de la liaison série avec le bras robotique. Chacune des positions est rappelée et il n'y a aucun risque que le robot soit endommagé par un mauvais ordre de mouvement.

C'est le robot avec le programme à cet avancement du projet qui sera présenté aux journées portes ouvertes de Polytech Lille ainsi qu'à l'évènement "Maker Faire" du 9-10-11 février 2018.

Nous allons donc voir en détails dans les pages suivantes le contenu de ce programme.

- **La fonction de récupération de l'axe demandé :**

Cette fonction sert à la récupération de l'axe que l'utilisateur veut bouger. L'opérateur rentre donc un nombre entre 1 et 5 pour faire bouger l'articulation correspondante (1 pour la base, 2 pour l'épaule, 3 pour le coude, 4 pour la rotation du poignet et enfin 5 pour le poignet ; cette notation restera la même pour tous les programmes). Il peut également rentrer la lettre 'H' pour demander au robot de revenir en position HOME, 'R' pour revenir en position initiale et 'X' pour exécuter l'action d'exposition.

```
int Moveo::Recuperation_Axe()
{
  char x[1]={0};
  int nbr = 0;

  while(Serial.available()>0)
  {
    x[0] = Serial.read();
    delay(100);
  }

  sscanf(x,"%d",&nbr); //si valeur décimale: conversion du tableau de char en int

  if(x[0] == 'R')
      return 10;

  if(x[0] == 'H')
    return 9;

  if(nbr<6 && nbr>0)
      return nbr;

  if(x[0] == 'X')
      return 7;
}
```

La fonction ‘Serial.read()’ retournant la donnée venant du port série sous la forme d’un char, il est nécessaire de le transformer en entier avant de le renvoyer. Cela se fait par le biais de la fonction sscanf().

- **Les fonctions de mouvement de chaque axe :**

Ces fonctions contiennent les ordres de mouvement pour chaque moteur. Il y a donc 5 fonctions de mouvement pour chaque articulation.

```
void Moveo::MouvementB(int order)
{
  int OrderPas = order*43.33; // conversion du nombre de degrés en nombre de pas
  int target = stepperB.currentPosition() - OrderPas;

  while(stepperB.currentPosition() != target)
  {
    stepperB.moveTo(target);
    stepperB.run();
  }
}
```

Pour faciliter l’utilisation du robot, l’utilisateur ne rentre pas d’ordres de mouvement en pas. En effet, cette donnée n’est pas visuelle et un pas n’équivaut pas au même angle pour chaque moteur à cause de sa conception mais également à cause de la segmentation paramétrée sur les drivers.

Ainsi, un mouvement d’un degré correspond à :

- 43.33 pas pour la base.
- 24.22 pas pour l’épaule.
- 26.42 pas pour le coude.
- 4.66 pas pour la rotation du poignet.
- 19.00 pas pour le poignet.

Ce nombre est obtenu en divisant le nombre de pas maximum pour un mouvement par le nombre de degrés maximum pour un mouvement.

- **Les fonctions de récupération de la position des moteurs :**

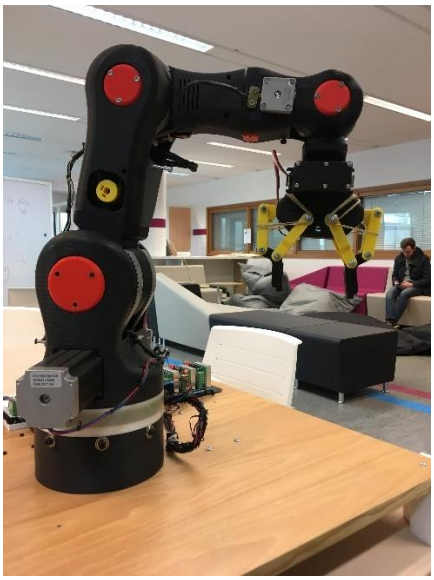
Ces fonctions servent à récupérer la position de chacun des moteurs du robot. Il y en a également 5 pour chaque articulation. Ces fonctions ont été créées pour suivre une « convention » autour de la position. En effet, selon leur position sur le robot, un moteur peut tourner dans son sens positif ou négatif. J’ai donc décidé que lorsque le robot se repliait sur lui-même, tous les moteurs devaient aller vers des positions négatives.

```
int Moveo::PositionB()
{
    int pB = 0;

    pB = - stepperB.currentPosition();

    return pB;
}
```

C’est pour cela que dans certaines fonctions comme celle ci-dessus, un signe ‘-‘ apparait devant la récupération de position.

- **Les fonctions de retour en position HOME et REPOS :**

Ces fonctions sont présentes afin qu'a tout moment, l'opérateur puisse demander au robot de retourner en position de REPOS (ci-dessous à gauche) ou en position HOME (ci-dessous à droite).



La position HOME est la position par défaut du robot, elle s'inspire des positions HOME des robots KUKA et ABB présents à Polytech Lille. La position REPOS quant à elle, est la position où l'alimentation peut être coupée sans endommager le robot. C'est la position où tous les axes sont en position 0.

```
void Moveo::Retour_Repos(int posB, int posE, int posC, int posRP, int posP)
{
    bool stop = false;

    stepperB.moveTo(0);
    stepperE.moveTo(0);
    stepperC.moveTo(0);
    stepperRP.moveTo(0);
    stepperP.moveTo(0);

    while(stop!= true)
    {
        stepperE.run();
        stepperB.run();
        stepperC.run();
        stepperRP.run();
        stepperP.run();

        if(stepperB.currentPosition()== 0 && stepperE.currentPosition()== 0 && st
        stepperP.currentPosition() == 0)
            stop = true;
    }

    Serial.println("Robot en position de repos\n");
}
```

Lorsque la fonction est appelée, le robot retourne sur chacune de ses positions 0. Le programme sort de la boucle lorsque chaque articulation est arrivée à la bonne place car les mouvements ne se terminent pas en même temps. Il faut donc vérifier que chaque moteur est bien placé.

```
void Moveo::Retour_HOME(int posB, int posE, int posC, int posRP, int posP, int HomeB,
{
    int movB, movE, movC, movRP, movP;

    movB  = HomeB-posB;
    movE  = HomeE-posE;
    movC  = HomeC-posC;
    movRP = HomeRP-posRP;
    movP  = HomeP-posP;

    bool stop = false;


    // Pour effectuer le mouvement avec tous les moteurs simulatnément:
    stepperB.move(-movB* 43.33);
    stepperE.move(movE*24.22);
    stepperC.move(movC*26.42);
    stepperRP.move(movRP*4.66);
    stepperP.move(movP*19.00);

    while(stop!= true)
    {
        stepperE.run();
        stepperB.run();
        stepperC.run();
        stepperRP.run();
        stepperP.run();


        if(-stepperB.currentPosition()<= (HomeB+1)*43.33 && stepperE.currentPosition(
        stepperRP.currentPosition() == HomeRP*4.66 && stepperP.currentPosition() == (
            stop = true;
    }

    Serial.println("Robot en position HOME!\n");
}

////////////////////////////////////////////////////////////////////////////////
```

La fonction de retour en position HOME quant à elle prend en argument la position de chaque moteur ainsi que la position HOME de chaque moteur. En faisant la différence entre ces 2 positions, on peut savoir le nombre de degrés à effectuer. On bouge ensuite de la même façon que dans la fonction de retour en position REPOS.

On peut remarquer la présence de 2 fonctions de mouvement différentes : stepperX.moveTo() et stepperX.move(). La différence vient du fait que le premier est un ordre de mouvement absolu, stepperX.moveTo(100) amènera le moteur à la position 100. Le stepperX.move(100) est quant à lui relatif, et fera bouger le moteur de 100 pas depuis sa position actuelle.

La fonction de mise en position HOME, peut se présenter sous 2 formes, une forme où chaque mouvement est fait un à un, ce qui équivaut à une suite de fonctions de mouvement que j'ai créé. Mais la 2 forme, celle qui a été retenue, est plus esthétique en termes de mouvement du robot. En effet, en mettant tous les stepperX.run() dans la même boucle, tous les moteurs vont se mouvoir en même temps pour atteindre sa position. Lorsque un moteur a atteint sa position, son stepper.run() n'effectuera plus aucune action tandis que les moteurs encore en mouvement continuerons leur course.

- **La fonction de calcul des zones impossible à atteindre pour les moteurs (deadzones).**

La fonction de gestion des deadzones est l'une des fonctions les plus importantes du programme. En effet, c'est cette dernière qui est capable de déterminer si l'opérateur demande un mouvement qui sort de la zone d'activité du robot. Le bon fonctionnement de cette fonction garantie ainsi l'intégrité matérielle du robot en empêchant par exemple d'aller à une position qui arracherait des fils ou forcerai sur les courroies.

```
bool Moveo::Deadzone(int moteur, int val, int pos)
{
    int i = moteur;
    int mov = val;
    int dir = pos + mov;

    switch (i)
    {
        case 1:
        {
            if(dir>180.00 || dir<0.00)
                return true;
            else
                return false;

            break;
        }

        case 2:
        {
            if(dir>80.00 || dir<-90.00)
                return true;
            else
                return false;
            break;
        }
```

La fonction Deadzone de la classe Moveo prend 3 arguments en entrée : le moteur qui va être utilisé, la valeur du mouvement demandé par l'opérateur et enfin la position actuelle du moteur qui va bouger.

La fonction calcule alors, grâce à la position et à l'ordre de mouvement, le point d'arrivée théorique du robot. Un switch, dépendant du moteur en question regarde alors si la valeur est atteignable. Si elle est atteignable, la fonction retourne un false, sinon un true pour dire que l'ordre n'est pas valable.

Les zones d'activité de chaque articulation sont :

- De 0° à 180° pour la base.
- De -90° à 80° pour l'épaule.
- De -115° à 115° pour le coude.
- De -130° à 270° pour la rotation du poignet.
- De -115° à 115° pour le poignet.

Remarque : Ces positions ne sont pas les positions extrêmes du robot. Ces positions sont placées légèrement avant les fins de course afin d'éviter tout risque de dégradation matérielle. Par exemple, lorsque l'épaule dépassent la position 80°, les courroies commencent à craquer. Arrivé en position 90°, les courroies se cassent en 2 du fait de leur assemblage.

D'autres fonctions ont été créées mais n'ont pas besoin d'être vue ici. En effet, ces fonctions n'ont pour utilité que de faciliter la lisibilité du code.

Nous allons maintenant voir la partie Arduino du programme en effet les fonctions vues auparavant faisaient partie d'un fichier C++. Ces fonctions sont utilisées dans le code principal qui est réalisé sous Arduino. Ce programme se décompose en 4 parties : la définition des PINs, la définition des variables globales, le setup et le programme principal qui est exécuté en boucle.

- **La définition des PINs :**

Comme la carte Arduino MEGA 2560 n'est pas directement reliée aux drivers de moteurs mais est montée avec un shield, il convient de définir les équivalences de chaque PIN de l'arduino pour le Ramps 1.4. En cherchant sur internet je trouve donc un fichier avec la majorité des #Define nécessaires à l'utilisation du Ramps. On y retrouve les PINs de Direction, d'Enable, de Step pour chaque moteur mais également les PINs de commande des ventilateurs, sondes de température etc… Des recherches supplémentaires ont été nécessaires afin de déterminer les PINs pour la commande du 5ème moteur et de la troisième sonde de température.

La définition de tous les PINs est disponible en Annexe [1]

```
// Définition des PINs pour le RAMPS 1.4
#define X_STEP_PIN         54
#define X_DIR_PIN          55
#define X_ENABLE_PIN       38
#define X_MIN_PIN           3
#define X_MAX_PIN           2
```

```
#define TEMP_0_PIN          13
#define TEMP_1_PIN          14
#define TEMP_2_PIN          15
```

Ces PINS sont nécessaire pour la définition des moteurs :

```
AccelStepper stepperB(1,X_STEP_PIN , X_DIR_PIN); // Moteur de la Base
AccelStepper stepperE(1,Y_STEP_PIN , Y_DIR_PIN); // Moteur de l'épaule
AccelStepper stepperC(1,Z_STEP_PIN , Z_DIR_PIN); // Moteur du coude
AccelStepper stepperRP(1,E0_STEP_PIN , E0_DIR_PIN); // Moteur de la rotation du poignet
AccelStepper stepperP(1,E1_STEP_PIN , E1_DIR_PIN); // Moteur du poignet
```

Pour chaque moteur, il faut définir s'il est commandé par un driver (le 1), son PIN de Step et son PIN de Direction.

- **Définition des variables globales**

```
//Variables globales de position
int posP  = 0;
int posRP = 0;
int posC  = 0;
int posE  = 0;
int posB  = 0;

// variables des positions HOME en degrés
int HomePoignet    = 80;
int HomeRotPoignet = 0 ;
int HomeCoude      = -80;
int HomeEpaule     = - 10;
int HomeBase       = 45;

//Variables de décision
int val = 0;
int mov = 0;
bool AU = false;
```

On retrouve dans cette partie les variables globales de position de chaque moteur, la définition des variables contenant la position de chaque articulation en position HOME, elle peut ainsi être modifier si l'utilisateur veut changer cette position. Enfin, on retrouve également les variables dites de décision, avec la variable 'val' dans laquelle est stocké l'axe que l'opérateur veut bouger, 'mov' qui contient le nombre de degrés que l'utilisateur veut effectuer et enfin la variable booléenne d'arrêt qui est utiliser pour sortir des menus.

- **Le setup**

C'est dans le setup que commence le programme exécuté par le robot :

```
void setup() {
  Initialisation();
  moveo.Attente();
  Homing();

  Serial.print("A vous de rentrer les positions que vous voulez atteindre:\n");
  Serial.print("1: Base ; 2: Epaule ; 3: Coude ; 4: Rotation poignet ; 5: Poignet\n");
  Serial.println("('H' pour retour en position HOME, 'R' pour le retour en position initiale");
  Affichage_Position();
}
```

On y retrouve la fonction Initialisation() qui initialise la liaison série et qui auparavant initialisait la vitesse de chaque moteur. Les vitesses sont maintenant initialisées dans le constructeur de la classe Moveo ainsi que les positions au démarrage des moteurs :

```
stepperP.setMaxSpeed(2000);
stepperP.setAcceleration(2000.0);

// Mise à l'origine des moteurs
stepperB.setCurrentPosition(0);
stepperE.setCurrentPosition(0);
```

On retrouve également une fonction qui attend que l'opérateur demande le début de la phase de mise en position HOME. Cela évite que le robot se mette à bouger dès sa mise sous tension.

Le robot se met ensuite en position HOME, affiche les positions de chaque moteur puis rentre dans le programme principal où l'opérateur pourra donner ses ordres de mouvement.

- **Le programme principal**

C'est la partie la plus importante du programme, la partie où l'opérateur donne des ordres au robot et que ce dernier s'exécute. Ce code s'exécute en boucle sans interruptions.

```
void loop()
{
  if(Serial.available()>0)
  {
    val = moveo.Recuperation_Axe();
    switch (val)
    {
      case 1: ////////////////////////
      {
```

Dans cette partie, on attend que le port série reçoive une donnée. Si une donnée est reçue, il s'agira de l'axe que l'opérateur veut bouger. On utilise donc la fonction de récupération de l'axe. Rappelons que cette fonction retourne un entier entre 1 et 6 pour l'articulation à bouger, un 9 pour le retour en position HOME et un 10 pour un retour en position REPOS.

S'en suit alors un switch qui amènera le programme à exécuter la partie correspondante à l'action voulue. Soit un mouvement d'axe, soit le retour à une position prédéfinie.

```
if(Serial.available()>0)
{

  mov = Recuperation_Valeur();

  if(moveo.Arret()=='E')
      AU = true;

  if(mov !=0)
  {
      if(moveo.Deadzone(1,mov,posB)==false)
      {
        Serial.print("Deplacement... ");Serial.print(mov);Serial.print(" degrés");Serial.print("\n");
        moveo.MouvementB(mov);
      }
      else
      {
        Serial.println("Valeur non atteignable car hors zone d'activité (0 à 180°)");
      }
    break;
```

Nous prendrons maintenant le cas où l'utilisateur veut bouger la base. On confirme à l'utilisateur qu'il va bouger la base puis on lui demande de combien de degrés il veut bouger. On poursuit alors avec le code ci-dessus.

Si l'utilisateur rentre un 'E', il sort du sous-menu et retourne au choix de l'articulation. S'il rentre un nombre différent de 0, on teste alors si le robot peut exécuter ce mouvement grâce à la fonction Deadzone(), dans le cas où ce n'est pas possible, le mouvement ne se fait pas et on précise à l'opérateur la zone d'activité de l'articulation. Si le mouvement est possible, on lance le mouvement avec la fonction de mouvement relative à la base.

Tous les autres cas du switch allant de 1 à 6 sont traités de la même façon que ci-dessus. Dans le cas des retours en positions HOME et REPOS, ils sont traités de la façon suivante :

```
case 10:  // Retour en position de repos ////////////////////////////////
  {
    Serial.println("Retour en position de repos demandé.");
    moveo.Retour_Repos(posB,posE,posC,posRP,posP);

    posP  =  (moveo.PositionP()/19);
    posRP =  (moveo.PositionRP()/4.66);
    posC  =  (moveo.PositionC()/26.4);
    posE  =  (moveo.PositionE()/24.2);
    posB  =  (moveo.PositionB()/43.33);

    mov=0;
    val=0;

    Serial.print("\nQuel Axe choisissez vous?\n");
    Serial.print("1: Base ; 2: Epaule ; 3: Coude ; 4: Rotation poignet ; 5: Poignet\n");
    Serial.println("('H' pour retour en position HOME, 'R' pour le retour en position initiale)");

    break;
  }
```

On appelle la fonction demandée, Retour_Repos() ou Retour_HOME(), puis on refait l'acquisition des positions, on réinitialise les variables de décisions puis on sort du switch.

De cette façon, l'utilisateur peut utiliser en continu le robot à sa guise.

Remarque : Une position d'exposition du robot a été créée pour la maker faire où le robot donne une carte de visite à un visiteur placé devant lui puis revient à la position initiale.

```
DEBUT DU PROGRAMME GLOBAL MOVEO BCN3D.

Liaison série initialisée.
Setup moteur effectué.
Appuyez sur 1 pour lancer la phase de HOMING.

Début de la phase de mise en position HOME...
...En cours...

Homing effectué!

A vous de rentrer les positions que vous voulez atteindre:
1: Base ; 2: Epaule ; 3: Coude ; 4: Rotation poignet ; 5: Poignet
('H' pour retour en position HOME, 'R' pour le retour en position initiale
Position du poignet: 81.33
Position de la rotation du poignet: 0.00
Position du coude: -80.04
Position de l'épaule: -10.00
Position de la base: 44.98
```

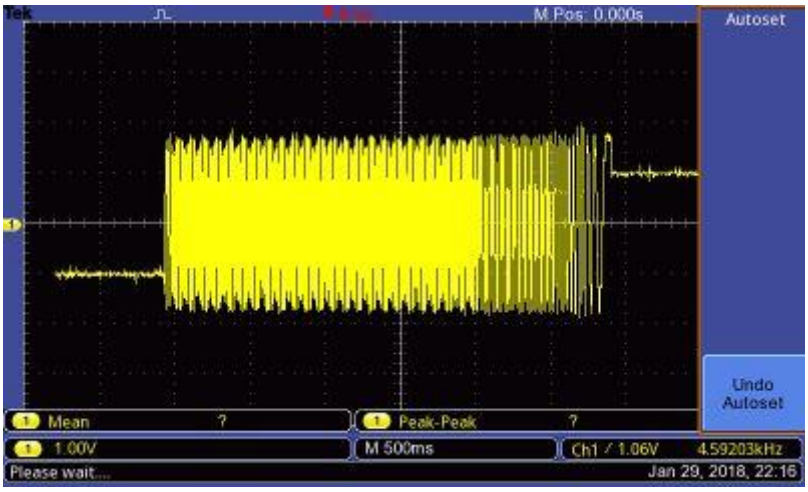
Ci- dessus, vous pouvez le résultat affiché sur le port série depuis le démarrage du bras jusque à la mise en position HOME. L'utilisateur communique donc par le biais de la liaison série de l'IDE Arduino. Chacune des positions est rappelée et il n'y a aucun risque que le robot soit endommagé par un mauvais ordre de mouvement.

C'est le robot avec le programme à cet avancement du projet qui sera présenté aux journées portes ouvertes de Polytech Lille ainsi qu'à l'évènement "Maker Faire" du 9-10-11 février 2018.

d) Mode collaboratif sur 1 axe

Le robot est désormais capable de se mouvoir comme voulu via la liaison série. Mais ce n'est pas encore un robot collaboratif. Pour cela, il doit être capable de détecter une collision, ce qui va maintenant être le nouvel objectif de mon projet.

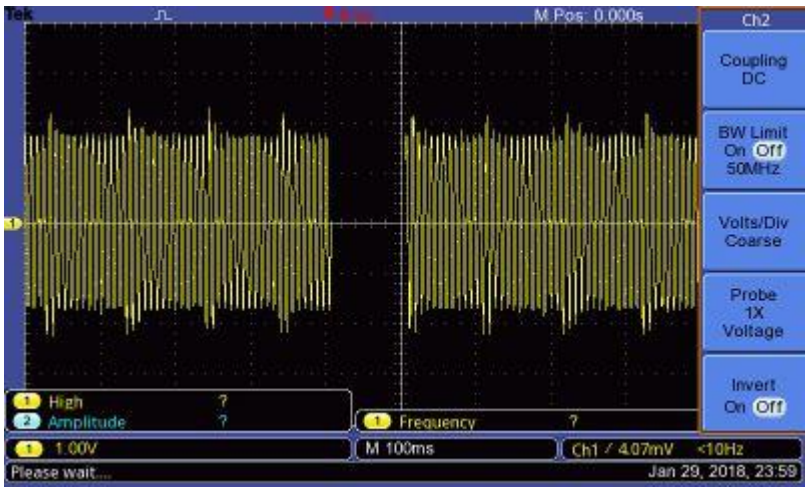
Pour détecter ces collisions, je décide de regarder du côté des moteurs. En effet, en cas de rencontre avec un obstacle, il y aura forcément une hausse du couple dans le moteur de l'articulation. Je décide donc de monitorer mon courant via une sonde de courant. Ce dernier est relié au couple par la relation Cm = Kc.Im, avec Cm le couple moteur, Kc la constante de couple et Im le courant traversant le moteur.

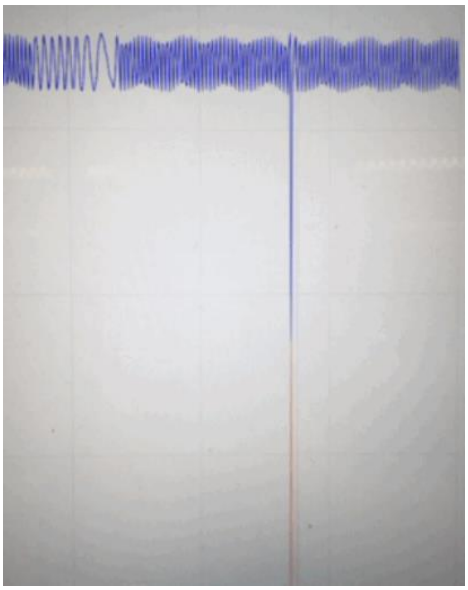


Le courant visualisé à l'oscilloscope se présente sous la forme sinusoïdale. A l'arrêt, il est soit positif soit négatif selon s'il penche à droite ou à gauche (courant de maintien de la position du moteur).

Maintenant que je suis fixé sur la forme d'onde du courant lorsque le moteur tourne ou est à l'arrêt, je décide de rajouter un obstacle qui va empêcher son mouvement :

On peut remarquer des sinusoïdes plus grandes que d'autres. Ce sont les hausses de couple qui l'entrainent. Mais ce pic ne reste pas sur plusieurs périodes car lorsque le moteur rencontre un obstacle en montant, il redescend puis remonte jusque à re-rencontrer l'obstacles. Ce pic d'une amplitude de 1.9A s'élève de 0.3A au-dessus des cycles normaux (1.6A). Je vais donc essayer de détecter ces pics.





Je n'arrive pas encore a récupérer de données sur les PINs auxiliaires du Ramps. J'utilise donc une arduino UNO pour l'instant afin de récupérer les données des transducteurs de courant. En effet, les documentations restent assez vagues quant aux numéros de PINs de la partie AUX ou SERVO.

Je lance un programme qui fait tourner la base en continu. Puis je lance un programme sur mon arduino UNO qui est connectée au transducteur de courant qui visualise le courant de la base. J'utilise alors l'outil 'Traceur Série' de l'IDE arduino pour visualiser le courant mesuré. Le programme en question affiche les formes d'onde du courant dans le moteur ainsi qu'un signal d'alarme. Lorsqu'un pic de courant est détecté, le signal d'alarme passe à 255 comme sur l'image ci-contre.

A gauche, en bleu, le courant dans le moteur, en rouge le signal d'alarme. Lorsque je place ma main pour bloquer le bras, un pic apparait. Le signal d'alarme rouge passe donc à 255. La détection d'obstacle est donc réussie. Le monitoring du courant se fait via un capteur à induction LEM qui est installé sur la phase B- du moteur. Le fait de monitorer sur plusieurs phases ajoute du bruit dans le signal qui ne devient plus exploitable.

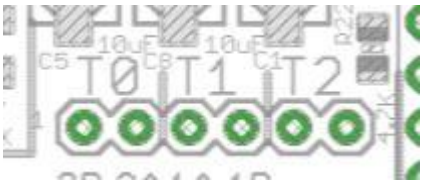
Je me penche maintenant sur un moyen de n'utiliser que le Ramps et non la carte arduino UNO couplée au Ramps pour détecter l'obstacle. Je pourrai alors faire un programme combinant la détection d'obstacle sur la rotation de la base et le mouvement en utilisant uniquement la MEGA. Je pourrai à ce moment là exécuter une tâche, lorsque le robot rencontre un obstacle il s'arrête puis reprend sa tâche lorsque l'obstacle est parti.

J'ai fait plusieurs recherches autour du Ramps, 4 possibilités s'offraient à moi pour récupérer le courant du moteur :

- La première était d'utiliser une arduino UNO sur laquelle brancher toutes les sondes, puis si une collision était détectée, la carte enverrait une donnée à l'arduino MEGA pour lui dire de stopper le mouvement.
- La deuxième solution était d'utiliser des PINs inutilisés du ramps comme le X_MIN_PIN, de le convertir en entrée pour lire la donnée dessus.
- La troisième était d'utiliser les PINs AUX du ramps. Certains étant analogiques, la possibilité de récupérer la donnée semblait viable.
- La dernière solution était d'utiliser les entrées associées aux sondes de températures pour récupérer la mesure.

La solution que j'ai finalement retenue est la dernière. En effet la première est viable, mais l'utilisation d'une carte qui communiquerai avec la MEGA n'est pas optimale. Elle pourrait néanmoins selon moi être revue dans le cas où on utilise plus de capteurs. En rajoutant les capteurs de proximité, les capteurs laser, l'accéléromètre et les transducteurs, on peut facilement imaginer la UNO avec un programme interne qui gèrera les capteurs, calculera les risques et qui enverra les données si nécessaires à la MEGA. La deuxième solution n'a pas non plus été retenue car elle ne retourne rien, en effet j'ai essayé de tourner les PINs en sortie mais la seule chose que je pouvais lire n'était que du bruit. La troisième solution, qui était d'utiliser les AUX de la Ramps pour lire les mesures, n'a pas été retenue car je n'arrivais pas à retrouver les PINs de la MEGA correspondants aux PINs de la Ramps. Je n'ai donc pas pu utiliser cette solution car ces PINs sont pour la plupart utilisé pour le branchement de cartes SD, écrans LCD... La Ramps étant principalement conçue pour les imprimante 3D, elle n'a pas été conçue pour qu'on puisse rajouter des capteurs.

J'utilise donc les entrées faites pour les sondes de température sur la Ramps. Le seul bémol étant que je n'ai que trois entrées, cela me permet d'au moins contrôler la rotation de la base, l'épaule et le coude. La rotation du poignet étant mécaniquement défectueuse et le poignet étant hors service pour cause thermique. Les trois entrées sont donc largement suffisantes.



J'alimente donc ma sonde avec les 5V que fourni le Ramps et je mets le GND et la sortie sur les pins du premier capteur de température. Je commence ensuite la création d'un programme simple pour la base : Une rotation de 90° dans un sens, puis de 90° dans l'autre en continu. Pour cela, aucuns problèmes, je maitrise la bibliothèque de mouvement des moteurs.

Le code est simple, je lui donne le nombre de pas à effectuer, ici 90 degrés. Puis je lance le mouvement, dans la boucle, j'effectue un pas et je fais l'acquisition du courant. Si j'ai atteint la position voulue, je sors du while. Si un pic de courant est détecté, je le dis, j'affiche la position où a eu lieu la collision et je lance la fonction test_collision() d'où je ne sortirai que lorsque l'obstacle ne sera plus présent :

```
while(arret != true)
{
  stepperB.run();

  courant = analogRead(TEMP_0_PIN); // acquisition de la valeur du transducteur de courant.

  if(stepperB.currentPosition()==PosB+400){
    arret = true;
  }


  if(courant>=545 || courant <=480)// Si il y a encore une hausse du courant = obstacle toujours présent
  {
    Serial.println("La collision est toujours présente.");
    Serial.print("Le programme reprendra à la position "); Serial.println(PosB);
    stepperB.setCurrentPosition(PosB);
    Serial.print("Position: "); Serial.println(stepperB.currentPosition());Serial.println();
    delay(500);
  }
```

Lorsqu'une collision est détectée, j'ai eu l'idée d'agir comme un aveugle le ferai. Une fois la collision détectée, je me stoppe pendant 1 seconde puis je regarde toutes les 500ms si l'obstacle est toujours présent. Ainsi le bras tourne et s'il ne rencontre plus rien sur 300 pas, il considère l'obstacle comme disparu. Il reprend donc sa tâche. Si l'obstacle est toujours présent, il faudra quelques pas pour détecter de nouveau la hausse de courant dans le moteur. C'est pour cela que si l'obstacle est détecté dans les 400 pas, il est compté comme étant toujours présent, et la position est réinitialisée à sa position au départ qui est mise en entrée de la fonction :

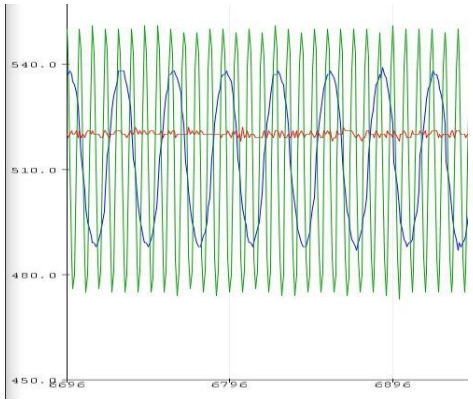
```
while(arret != true)
{
  stepperB.run();

  courant = analogRead(TEMP_0_PIN); // acquisition de la valeur du transducteur de courant.

  if(stepperB.currentPosition()==PosB+400){
    arret = true;
  }


  if(courant>=545 || courant <=480)// Si il y a encore une hausse du courant = obstacle toujours présent
  {
    Serial.println("La collision est toujours présente.");
    Serial.print("Le programme reprendra à la position "); Serial.println(PosB);
    stepperB.setCurrentPosition(PosB);
    Serial.print("Position: "); Serial.println(stepperB.currentPosition());Serial.println();
    delay(500);
  }
```

e) Mode collaboratif sur 3 axes

Maintenant que la méthode de la détection d'obstacle et de reprise de tâche est maitrisée sur la base, il est temps d'étendre l'aspect collaboratif sur l'épaule et le coude du robot. En effet, je n'ai pu avoir que trois entrées pour mes transducteurs de courant, je monitore donc mes 3 premières articulations. J'installe donc mes 2 nouvelles sondes de courant sur les phases B- de mes moteurs au niveau des drivers TB6560

Je lance ensuite mon programme affichant les 3 formes d'onde sur le traceur série de l'IDE Arduino.

```
Serial.print(currentB);
Serial.print(", ");
Serial.print(currentE);
Serial.print(", ");
Serial.print(currentC);
Serial.println();
```



Ci-contre, les formes d'ondes des 3 articulations avec en Bleu le courant de la base, en Rouge celui du coude et en vert le courant du coude.

A partir de là, je suis capable de définir les seuils à mettre pour considérer qu'un obstacle est présent. Ces seuils sont :

- De 480 à 545 pour la base.
- De 470 à 552 pour l'épaule.
- De 470 à 550 pour le coude.

Je créé une fonction qui prend en entrée le courant de chaque articulation et les teste. Si une collision est détectée, on précise sur quelle articulation elle est détectée, on enregistre la position de collision puis on lance la fonction qui déterminera lorsque le champ sera libre pour reprendre la tâche.

```
void DetectCollision(int currentB, int currentE, int currentC, int currentRP,
{
  int posColl = 0; // variable de stockage de la position à la collision

  // Test courant de la base
  if(currentB>=545 || currentB <=480) // Si hausse de couple détectée
    {
      Serial.print("Collision(B)! : ");Serial.println(currentB);
      posColl = stepperB.currentPosition();
      Serial.print("Position= ");Serial.println(posColl);
      test_collision(1,posColl);
    }

  // Test courant de l'épaule
  if(currentE>=552 || currentE <=470)
    {
```

La fonction test_collision() prend en argument un entier entre 1 et 6 pour le moteur en question puis la position de collision :

```
stepperB.move(10000);

while(arret != true)
  {
    stepperB.run();
    courant = analogRead(TEMP_0_PIN); // acquisition de la valeur du t
    if(stepperB.currentPosition()==Pos+200){
      arret = true;
    }


    if(courant>=545 || courant <=480)// Si il y a encore une hausse du
    {
      Serial.println("La collision(B) est toujours présente.");
      Serial.print("Le programme reprendra à la position "); Serial.pri
      stepperB.setCurrentPosition(Pos);
      Serial.print("Position: "); Serial.println(stepperB.currentPositi
      delay(500);
    }
```

La fonction commence par un switch qui va amener le programme sur la procédure à suivre selon le moteur. La partie de code ci-dessus montre la partie de code dans le cas 1, c'est-à-dire dans le cas de la base. Les autres cas sont traités de façon similaire à la différence du nombre de pas à faire pour considérer que l'obstacle n'est plus présent ainsi que les seuils de détection.

Dans cette fonction fait bouger le moteur en question, puis on refait l'acquisition du courant, si une hausse de courant est de nouveau détectée, l'obstacle est toujours présent. On considère alors que le robot n'a pas changé de place et on réinitialise sa position à celle de la collision initiale. On attend 500ms puis on réitère la procédure.

```
  Serial.println("La collision(B) n'est plus présente.");
  Serial.print("Reprise...");Serial.println(stepperB.currentPosition());
  break;
}
```

Si le robot arrive à faire un certain nombre de pas, dans notre cas 200 pas, sans détecter de collision, on considère que l'obstacle n'est plus présent. On peut donc reprendre notre tâche où nous l'avons laissé.

//

PHOTO MONITEUR SERIE

//

f) Moveo est-il collaboratif ?

Il faut maintenant vérifier que le robot répond bien aux normes en vigueur. Ces normes sont données par l'INRS au paragraphe 4.5 de la brochure sur Sécurité des Equipements Prévention des risques mécaniques.

Ce paragraphe se résume dans le tableau suivant :

	Valeurs n°1*	Valeurs n°2*
Effort maximal s'exerçant sur des parties du corps	75 N	150 N
Énergie cinétique maximale de la partie mobile	4 J	10 J**
Pression de contact maximale	50 N/cm²	

Dans notre cas, nous suivrons la série de valeurs 2 car le robot collaboratif est considéré comme compliant. Il faut donc calculer l'énergie à l'impact créée par le robot :

La vitesse du robot est de 0.24 m/s. Cette valeur est obtenue en utilisant la formule $\dot{\sigma} \cdot R^2$ avec $\dot{\sigma}$ la vitesse angulaire et R^2 la longueur de la partie mobile de robot qui vaut 63cm.

Son poids est de 4.8 Kg. Cette valeur est obtenue depuis le modèle 3D de Moveo. En retirant les pièces métalliques du robot comme les vis et les roulements, on peut obtenir le volume de PLA que le robot contient. La masse volumique de la PLA étant de 1,25 g/cm^3, on obtient le poids du robot. Enfin, l'impression a été réalisée avec un remplissage de 20%. On obtient donc en rajoutant le poids de chaque moteur donné par les datasheets, le poids total de tout le robot.

Pour calculer l'énergie cinétique à l'impact, il faut ensuite considérer que l'ensemble du poids du robot est au niveau de l'impact s'applique de façon linéaire. On applique donc la formule de l'énergie cinétique en translation $M \cdot \frac{v^2}{2}$ avec M le poids du robot et v la vitesse du robot.

Son énergie à l'impact est alors de **0.07J**.

Le robot Moveo rentre donc plus que largement dans les normes. Une si faible énergie est due au fait que le robot soit en PLA et ne se déplace pas à si grande vitesse. La norme a quant à elle été faite pour des robots fait en matières métallique et se déplaçant à des vitesses supérieures au m/s.

Le robot Moveo BCN3D peut donc être considéré comme collaboratif sur le premier axe.

III/ DIFFICULTES RENCONTREES

Comme lors de tout projet, j'ai été, ou je suis encore confronté à quelques difficultés. Des points qui se sont présentés comme des freins à l'avancement de mon projet.

Le problème majeur que j'ai rencontré concerne le matériel nécessaire à la motorisation du bras. Le design du bras est fixé et ne peut plus être modifié. Les moteurs et l'ensemble de l'électronique autour sont donc également imposés. Lors de mon lancement de projet, un devis avait été lancé chez un fournisseur qui proposait un kit comportant tous les éléments nécessaires à la motorisation du bras. Malheureusement, alors que j'étais en train de faire la fin de la conception de mon bras, il a été conclu que le fournisseur, qui ne répondait pas, n'enverrait pas le kit. Il a donc fallu que j'ailles dans la BOM (Bill Of Materials) du Moveo BCN3D afin d'aller chercher chaque composant nécessaire moi-même chez des fournisseurs tels que Gotronics, RS ou Mouser. La plupart du matériel a été retrouvé mais certains composants, comme le moteur avec le réducteur 5:1 ou les drivers TB6560, n'étant pas commun, ne se retrouvaient pas chez ces fournisseurs. Ils n'ont donc pas pu être commandés de suite. De plus, les drivers n'ont pu être commandés que sur un site qui possède des temps de livraison assez élevé. Ce qui m'a amené à recevoir mes cartes TB6560, primordiales à l'avancement du projet qu'à la rentrée 2018. Je n'ai donc pu commencer à réellement programmer qu'à la rentrée 2018. Pour éviter de prendre trop de retard, je me suis acheté une carte moi-même pour pouvoir commencer la programmation chez moi lors des vacances de Noel. Les méthodes de programmation étaient donc maitrisées à la rentrée ce qui m'a permis de ne pas perdre une semaine de plus en janvier.

Un autre problème que j'ai rencontré lors du projet et l'échauffement des moteurs lorsque ceux-ci sont à l'arrêt. En effet, pour maintenir en position les moteurs, les drivers TB6560 envoient en continu de la puissance dans les moteurs. Lorsque les moteurs sont en mouvement, une grande partie de cette puissance se dissipe en mouvement et peu en chaleur. Lorsque le moteur est à l'arrêt, c'est l'inverse, la puissance n'est pas dissipée en mouvement mais est quasiment intégralement dissipée en chaleur. Sur les moteurs de la base et de l'épaule, cet échauffement est négligeable. Le coude chauffe à peine après une utilisation prolongée. Néanmoins, les pertes en chaleur sont présente dans le moteur de la rotation du poignet, assez pour qu'on puisse sentir sa chaleur au travers de la PLA environnante. Le moteur du poignet quant à lui, chauffe de façon très dangereuse. En effet, après une utilisation de plus d'une heure, ce dernier chauffe au point d'attraper une ampoule sur mon doigt. Ainsi la poulie imprimée en PLA, se déforme sous la contrainte exercée par la courroie en mouvement et finit par se désaxer, rendant l'articulation inopérante.

IV/ PERSPECTIVES

a) Travail Restant

A ce stade du projet, ce dernier n'est pas abouti. En effet, il reste encore quelques tâches à réaliser comme étendre le mode collaboratif à l'ensemble du robot et non aux 3 premières articulations. Cela implique de trouver un nouveau moyen d'ajouter des capteurs au Ramps ou de trouver une méthode alternative à l'acquisition des données capteurs. De plus, les seuls capteurs utilisés sont les sondes de courant LEM, le cahier des charges impliquait l'utilisation de capteurs supplémentaires, tels qu'un accéléromètre pour la redondance matérielle afin de détecter les collisions. Le cahier des charges spécifiait également l'utilisation de capteurs de proximités, comme des capteurs à ultrasons ainsi que des capteurs laser afin de prévoir les collisions et ainsi les éviter. Enfin, pour terminer le projet, il aurait fallu déterminer le modèle du robot afin de le pouvoir le commande en cinématique inverse, c'est-à-dire gérer la position de la pince par rapport à la base.

b) Améliorations Envisageables

Ce projet, qui se présente comme transversal, offre de nombreuses perspectives d'améliorations. En effet, pour mettre en œuvre la cinématique inverse, on peut imaginer que le robot puisse être contrôlé via un joystick Arduino. Des compétences en réseau pourrait être mise en œuvre en commandant le robot de façon distante via une RaspberryPi. On pourrait essayer de mettre en place un mode apprentissage où le robot apprend de lui-même une position dans laquelle il est, puis lui faire apprendre des tâches complètes en lui faisant mémoriser chacune des positions pour la réaliser. Des projets de traitement d'images peuvent également être pensés. En installant une caméra sur le robot, ce dernier pourrait chercher un objet prédéfini puis essayer d'aller le chercher.

La conception du robot peut également être améliorée. Le département de Conception Mécanique de Polytech Lille travaille déjà sur un PFE visant à son amélioration. Les aspects motorisation peuvent ainsi être revus par le département IMA afin de, par exemple, trouver un schéma électrique ou revoir les moteurs afin de limiter leur échauffement.

CONCLUSION

A la fin du temps alloué au projet, il reste encore du travail à réaliser. En effet, un élément bloquant a induit un retard notable sur l'avancement du projet. Néanmoins, en omettant ce retard, j'ai réussi à avancer au rythme imposé par mon calendrier prévisionnel. Le robot est conçu de manière pérenne et complète, ce qui permettra au projet d'être repris dans les prochaines années. Ce projet propose de nombreuses suites sur de nombreux domaines différents.

Ce projet s'est montré complet, m'amenant à mobiliser de nombreuses compétences autour de la mécatronique : j'ai eu à utiliser des compétences en conception mécanique et en électronique, des compétences en informatique et en automatique ont été mobilisée dans la deuxième partie du projet. On peut donc dire que ce projet, en plus d'être innovant, est très formateur.

Le lieu où je travaille la plupart du temps, le Fabricarium de Polytech, est un open-space où je rencontre beaucoup de personnes d'horizons différents qui m'ont porté conseils, m'ont aidé ou m'ont questionné et ce qui a rendu le projet encore plus enrichissant.

BIBLIOGRAPHIE

Informations sur la cobotique :

https://projets-ima.plil.fr/mediawiki/index.php/P25_D%C3%A9veloppement_d%27un_cobot

http://www.humarobotics.com/la-robotique-collaborative/

http://www.mb-s.fr/robot-collaboratif-vs-robot-industriel-traditionnel.html

https://www.generationrobots.com/blog/fr/2015/01/robots-collaboratifs-et-robots-traditionnels-les-5-differences-cles/

Vidéos de présentations de différents cobots :

https://youtu.be/8N9TsiMQ_eI

https://youtu.be/S4mULTknb2I

https://www.youtube.com/watch?v=2KfXY2SvlmQ

Informations sur la librairie AccelStepper :

http://www.airspayce.com/mikem/arduino/AccelStepper/

https://github.com/adafruit/AccelStepper

Base de la programmation du bras :

http://www.robot-maker.com/forum/topic/11216-bras-robot-bcn3d-moveo/page-12

ANNEXES

- **Définition des PINs Ramps 1.4**

```
#define X_STEP_PIN         54
#define X_DIR_PIN          55
#define X_ENABLE_PIN       38
#define X_MIN_PIN           3
#define X_MAX_PIN           2


#define Y_STEP_PIN         60
#define Y_DIR_PIN          61
#define Y_ENABLE_PIN       56
#define Y_MIN_PIN          14
#define Y_MAX_PIN          15

#define Z_STEP_PIN         46
#define Z_DIR_PIN          48
#define Z_ENABLE_PIN       62
#define Z_MIN_PIN          18
#define Z_MAX_PIN          19

#define E0_STEP_PIN         26
#define E0_DIR_PIN          28
#define E0_ENABLE_PIN       24

#define E1_STEP_PIN        36
#define E1_DIR_PIN         34
//#define E1_ENABLE_PIN      24   Non Connu

#define SDPOWER            -1
#define SDSS                53
#define LED_PIN             13

#define FAN_PIN          9

#define PS_ON_PIN        12

#define KILL_PIN         -1

#define HEATER_0_PIN       10
#define HEATER_1_PIN       8

#define TEMP_0_PIN         13   // ANALOG NUMBERING
#define TEMP_1_PIN         14   // ANALOG NUMBERING
#define TEMP_2_PIN         15   // ANALOG NUMBERING
```

- **Bibliothèque Personnelle Moveo : Fichier Moveo.h**

```h
#ifndef Moveo_h
#define Moveo_h

#include <Arduino.h>


class Moveo
{
    public:
        Moveo();
        void Attente();

        int Recuperation_Axe();
        void Home(int HomeB, int HomeE, int HomeC, int HomeRP, int HomeP);

        void MouvementB(int order);
        void MouvementE(int order);
        void MouvementC(int order);
        void MouvementRP(int order);
        void MouvementP(int order);

        int PositionB();
        int PositionE();
        int PositionC();
        int PositionRP();
        int PositionP();

        char Arret();

        void Retour_Repos(int posB, int posE, int posC, int posRP, int posP);
        void Retour_HOME(int posB, int posE, int posC, int posRP, int posP,
                         int HomeB, int HomeE, int HomeC, int HomeRP, int HomeP);
        void Retour_EXPO(int posB, int posE, int posC, int posRP, int posP);

        bool Deadzone(int moteur, int val, int pos);
};
#endif
```

- **Bibliothèque Personnelle Moveo : Fichier Moveo.cpp**

```
Moveo::Moveo()
{
    stepperB.setMaxSpeed(2000);
    stepperB.setAcceleration(2000.0);
    stepperE.setMaxSpeed(2000);
    stepperE.setAcceleration(2000.0);
    stepperC.setMaxSpeed(2000);
    stepperC.setAcceleration(2000.0);
    stepperRP.setMaxSpeed(500);
    stepperRP.setAcceleration(500.0);
    stepperP.setMaxSpeed(2000);
    stepperP.setAcceleration(2000.0);

    // Mise à l'origine des moteurs
    stepperB.setCurrentPosition(0);
    stepperE.setCurrentPosition(0);
    stepperC.setCurrentPosition(0);
    stepperRP.setCurrentPosition(0);
    stepperP.setCurrentPosition(0);
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void Moveo::Attente()
{
  int GO=0;
  Serial.print("Appuyez sur 1 pour lancer la phase de HOMING.\n");

  while(GO != 49)
    GO=Serial.read();

  delay(100);
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


int Moveo::Recuperation_Axe()
{
  char x[1]={0};
  int nbr = 0;

  while(Serial.available()>0)
  {
    x[0] = Serial.read();
    delay(100);
  }

  sscanf(x,"%d",&nbr); //si valeur décimale: conversion du tableau de char en int

  if(x[0] == 'R')
      return 10;

  if(x[0] == 'H')
    return 9;

  if(nbr<6 && nbr>0)
      return nbr;

  if(x[0] == 'X')
      return 7;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
void Moveo::Home(int HomeB, int HomeE, int HomeC, int HomeRP, int HomeP)
{
    int HomeBase     =  -HomeB * 43.33;
    int HomeEpaule   =  -HomeE * 24.22;
    int HomeCoude    = HomeC * 26.42;
    int HomeRPoignet =  HomeRP* 4.66;
    int HomePoignet  = HomeP * 19.00;

    bool stop = false;

    stepperB.moveTo(HomeBase);
    stepperE.moveTo(HomeEpaule);
    stepperC.moveTo(HomeCoude);
    stepperRP.moveTo(HomeRPoignet);
    stepperP.moveTo(HomePoignet);

    while(stop!= true)
    {
        stepperE.run();
        stepperB.run();
        stepperC.run();
        stepperRP.run();
        stepperP.run();

        if(stepperB.currentPosition()== HomeBase && stepperE.currentPosition()== HomeEpaule
            && stepperC.currentPosition()== HomeCoude &&
            stepperRP.currentPosition() == HomeRPoignet
            && stepperP.currentPosition() == HomePoignet)
            stop = true;
    }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void Moveo::MouvementB(int order)
{
  int OrderPas = order*43.33; // conversion du nombre de degrés en nombre de pas
  int target = stepperB.currentPosition() - OrderPas;

  while(stepperB.currentPosition() != target)
  {
    stepperB.moveTo(target);
    stepperB.run();
  }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void Moveo::MouvementE(int order)
{
  int OrderPas = order*24.22; // conversion du nombre de degrés en nombre de pas
  int target = stepperE.currentPosition() - OrderPas;

  while(stepperE.currentPosition() != target)
  {
    stepperE.moveTo(target);
    stepperE.run();
  }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
void Moveo::MouvementC(int order)
{
  int OrderPas = -order*26.42; // conversion du nombre de degrés en nombre de pas
  int target = stepperC.currentPosition() - OrderPas;

  while(stepperC.currentPosition() != target)
  {
    stepperC.moveTo(target);
    stepperC.run();
  }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void Moveo::MouvementRP(int order)
{
  int OrderPas = order*4.66; // conversion du nombre de degrés en nombre de pas
  int target = stepperRP.currentPosition() - OrderPas;

  while(stepperRP.currentPosition() != target)
  {
    stepperRP.moveTo(target);
    stepperRP.run();
  }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void Moveo::MouvementP(int order)
{
  int OrderPas = -order*19.00; // conversion du nombre de degrés en nombre de pas
  int target = stepperP.currentPosition() - OrderPas;

  while(stepperP.currentPosition() != target)
  {
    stepperP.moveTo(target);
    stepperP.run();
  }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int Moveo::PositionB()
{
    int pB = 0;

    pB = - stepperB.currentPosition();

    return pB;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int Moveo::PositionE()
{
    int pE = 0;

    pE = -stepperE.currentPosition();

    return pE;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
int Moveo::PositionC()
{
    int pC = 0;

    pC = stepperC.currentPosition();

    return pC;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int Moveo::PositionRP()
{
    int pRP = 0;

    pRP = stepperRP.currentPosition();

    return pRP;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int Moveo::PositionP()
{
    int pP = 0;

    pP = stepperP.currentPosition();

    return pP;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


char Moveo::Arret()
{
    char x[1];
    char out;

    while(Serial.available()>0)
    {
        x[0] = Serial.read();
        delay(100);
    }

    if(x[0] == 'E')
        out = 'E';


    return out;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
void Moveo::Retour_Repos(int posB, int posE, int posC, int posRP, int posP)
{
    bool stop = false;

    stepperB.moveTo(0);
    stepperE.moveTo(0);
    stepperC.moveTo(0);
    stepperRP.moveTo(0);
    stepperP.moveTo(0);

    while(stop!= true)
    {
        stepperE.run();
        stepperB.run();
        stepperC.run();
        stepperRP.run();
        stepperP.run();

        if(stepperB.currentPosition()== 0 && stepperE.currentPosition()== 0
            && stepperC.currentPosition()== 0 &&  stepperRP.currentPosition() == 0
            && stepperP.currentPosition() == 0)
            stop = true;
    }

    Serial.println("Robot en position de repos\n");
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void Moveo::Retour_EXPO(int posB, int posE, int posC, int posRP, int posP)
{
    int movB  =  50 - posB;
    int movE  = -20 - posE;
    int movC  = -50 - posC;
    int movRP = -posRP;
    int movP  =  60 - posP;

    bool stop = false;

    stepperB.move(-movB*43.33);
    stepperE.move(-movE*24.22);
    stepperC.move(movC*26.42);
    stepperRP.move(movRP*4.66);
    stepperP.move(movP*19.00);

    while(stop!= true)
    {
        stepperB.run();
        stepperE.run();
        stepperB.run();
        stepperC.run();
        stepperRP.run();
        stepperP.run();


        if(PositionB()/43.33>=49.9 && PositionE()/24.22<=-19.50
            && PositionC()/26.42 <= -49.9 && PositionRP()/4.66 == 0
            && PositionP()/19.00>= 59.9)
            stop = true;
    }
    delay(4000);
    Retour_Repos(posB,posE,posC,posRP,posP);

}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
bool Moveo::Deadzone(int moteur, int val, int pos)
{
    int i = moteur;
    int mov = val;
    int dir = pos + mov;

    switch (i)
    {
        case 1:
        {
            if(dir>180.00 || dir<0.00)
                return true;
            else
                return false;

            break;
        }

        case 2:
        {
            if(dir>80.00 || dir<-90.00)
                return true;
            else
                return false;
            break;
        }

        case 3:
        {
            if(dir>115.00 || dir<-115.00)
                return true;
            else
                return false;
            break;
        }


        case 4:
        {
            if(dir>270.00 || dir<-130.00)
                return true;
            else
                return false;
            break;
        }

        case 5:
        {
            if(dir>115.00 || dir<-115.00)
                return true;
            else
                return false;
            break;
        }
    }
}
```