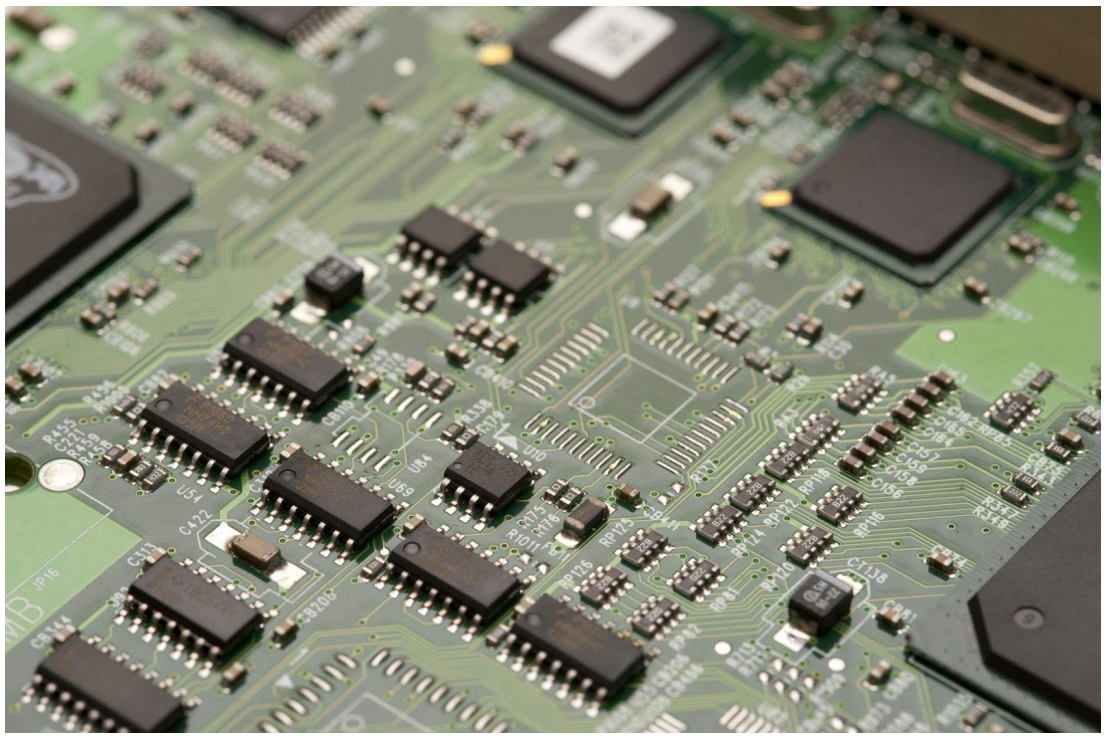
POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Rapport de Projet de Fin d'Études IMA5

P20 : Placeur de composants sur PCB



Automatic Soldering System Project

Jean Wasilewski et Pierre Letousey

Filière Informatique Microélectronique Automatique

2015 - 2016

Sommaire

Remerciements

Nous tenons à remercier chaleureusement en premier lieu nos tuteurs de projet M. Xavier Redon, M. Thomas Vantroys et M. Alexandre Boé pour nous avoir apporté leur soutien et leurs conseils avec patience et disponibilité tout au long de ce projet. Nous souhaitons également remercier très fortement M. Thierry Flamen, responsable du service électronique de Polytech Lille, pour son expertise et son apport de connaissances, nous ayant permis d'établir main dans la main les besoins et enjeux du projet.

Nous souhaitons, par la même occasion, remercier M. Rodolphe Astori, enseignant en Conception Mécanique et responsable du FabLab de Polytech Lille pour les conseils de conception qu'il a pu nous apporter. Par ailleurs, nous tenons également à remercier chaleureusement M. Antoine Urquizar, contributeur actif au Fablab de Polytech Lille, pour son partage d'expérience à propos de la création de son imprimante 3D RepRap, projet similaire au nôtre.

Enfin, nous souhaitons remercier l'atelier mécanique de PolytechLille, et plus particulièrement M. Yohann Dhondt pour l'aide qu'il nous a apportée durant l'usinage de certaines pièces. Nous souhaitons aussi remercier M. David Perraux et M. Jean-Pierre Parent pour leur aide sur la découpe des pylônes.

Présentation du projet et enjeux

1> Présentation

Le service EEI de Polytech s'est récemment équipé d'une machine de gravure. En plus de la gravure proprement dite, la machine permet le dépôt de pâte à braser pour coller les composants CMS (Montés en Surface). Après passage au four, la carte est prête (hors insertion de composants traversants qui reste manuelle).

Le but de ce projet est de réaliser une machine permettant de placer les composants suivant les fichiers fournis par le logiciel de CAO. Elle pourra s'appuyer sur une base de CNC (Computer Numerical Control). Elle permettra de récupérer des fichiers provenant du logiciel de CAO et les transformer en déplacement de la machine, de prendre les composants par aspiration, de les déposer à la bonne position avec la bonne orientation.

Afin de mener ce projet à bien, nous avons maintenu un contact constant avec M. Flamen, responsable du service EEI de PolytechLille. Il nous a permis de cibler les exigences attendues, notamment en terme de précision. Il nous a également apporté un grand apport de connaissances dans les divers domaines en relation avec notre projet.

2> Cahier des charges

En accord avec cette présentation, nous avons établi le cahier des charges suivant. Notre machine doit être capable de prendre un composant. Pour cela, il sera nécessaire de stocker les composants à placer et d'identifier leur position. Ensuite, il sera nécessaire de maintenir le composant durant le déplacement par aspiration.

La machine devra repérer la position de la carte nue. La solution devra permettre de déplacer les composants par translation selon les trois axes. De plus, il sera aussi nécessaire de permettre une rotation selon l'axe Z afin d'orienter correctement le composant sur les pastilles.

Durant la dépose du composant, il sera obligatoire de s'assurer que l'outil d'aspiration n'endommage ni la carte, ni le composant lors de la dépose. Pour cela, il s'agira de détecter le contact entre l'aiguille et le composant/carte.

Enfin, il sera nécessaire de connaître la position désirée pour la dépose du composant. Pour cela, la solution devra permettre de récupérer les fichiers provenant d'un logiciel de CAO et de les transformer en déplacements machines. Pour permettre cela, le projet devra fournir une interface visuelle (sur ordinateur) indiquant les éléments de la machine et permettant leur calibrage.

3> Liste des tâches

Pour avoir une meilleure visibilité au cours de notre travail, nous avons choisi de séparer les différents objectifs en tâches. Nous avons donc été amenés à créer la liste de tâches suivante.

- Créer l'infrastructure mécanique de la machine (châssis)
  - Fabrication du châssis
  - Design des pièces mobiles et impression 3D de ces pièces au FabLab (chariot comportant l'outil d'aspiration et "guides tubes" assurant la transformation de la rotation des moteurs en mouvement de translation (glissières) sur des barres métalliques)
  - Assembler l'outil d'aspiration au chariot
- Concevoir la commande d'aspiration par pompe
- Organiser la commande des moteurs du système pour l'asservissement en position de la machine
  - Étude et prise en main des drivers pour les moteurs Pas à Pas
- Réalisation de l'interface commande-puissance
- Réaliser la conversion des données de positions (relatives) de l'emplacement souhaité des composants en mouvement pour la machine
  - Concevoir l'interface de récupération des données du fichier Gerber de l'utilisateur
  - Caractériser des différents repères (repère de la machine, repère de l'outil et repère de la carte) et établir le lien entres eux
  - Détecter l'origine du repère de la carte au moyen d'une caméra fixe

Travail effectué sur le projet

1> Fabrication de l'infrastructure mécanique

Notre projet comprend une partie extrêmement importante de conception. Sur conseils de M. Astori, nous avons donc décidé d'utiliser une organisation particulière. Le principe de cette organisation est de créer des pièces (on parle ici uniquement de leur impression à l'imprimante 3D et de la phase de conception sous Catia) rapidement, avec une faible taux de remplissage (de l'ordre de 20 à 30%). Le but de cette méthode est de créer un nombre de pièces plus important, permettant plus facilement rectifications en fonction des remarques que nous tirons du montage.

a) La partie usinage

Nous avons commencé par acheter une support contre-plaqué en bois. Sur ce support, nous avons disposé quatre pylônes reliés deux à deux par un axe métallique. Nous avons donc découpé à la scie à chantourner plusieurs linteaux de bois pour fabriquer les pylônes. Ensuite, nous avons usiné les pylônes à l'aide d'une perceuse à colonne pour leur permettre de recevoir l'axe métallique.

Figure 1 : Usinage des pylônes à la perceuse à colonne

Ensuite, nous avons eu la surprise de recevoir des poulies non percées. Nous avons donc dû les percer. Nous avons percé les deux premières à l'aide de la perceuse à colonne. Le désavantage de cet outil est son imprécision sur la largeur de perçage. De plus, il n'était pas possible d'y ajouter un mécanisme permettant son maintien au moteur. Fort heureusement, nous avons eu la chance d'être aidés par l'atelier mécanique. Ainsi, nous avons usiné les deux dernières poulies à l'aide d'un tour conventionnel (sous la supervision du responsable de l'atelier mécanique). Nous

avons ensuite, toujours avec l'aide de M.Yohann Dhondt, percé les deux poulies selon un axe perpendiculaire au précédent et créé un filetage pour permettre l'insertion d'une vis sans tête pour assurer la fixation au moteur.

Figure 2 : Tour conventionnel utilisé pour le perçage

Pour des raisons pratiques et esthétiques, nous avons demandé à la menuiserie de Polytech Lille de corriger nos pylônes pour assurer leur perpendicularité. Cette correction a été effectuée par M. Perraux et M. Parent à l'aide d'une scie à bande. Par la suite, nous avons marqué le support pour faciliter le perçage et l'ajustement des pylônes. Une fois les marquages, nous avons percé le support et les pylônes pour assurer une bonne fixation.

Cependant, nous avons été contraints de mettre en place de nouveaux pylônes qui, pour des questions d'organisation, n'ont pas pu être rectifiés par le service menuiserie. Nous avons donc essayé de les découper au mieux en prenant en compte leur torsion.

b) La partie impression 3D

1) Pièces "guide axe" mobiles

Nous avons commencé par créer les premières versions des pièces mobiles se fixant sur les barres métalliques, qui permettent le déplacement selon l'axe X, et permettent d'embarquer le moteur pour le déplacement selon Y. Nous les avons créées à l'aide du logiciel Catia. Les cotes imposées sur le schéma sont celles fournies par la documentation et par mesure des axes métalliques au pied à coulisse.

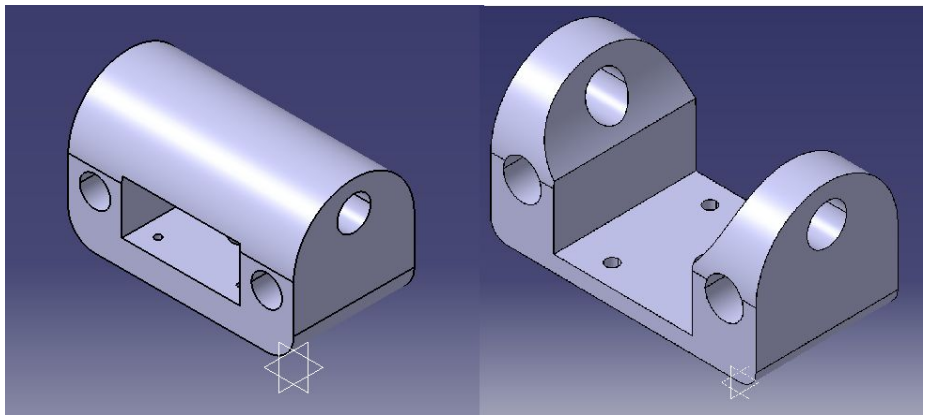


Figure 3 : Version 1 et 2 du guide axe avec moteur

Après tests sur la maquette, nous avons constaté les problèmes de frottements et de résistance à l'avancement que ces pièces engendrent sur certaines parties de nos axes métalliques. En effet, nos deux axes X ayant un parallélisme imparfait, le choix de notre liaison pivot glissant en bloquant la rotation autour de l'axe n'est pas adapté.

Nous avons donc modifié ces pièces en transformant la liaison "tube - cyllindre" par une liaison "roue incurvée - cylindre", à la manière par exemple d'un rail de roller coaster (montagnes russes). Nous avons également doublé les moteurs sur l'axe X, un entraînement est donc créé sur les deux chariots, permettant une translation selon l'axe X plus fluide et plus homogène.

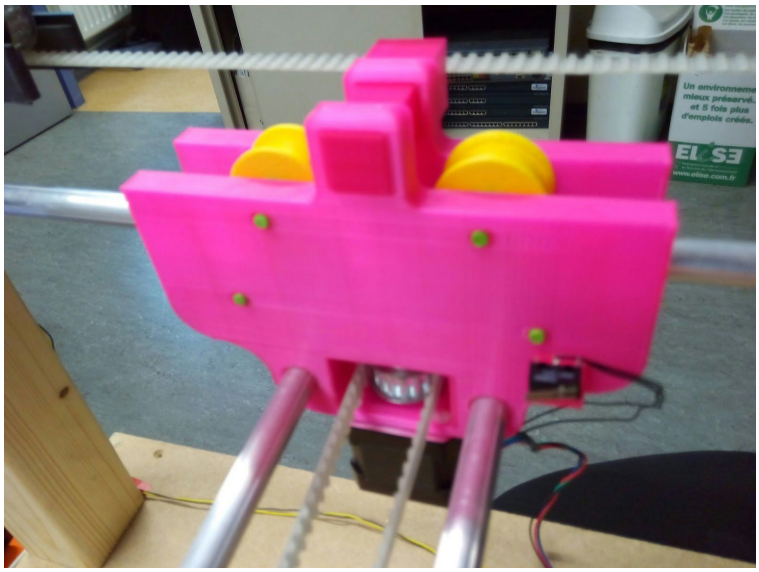


Figure 4 : Nouvelles versions des guides axes pour X

2) Conception et impression du chariot comportant l'outil

Le chariot est la partie la plus complexe de la maquette. Il doit assurer un grand nombre de fonctions. Il embarque un moteur pour la translation en Z, un moteur pour la rotation en Z, un système permettant d'adapter un préhenseur (dans notre cas: une seringue) au système à pression pneumatique et un système permettant de détecter la force appliquée pendant la pose du CMS.

Tout d'abord, son mouvement de translation avec l'axe Y est assuré par le même système de poulie que les guides axes. La translation en Z est assurée par un moteur engrènant une vis sans fin. Cette vis lie un solide qui assure alors une liaison glissière. Une pièce imprimée encastrée à ce solide nous permet de lier notre arrivée de pression négative à notre seringue.

Pour permettre une rotation selon l'axe Z, nous avons intégré un logement pour placer un moteur lié à un engrenage. Cet engrenage est destiné à venir faire tourner la pièce centrale (la pièce en rouge sur la photo suivante) à l'aide d'un engrenage interne.

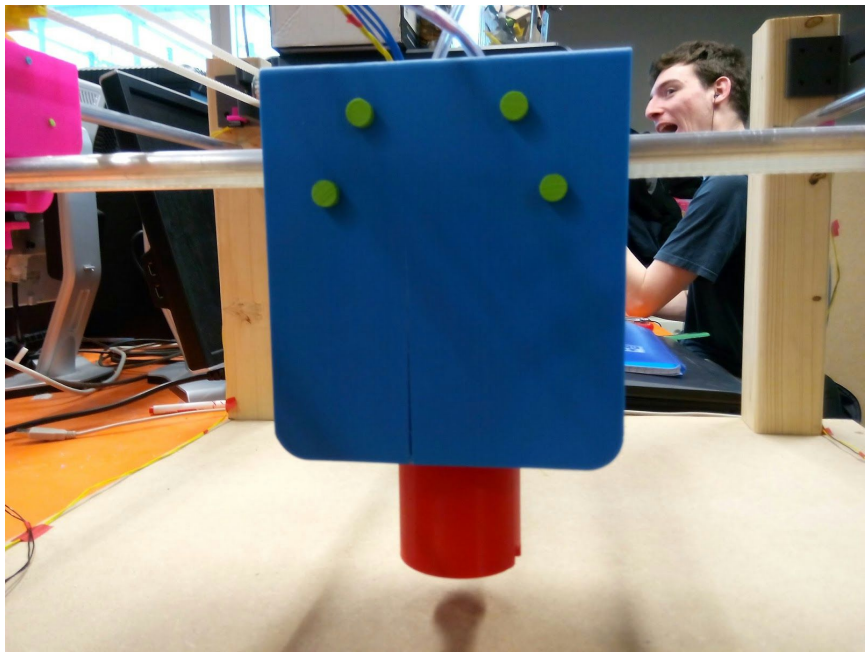


Figure 5 : Photo du chariot réalisé

Le chariot créé comporte également la solution choisie pour détecter un contact entre l'aiguille et un objet. En effet, nous n'avons pas assez de place pour intégrer des capteurs fin de course sur notre système moteur-glissière assurant la translation en Z. Cependant, ce dont nous avons besoin pour cette fonctionnalité "Détecter le contact entre le composant et la carte afin de ne pas les endommager" est réalisé par un capteur de position linéaire type potentiomètre linéaire. En effet, par un système de ressort intégré dans le chariot, nous pouvons obtenir la petite distance dont la pièce maintenant le moteur-glissière Z s'est déplacée et ainsi, la force exercée sur la carte/composant par l'aiguille.

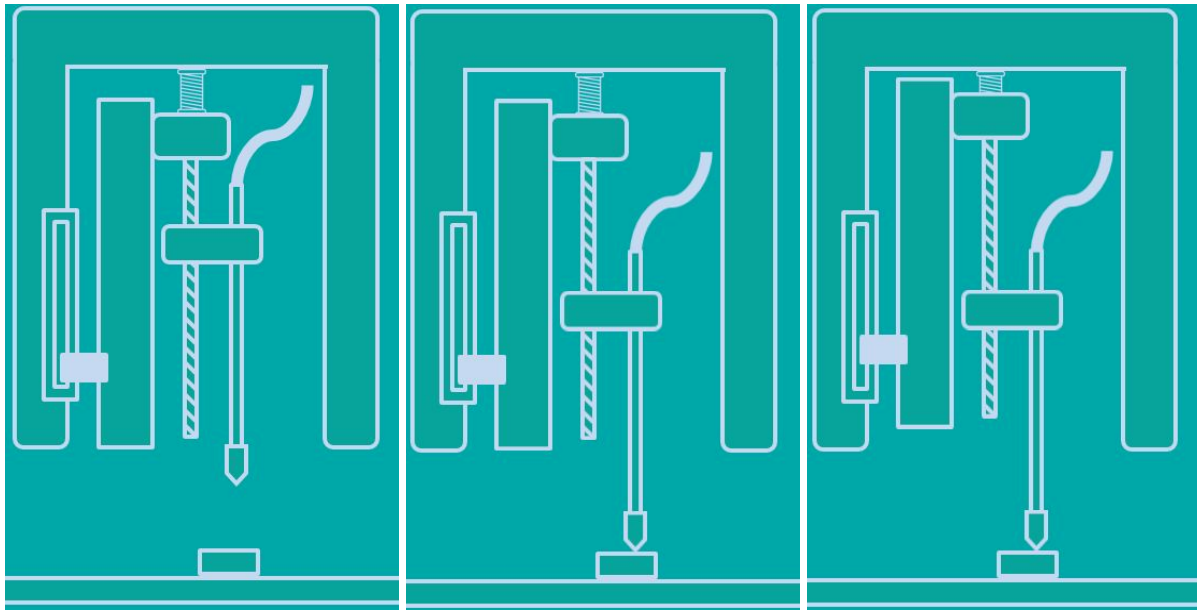


Figure 6 : Schémas explicatifs du dispositif de détection de contact

Ainsi, nous avons donc terminé de créer les éléments mécaniques et de les assembler pour obtenir notre infrastructure mécanique finale.

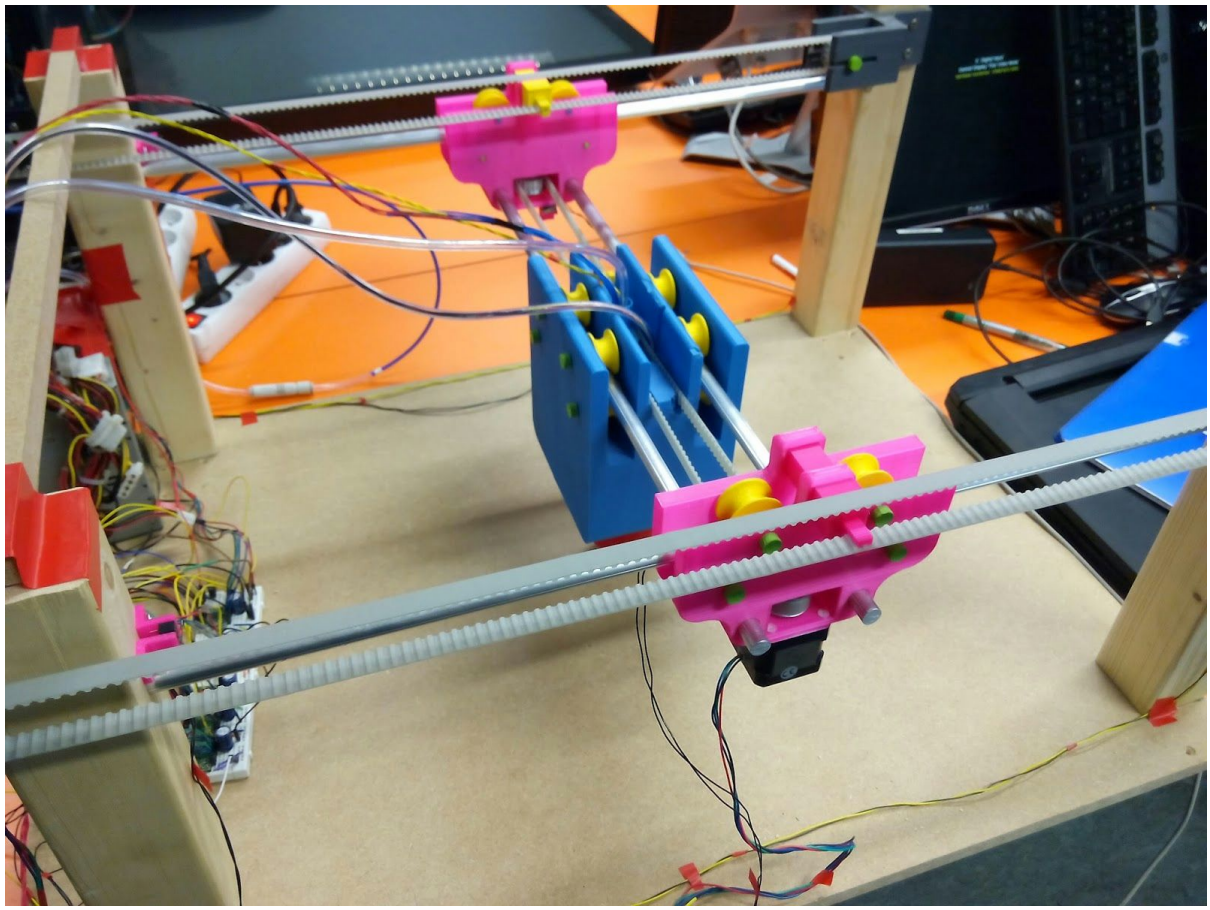


Figure 7 : Photo de l'infrastructure mécanique et des éléments assemblés

2> Motorisation et asservissement en position

a) Besoins et choix des moteurs

Nous avons besoin de déplacer notre chariot comportant l'outil d'aspiration qui manipule les composants selon les 3 axes de l'espace X, Y et Z. Afin d'assurer les translations dans le plan (X,Y), nous nous sommes orientés vers un système moteur-poulie-courroie sur chacun de ces 2 axes. Nous avons besoin d'une précision en position de l'ordre de 0,2mm et n'avons pas un besoin en couple important, ni d'un asservissement en vitesse spécifique.

Nous avons donc choisi d'utiliser des moteurs Pas à Pas, qui offrent les avantages d'être précis en terme de position angulaire, de pouvoir adapter cette précision en les commandant en micro-pas et surtout de pouvoir réaliser l'asservissement en position sans capteurs. En effet, nous nous baserons sur le comptage du nombre de pas que l'on a fait effectuer au moteur, en supposant qu'aucun pas n'est sauté durant le déplacement. En effet, dans notre application la charge a entraîner est suffisamment faible et le déplacement se fera a une vitesse suffisamment lente afin d'assurer cette hypothèse.

Le déplacement de position est alors déduit ainsi :

$$\Delta X = \mu_{micropas} * \theta_{Pas_{nominal}} * R_{poulie} * \Delta N_{Pas}$$

avec $\theta_{Pas_{nominal}}$ le pas angulaire de nos moteurs en radian (ici, 1,8°=0,031415 rad) ,$\mu_{micropas}$ le facteur de micro-pas choisi (typiquement 1/2, 1/4, voire 1/8), R_{poulie} le rayon primitif de la poulie crantée en mm (ici 7,15mm) et ΔN_{Pas} le nombre de pas demandé (dans un sens ou dans un autre).

Ceci nous assure donc en théorie, de disposer d'une résolution en déplacement d'une précision de l'ordre de **$\Delta X_{min\_théorique}$ = 0,028 mm** en commandant nos moteurs en 8ème de pas. Ceci est conforme au cahier des charges en terme de précision pour poser du CMS Classe 5, nécessitant une précision de 0,2mm, pour les déplacements en X et en Y.

Pour enfin connaître la position absolue, il s'agit de connaître la position initiale du chariot. Ceci est réalisé avec des capteurs "fin de course", type contacteurs à lamelles. En effet, nous effectuons avant le déplacement une "mise à zéro" en terme de position, en commandant le déplacement du chariot jusqu'à atteindre les contacteurs en X = 0 et en Y = 0.

Les capteurs de positions qui sont disposés de part et d'autre de chaque axe (début et fin de course), sont également utilisés en tant que sécurité : si un contact est détecté, les moteurs s'arrêtent, afin de ne pas endommager ces derniers ou la maquette.

b) Commande des moteurs Pas à Pas

Le principal inconvénient des moteurs Pas à Pas réside dans leur commandei. En effet, un moteur Pas à Pas est un moteur particulier sans balai et alimenté par une alimentation continue. Il comporte plusieurs bobines permettant de créer un champ magnétique à l'intérieur et ainsi d'orienter des aimants fixés au rotor dans la direction du champ magnétique créé, ce qui provoque la rotation de l'arbre. On utilise le terme Pas à Pas car le champ magnétique créé à l'intérieur se fait par incréments. Ainsi pour faire tourner l'arbre entièrement, il faut effectuer l'alimentation des bobines pour une position 1, puis position 2, et ainsi de suite, selon la géométrie et la technologie du moteur utilisées.

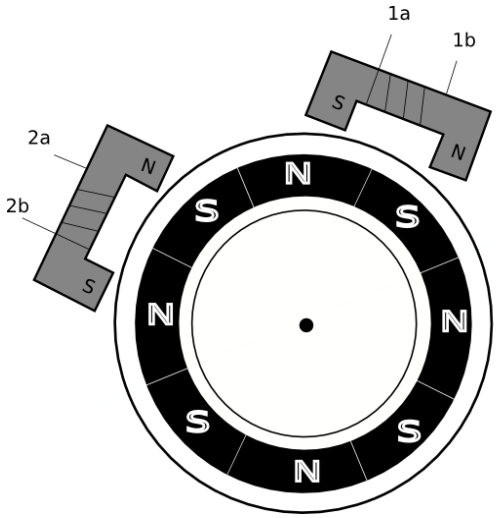


Figure 7 : Schéma symbolisant les phases et l'arbre du moteur Pas à Pas bipolaire

Ainsi, son contrôle s'effectue par une séquence d'alimentation spécifique de ses différentes phases. Dans notre cas, nous avons choisi des moteurs bipolaires (de type Hybride, c'est à dire combinant les avantages des technologies "à aimant permanent" et à "réluctance variable"). Le sens de rotation de ces moteurs dépend de l'ordre d'alimentation des 2 bobines, ainsi que du sens du courant.

Ce pilotage peut s'effectuer au moyen d'une électronique de puissance type pont en H, sur laquelle on pilote les séquences de commutations des différentes cellules de commutation (transistor + diode en sens inverse en parallèle). Cependant une telle structure est compliquée à mettre en place au niveau des séquences, car elle n'intègre pas directement de limitation active du courant délivré dans la bobine et surtout car elle n'intègre pas directement de variation de courant afin de contrôler le moteur en micro-pas.

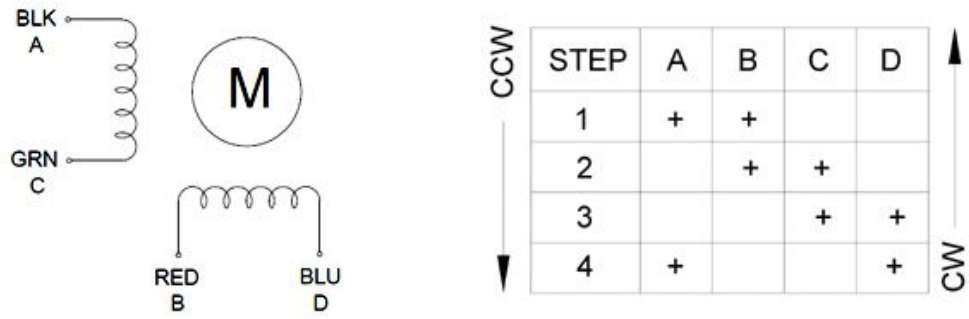


Figure 8 : Schéma de la séquence d'alimentation complète du moteur bipolaire, en pas complet

Présentant des prix très faibles (environ 5€) pour l'économie de temps et la simplicité d'utilisation qu'ils représentent, nous avons donc cherché des solutions de drivers adaptés à nos moteurs. Basé sur le circuit A4988 d'Allegro, nous nous sommes tournés vers ce driver pour moteur bipolaire (Pololu A4988 Stepper Motor Driver Carrier). Son principe est simple : il intègre directement cette structure de pont en H et les séquences d'alimentation. Il permet ainsi de faire effectuer au moteur un pas en envoyant une impulsion (front montant) sur sa pin STEP, dans le sens de rotation défini par l'état (haut ou bas) de sa pin DIR.

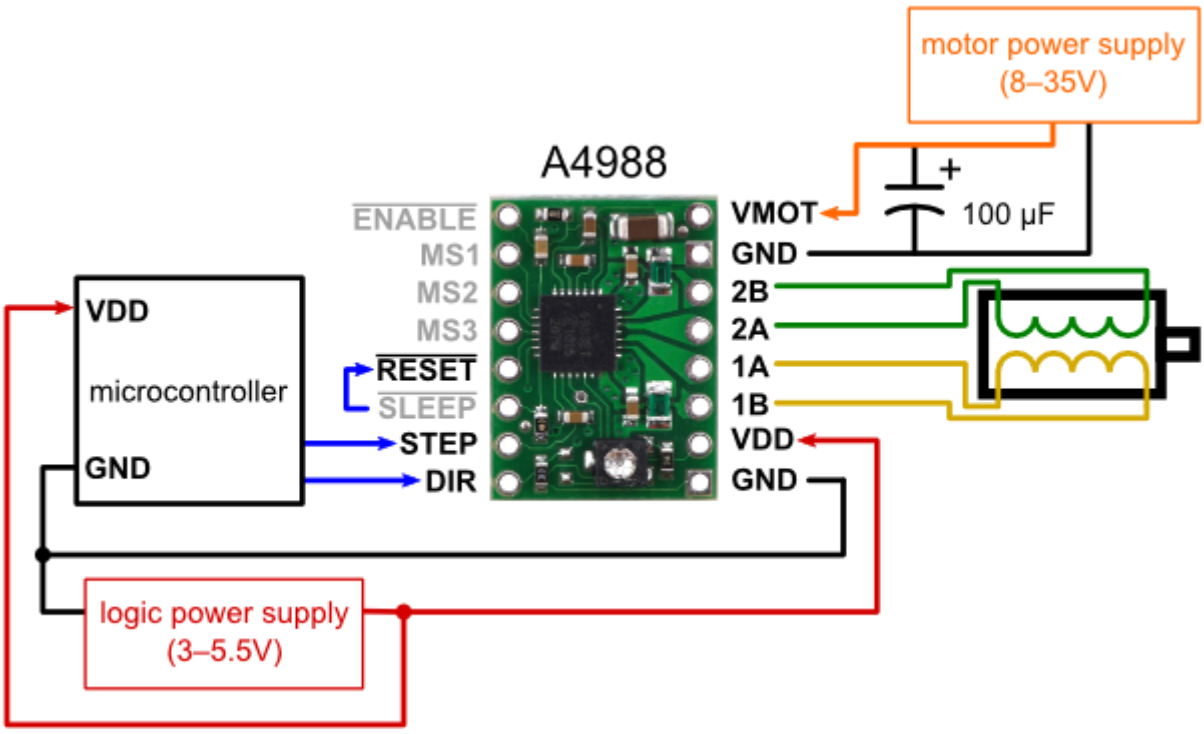


Figure 9 : Schéma du fonctionnement et du branchement du driver

Ce driver réalise également l'interface commande-puissance, il comporte une alimentation séparée pour le circuit logique (5V) et l'alimentation de puissance des moteurs (ici 12V). Il est très important pour éviter la surchauffe et l'endommagement des moteurs de contrôler le courant envoyé dans ses bobines. Nous limitons ainsi activement ce courant grâce au driver au moyen d'un potentiomètre à régler sur ce dernier. Nous avons réglé cette valeur par rapport aux courants nominaux par phase de nos moteurs.

L'avantage majeur de ce driver est qu'il comporte une solution de fonctionnement en micro-pas directement. Par l'état de ses pins MS1, MS2 et MS3, il nous permet de piloter du pas complet, au 1/16 de pas (maximum). Nous ne piloterons pas jusqu'au 1/16 de pas pour des soucis de stabilité du couple mécanique et pour éviter de sauter des pas, 1/4 voire 1/8 de pas devrait être suffisant. Et en effet, au gré des différents tests, le pilotage en 8ème de pas a montré un fonctionnement correct et stable.

3> Ajout de l'outil de manipulation par aspiration

L'outil qui se charge d'attraper et déposer les composants est composé d'une aiguille fine qui les maintient par aspiration. Le moyen qui devait initialement créer de l'aspiration devait être une pompe à vide type pompe d'aquariophilie, en raison de la faible aspiration dont nous avons besoin et qui a l'avantage d'être très peu coûteuse.

L'inconvénient de ce genre de pompe est que son arrivée d'air (coté aspiration donc) n'est pas accessible et connectable. Il s'agit en effet d'une large membrane souple et perméable, en matière type "feutre". Nous nous sommes donc procuré un tube à effet Venturi, qui "transforme" une pression entrante positive ("souffle") en une pression sortante négative ("aspiration").

De par sa conception et par le principe physique mis en oeuvre, le tube à effet Venturi nécessite une pression et un débit suffisamment élevés, que notre pompe d'aquariophilie ne nous offre pas. Cependant, après discussion avec Thierry Flamen, la salle du service EEI possèdera dans un futur proche un raccordement en air comprimé. Ainsi, pour une utilisation au service EEI, il suffira de connecter notre câble d'arrivée d'air à ce raccordement. Dans l'attente des travaux, un compresseur à air peut tout à fait être utilisé.

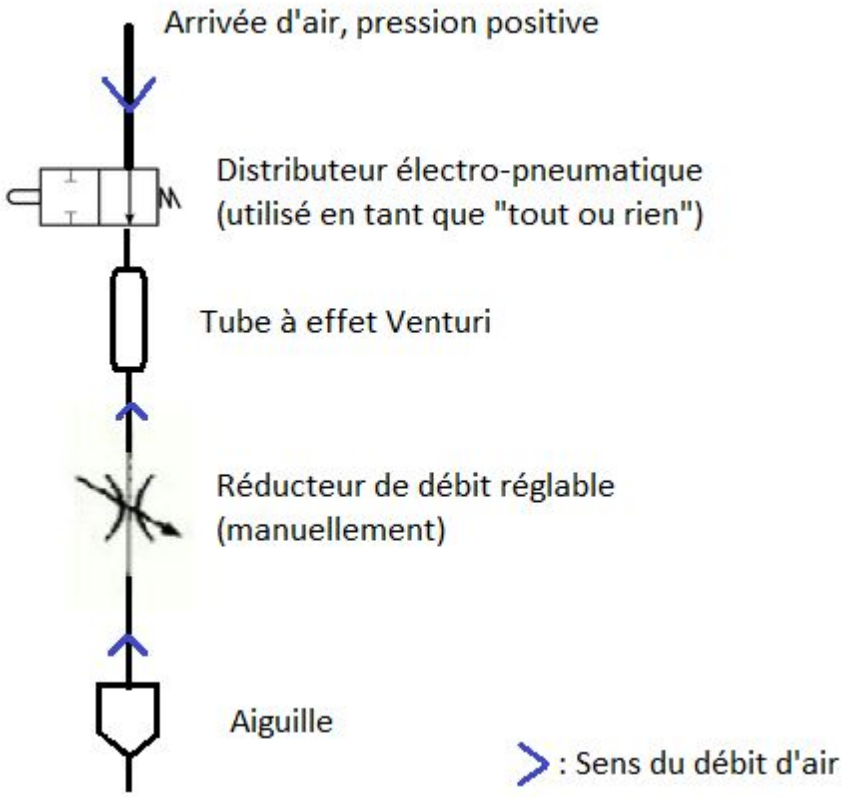


Figure 10 : Schéma de fonctionnement du circuit pneumatique

Afin de commander ou non l'aspiration dans l'aiguille, nous utilisons un distributeur électro-pneumatique. Le distributeur devant être alimenté avec une tension de 24V, nous utilisons un optocoupleur en guise d'interrupteur commandé, piloté au moyen d'une sortie d'un Arduino.

4> Commande et lecture des éléments du système via Arduino

a) Description des fonctions réalisées par l'Arduino Mega

Afin de relier les divers éléments du système à l'application pour l'utilisateur, nous utilisons un Arduino Mega. En effet, ce dernier est utilisé pour jouer le rôle d'interface entre les ordres donnés par l'application PC et le contrôle des actionneurs / lecture des capteurs.

Voici les tâches dont se charge l'Arduino en terme de sorties :

- Contrôler la commande d'aspiration des composants
- Permettre d'envoyer les ordres aux drivers qui commandent les 5 moteurs, à savoir :
    - 2 moteurs pour la translation selon l'axe X
    - 1 moteur pour la translation selon l'axe Y
    - 1 moteur-glissière pour la translation selon l'axe Z
    - 1 moteur pour la rotation autour de Z

    Pour chacun des drivers, 6 sorties logiques sont à commander :
    - 1 bit pour l'activation du moteur (Enable)
    - 1 bit pour le choix du sens de rotation (Dir)
    - 1 bit pour exécuter les pas (Step)
    - 3 bits de sélection du micro-pas (MS1, MS2, MS3)

Voici les tâches dont se charge l'Arduino en terme d'entrées :

- Lecture de l'état des 6 capteurs fin de courses (contacteurs à lamelles, appuyés ou bien non appuyés) disposés pour X1_Min, X2_Min, Y_Min, X1_Max, X2_Max et Y_Max.
- Conversion et lecture de la tension analogique, image de la position du potentiomètre linéaire, représentant la force appliquée sur l'aiguille selon l'axe Z

L'Arduino permet ainsi à l'application d'effectuer les fonctions d'écriture (surnommées 'set') qui se chargent de :

- donner une consigne de nombre de pas au moteur sélectionné, par rapport à une position d'origine
- choisir le facteur de micro-pas sur le moteur sélectionné (pas complet, ½ pas, ¼ de pas, ⅛ de pas ou 1/16 de pas)
- commander l'optocoupleur relié au distributeur pneumatique pour l'allumage ou l'extinction de l'aspiration dans l'aiguille
- choisir la limite maximale de force que l'aiguille peut appliquer sur l'axe Z. Ceci est réalisé en définissant une limite de position du potentiomètre au delà de laquelle nous arrêtons de commander le moteur Z
- choisir la vitesse du moteur sélectionné, en définissant le temps en ms que nous attendons entre deux pas
- verrouiller ou déverrouiller en position le moteur sélectionné, en envoyant ou pas le courant dans les bobines des moteurs, provoquant un couple de maintien ou non

Pour que l'application PC puisse connaître les paramètres programmés sur l'Arduino et l'état des capteurs, l'Arduino permet aussi d'effectuer les fonctions de lecture (que nous avons surnommées 'get') et qui communique :

- l'état de de verrouillage en position des moteurs
- la position d'un moteur sélectionné par rapport à l'origine
- le facteur de micro-pas du moteur sélectionné
- l'état de la consigne d'allumage de l'aspiration
- la limite maximale de force que l'aiguille peut appliquer sur l'axe Z choisie
- l'état des capteurs fin de course
- la valeur convertie par l'ADC de la tension sur le potentiomètre
- la vitesse des moteurs
- la version du logiciel utilisé sur l'Arduino Mega

Dans le fonctionnement interne de l'Arduino, il s'assure pour chaque requête de déplacement que l'action demandée est réalisable. Le programme vérifie en effet que les contacteurs ne sont pas enclenchés tout au long du déplacement pour X et Y, ou que le potentiomètre n'a pas dépassé la valeur maximale qui a été fixée. Dans le cas contraire, l'action sur les moteurs est stoppée et l'Arduino communique un code d'erreur. Un code d'erreur est aussi communiqué en cas de jeu de paramètres demandés incorrect lors d'un appel à chaque fonction.

b) Protocole de communication entre le logiciel et l'Arduino

Nous avons été amené à concevoir un protocole de communication pour assurer l'exécution des ordres et la coopération entre l'application et l'Arduino. Ils communiquent via le port série et nous avons optimisé le protocole pour le rendre le plus compact possible afin de limiter le nombre de communications. Le schéma suivant décrit le fonctionnement de l'Arduino Mega.

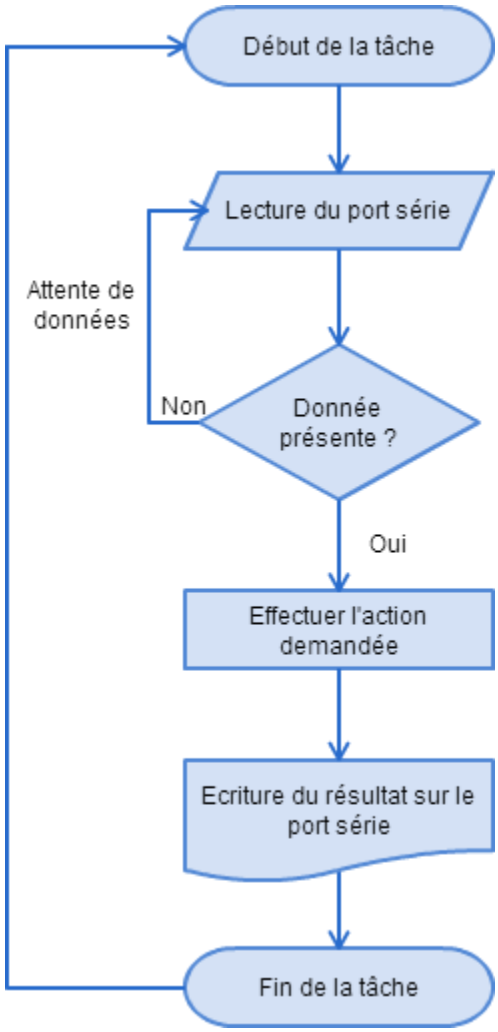


Figure 11 : Schéma du principe de fonctionnement du programme sur l'Arduino

Nous disposons de 2 types de fonctions : les fonctions d'écriture ('set') et les fonctions de lecture ('get'). Les fonctions "d'actions" ('set') retournent un code de retour à la fin de chaque action, disant si elle s'est terminée correctement ou si une erreur est survenue, comme le déclenchement d'un contact ou la saisie d'un paramètre incorrect. Les fonctions de lecture ('get') retournent quand à elles l'information demandée.

Le descriptif complet de notre protocole de communication peut se trouver à la fin de ce rapport en Annexe 1 ou sur notre Wiki de projet, rubrique Protocole.

Par ailleurs, les codes du programme C implémenté sur l'Arduino sont disponibles sur GitHub à cette adresse : https://github.com/henyxia/ASSP_Mega

5> Développement du logiciel d'interfaçage utilisateur

Nous avons créé une application permettant de contrôler facilement la maquette. En effet, la liaison avec l'Arduino Mega fonctionne parfaitement en utilisant uniquement un terminal accédant au port série. Cependant, pour des raisons esthétiques et pratiques, nous avons réalisé une interface permettant la commande en mode manuel.

a) Utilisation d'un framework multi système d'exploitation

Comme l'intégralité de notre travail est placé sous le drapeau de l'open source, nous avons souhaité permettre à n'importe qui de pouvoir compiler notre application sur son ordinateur, et cela quelque soit son système d'exploitation. Pour permettre la réalisation de cet objectif, nous avons choisi un framework parmis ceux permettant cette inter-opérabilité. Son code source est d'ailleurs disponible à l'adresse suivante: http://github.com/henyxia/ASSP

Les collections d'objets et de librairies (aussi appelés framework) multi-OS se comptent sur les doigts de la main et ont tous leurs avantages et leurs inconvénients. Parmis les plus connus (wxWidgets, GTK+, FLTK, FOX, …), nous avons choisi de prendre Qt. Ce framework a l'avantage d'intégrer des interfaces multiplateformes pour le port série et la Webcam.

Nous noterons quand même le coté négatif de ce choix: le développement de la partie du port série a pris plus de temps que prévu. En effet, la gestion du buffer pour le port série est assez mal expliquée et semble être peu indépendante du système d'exploitation. Pour palier à ce problème, nous avons choisi d'utiliser un protocole de communication octet à octet avec accusé d'émission et de réception.

b) Création d'une interface intuitive

Pour permettre à un utilisateur lambda de pouvoir utiliser notre interface, nous avons essayé de la rendre la plus intuitive possible. Pour cela, nous avons décomposé en trois zones distinctes les différentes informations. La première partie comporte une fenêtre OpenGL permettant la génération du rendu des fichiers GERBERs ouverts. Le deuxième volet est une fenêtre permettant le retour d'erreur et l'affichage d'informations sur l'état de l'application ou de la communication. Enfin, la troisième permet de visualiser en temps réel les variables de l'Arduino MEGA et d'ordonner des consignes.

Figure 12 :Découpe de zones de l'application

c) Interprétation des fichiers GERBERs

Pour permettre l'utilisation des fichiers GERBERs dans notre application, nous avons été amené à étudier la norme régissant ces fichiers. Bien que la lecture de cette norme soit difficile du fait de sa rigueur, nous sommes parvenus à en comprendre ces bases. Cependant, par manque de temps, il n'a pas été possible de mettre en place cette reconnaissance dans notre application.

Bilan des réalisations

Avant de conclure ce rapport, nous souhaitons résumer l'état final des différentes tâches qui nous ont été confiées.

- Créer l'infrastructure mécanique de la machine (châssis)

Le châssis est fonctionnel. Des améliorations peuvent être apportées en remplaçant le support bois par du métal, afin d'éviter au maximum les problèmes de compliance. De plus, les pièces conçues et imprimées à l'imprimante 3D pourraient être plus optimisées (voir pour un projet inter-départements avec des élèves en Conception Mecanique).

- Concevoir la commande d'aspiration par pompe

La commande d'aspiration est fonctionnelle et est prête à être raccordée à une arrivée d'air. Le système Venturi assure une aspiration suffisante pour déplacer des CMS.

- Réaliser la conversion des données de positions (relatives) de l'emplacement souhaité des composants en mouvement pour la machine

Cette partie n'est malheureusement pas achevé. Nous avons étudié la norme GERBER. Par manque de temps, nous n'avons pas pu faire la liaison entre ces fichiers et la commande de la maquette.
Il était prévu, pour offrir un moyen de revérifier le positionnement et pour palier aux imprécisions (type jeux sur la maquette...) sur la position initiale, la position de la poulie et son diamètre, de mettre en place un système d'ajustement via une caméra, où l'utilisateur viendrait valider le placement du composant avant dépose sur la carte. Cependant, nous n'avons pas eu le temps d'intégrer ceci dans la solution, bien qu'un espace prévu à cet effet ait été fabriqué sur le chariot. Nous n'avons pas non plus, pour les mêmes raisons, fabriqué la matrice de repérage des composants.

- Organiser la commande des moteurs du système pour l'asservissement en position de la machine

La commande en position est totalement fonctionnelle. La précision de la commande est sans erreurs. L'imprécision éventuelle est liée au châssis mais ne devrait pas empêcher le dépôt de composants CMS.

- Réalisation de l'interface commande-puissance

La réalisation de l'interface commande puissance fonctionne parfaitement. Pour augmenter la sécurité du matériel et diminuer la complexité de la commande, le choix des drivers Polulu s'est avéré excellent.

Conclusion

Au terme d'un semestre consacré majoritairement au développement de ce projet de fin d'étude, nous sommes réellement heureux et fiers des avancées qui ont été accomplies. Les difficultés rencontrées étaient souvent de natures différentes du fait de la multidisciplinarité de ce projet. Cependant, une quantité importante de travail, ainsi qu'un suivi de la part de nos encadrants nous ont permis d'atteindre la majorité de nos objectifs.

Bien que ce projet avait pour vocation d'être intégré au terme du semestre, il reste quand même utile pour le pôle EEI. Il pourra en effet être repris comme futur projet en IMA et ainsi achever l'interface utilisateur.

Annexe

Les codes d'erreur

Commande d'initialisation d'une tâche (PC->Mega)

Bits	7	6	5	4	3	2	1	0
Data	Paramètre				Fonction appelée			

Commande de réponse d'une tâche (Mega->PC)

Bits	7	6	5	4	3	2	1	0
Data	Donnée dépendant de la fonction				Code de retour			

Liste des codes de retour

Code retourné					Nom dans le header	Action à prendre	Description du code
b3	b2	b1	b0	Hex			
0	0	0	0	0x00	CMD_OK	Aucune	La commande s'est exécutée correctement
0	0	0	1	0x01	CMD_NOT_KNOWN	Report immédiat	La commande demandée n'est pas connue
0	0	1	0	0x02	CMD_DEST_UNREACHABLE	Report immédiat	L'appel à la commande setDest a le paramètre *dest* trop grand sur le moteur choisi
0	0	1	1	0x03	CMD_LOCK_MIN_X1	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe X 1 à son minimum
0	1	0	0	0x04	CMD_LOCK_MAX_X1	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe X 1 à son maximum
0	0	1	1	0x05	CMD_LOCK_MIN_X2	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe X 2 à son minimum
0	1	1	0	0x06	CMD_LOCK_MAX_X2	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe X 2 à son maximum
0	1	1	1	0x07	CMD_LOCK_MIN_Y	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe Y à son minimum
1	0	0	0	0x08	CMD_LOCK_MAX_Y	Report immédiat ou aucun	Le chariot a déclenché le contact à lamelle souple sur l'axe Y à son maximum
1	0	0	1	0x09	CMD_LOCK_Z	Report immédiat ou aucun	L'aiguille a touché un objet et a dépassé la force maximum à appliquer dessus
1	0	1	0	0x0A	WAIT_FOR_NEXT_FRAME	Report immédiat ou aucun	La fonction appelée a été comprise, l'Arduino est en attente de la trame suivante.
1	0	1	1	0x0B	WAIT_FOR_IT_1	Aucun	La réponse est partielle, il manque encore une trame.
1	1	0	0	0x0C	WAIT_FOR_IT_2	Aucun	La réponse est partielle, il manque encore deux trames.

Les fonctions de lecture

Liste des fonctions

Fonction appelée					Nom de la fonction	Paramètre 1	Paramètre 2	Description fonction
b3	b2	b1	b0	Hex				
0	0	0	0	0x00	setDest	selectedMotor[2bits]	destination[16bits]	Donne la consigne de position en seizième de pas par rapport à l'origine
0	0	0	1	0x01	setMS	selectedMotor[2bits]	microStep[3bits]	Donne la consigne du nombre de micropas par tour pour un moteur
0	0	1	0	0x02	setPump	onOff[1bit]	ø	Donne la consigne d'allumage ou d'extinction de la pompe à vide
0	0	1	1	0x03	setADC	adcLevel[10bits]	ø	Donne la consigne d'allumage de la valeur maximale lisible par l'ADC
0	1	0	0	0x04	setRelease	lockToRelease[3bits]	ø	Donne la consigne de déverrouillage d'un nœud précédemment verrouillé
0	1	0	1	0x05	setSpeed	selectedMotor[2bits]	delayStep[8bits]	Donne le temps en ms à attendre entre 2 pas (<=> la vitesse du moteur)
0	1	1	0	0x06	setMotLocked	lockedOnOff[1bit], selectedMotor[2bits]	ø	Verrouille ou déverrouille en position le moteur sélectionné (<=> couple de maintien ou non)

Détails sur la construction des fonctions d'écriture

Fonction appelée	Trame 1				Trame 2								Trame 3							
	b7	b6	b5	b4	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
setDest	ø		selectedMotor[1:0]		destination[15:8]								destination[7:0]							
setMS	ø		selectedMotor[1:0]		ø					microStep[2:0]			ø							
setPump	ø			onOff	ø															
setADC	ø		adcLevel[9:8]		adcLevel[7:0]								ø							
setRelease	ø	lockToRelease[2:0]			ø															
setSpeed	ø		selectedMotor[1:0]		delayStep[7:0]								ø							
setMotLocked	ø	onOff	selectedMotor[1:0]		ø															

Les fonctions d'écriture

Liste des fonctions

Fonction appelée					Nom de la fonction	Paramètre 1	Description fonction
b3	b2	b1	b0	Hex			
0	1	1	1	0x07	getMotLocked	ø	Retourne l'état du verrouillage en position de tous les moteurs
1	0	0	0	0x08	getDest	selectedMotor[2bits]	Retourne la position d'un moteur donné en seizième de pas par rapport à l'origine
1	0	0	1	0x09	getMS	ø	Retourne la consigne du nombre de micropas par tour pour tous les moteurs
1	0	1	0	0x0A	getPump	ø	Retourne la consigne d'allumage actuelle de la pompe à vide
1	0	1	1	0x0B	getADC	ø	Retourne la consigne d'allumage de la valeur maximale lisible par l'ADC
1	1	0	0	0x0C	getRelease	ø	Retourne l'état de verrouillage des nœuds
1	1	0	1	0x0D	getADCvalue	selectedPin[2bits]	Retourne la valeur convertie par l'ADC de la pin analogique sélectionnée
1	1	1	0	0x0E	getSpeed	selectedMotor[2bits]	Retourne le délai entre 2 pas choisis pour le moteur sélectionné
1	1	1	1	0x0F	getVersion	ø	Retourne la version du logiciel utilisée sur l'Arduino Mega (Actuelle: 0.1)

Détails de la construction des trames de lecture

Fonction appelée	Appel (PC->Mega)				Retour (Mega->PC)													
	Trame 1				Trame 1				Trame 2								Trame 3	
	b7	b6	b5	b4	b7	b6	b5	b4	b7	b6	b5	b4	b3	b2	b1	b0	b7	b6	b5	b4	b3	b2	b1	b0
getMotorLocked	ø				lockedRotZ	lockedZ	lockedY	lockedX	ø								ø							
getDest	ø		selectedMotor[1:0]		ø				dest[15:8]								dest[7:0]							
getMS	ø				msY[0]	msX[2:0]			msR[2:0]	msZ[2:0]			msY[2:1]				ø							
getPump	ø			onOff	ø				ø								ø							
getADC	ø				ø		adcLevel[9:8]		adcLevel[7:0]								ø							
getRelease	ø				lockMinX1	lockMaxX1	lockMinX2	lockMaxX2	ø						lockMinY	lockMaxY	ø							
getADCvalue	ø		selectedPin[1:0]		ø		adcValue[9:8]		adcValue[7:0]								ø							
getSpeed	ø		selectedMotor[1:0]		ø				delayStep[7:0]								ø							
getVersion	ø				minorVersion[3:0]				majorVersion[7:0]								ø							