Projet Tuteuré

MISE EN OEUVRE DU PRINCIPE DE NUMERISATION EN TROIS DIMENSIONS - SCANNER 3D -

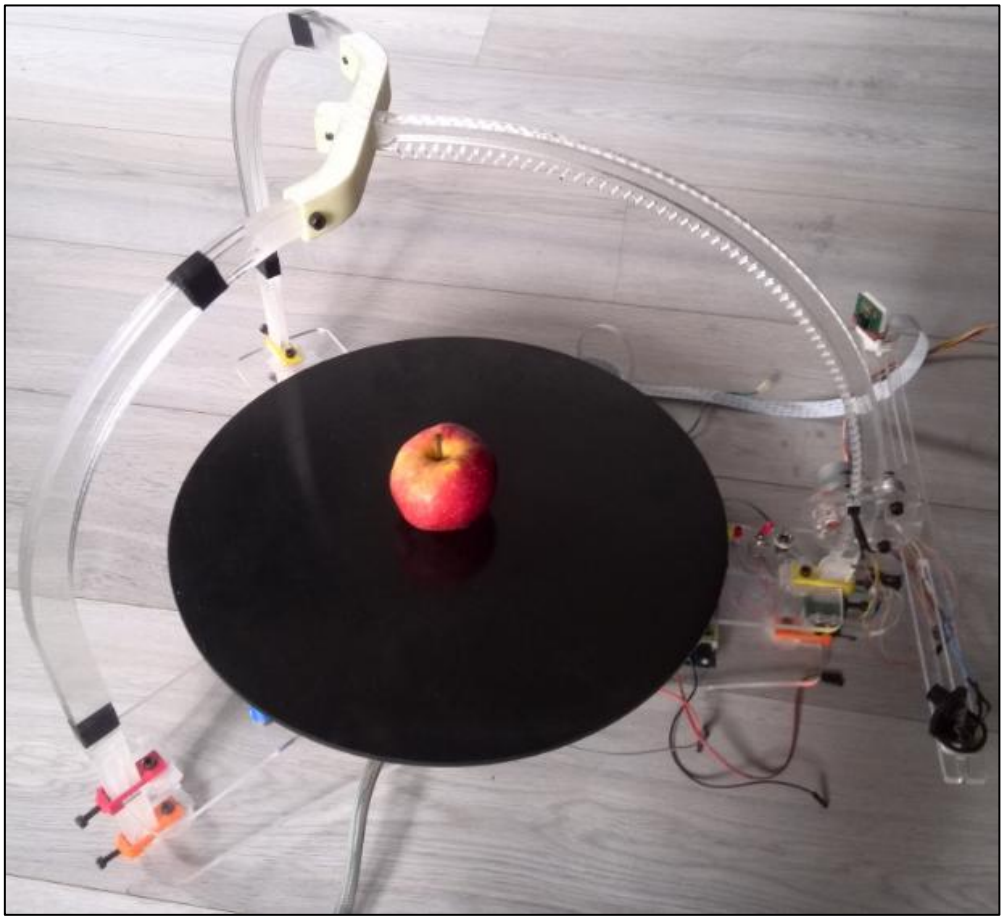
Rapport Personnel

LANIESSE Ekhi

Licence Professionnelle Mécatronique
promotion 2014-2015

L'objectif du projet consiste à mettre en œuvre le principe de scanner 3D par triangulation laser. Le principe et les différentes parties du système ont été exposés dans le rapport de présentation. Dans ce rapport d'activité du projet, je vais détailler mon travail, les différentes solutions que nous avons choisi avec Jérémy Cirot pour la partie mécanique et électronique. J'expliquerais la conception de la carte électronique. J'exposerais les programmes de traitement d'image que j'ai élaboré, la création de bibliothèques qui permettent de simplifier la programmation de l'Arduino, et les programmes qui permettent aux cartes de communiquer entre elles. Les différents problèmes rencontrés et leurs solutions seront détaillés.

Ce rapport se conclura par l'avancée du projet dans son ensemble, en effet ce projet conséquent par son niveau technique élevé n'est pas terminé. Le travail restant et les améliorations imaginables seront exposés.



Le FabLab n'a pas formulé un cahier des charges concret : nous devons réaliser le scanner en fonction de nos connaissances, et des moyens et ressources dont on dispose entre le FabLab et la formation mécatronique. Pour bien nous situer et évaluer le travail que nous devons réaliser, nous avons décidé de commencer le projet dans notre temps libre, au FabLab, le jeudi après-midi.

Au terme de cette période de recherche et d'essais, nous avons choisi la structure mécanique du scanner 3d et les éléments électroniques et informatiques qui vont composer le scanner 3D : voir l'explication détaillée dans le rapport de présentation.

I. Partie mécanique

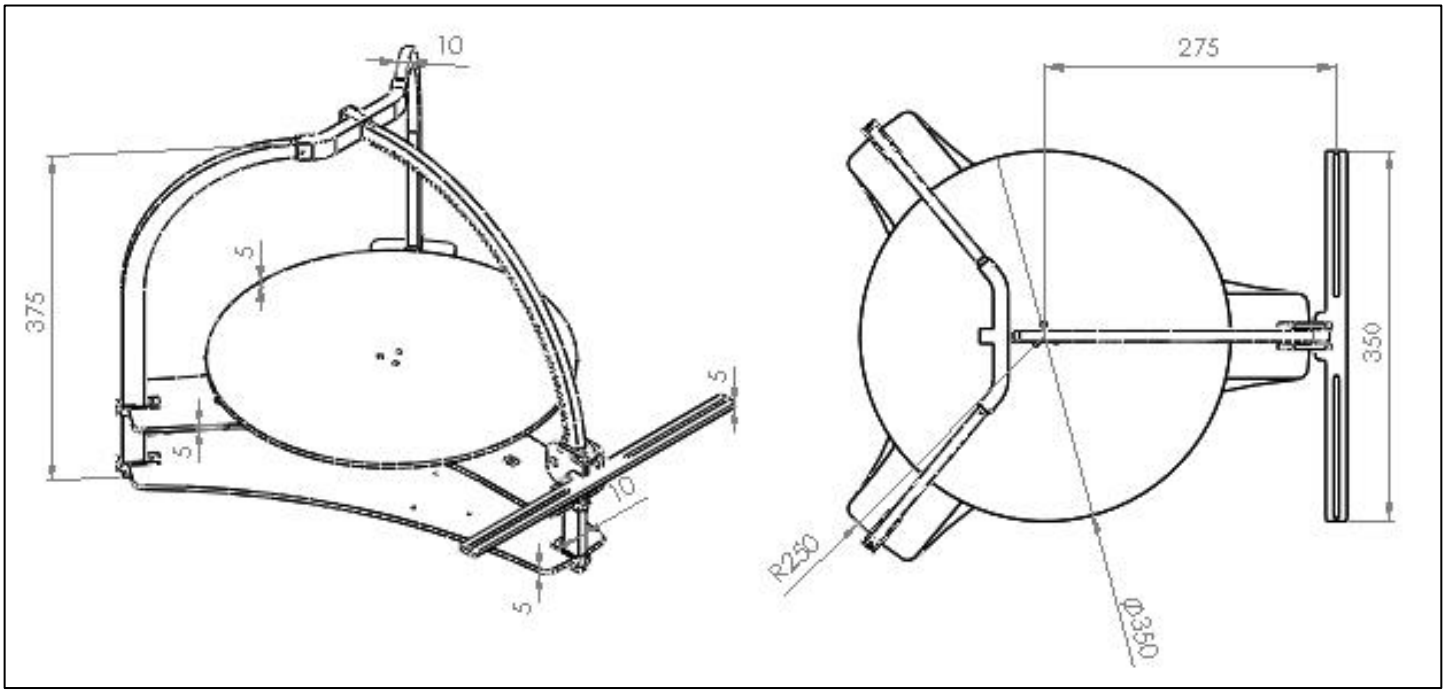
Comme exposé dans le rapport de présentation la structure est composée d'un trépied qui soutient un plateau tournant. Une des jambes du trépied soutient une réglette pour positionner le laser et la camera. Pour fabriquer les pièces nous utiliserons les machines du FabLab : la machine à découper laser et l'imprimante 3D.

Imprimante 3D MAKERBOT REPLICATOR 2X :
Surface utile du plateau : 24.6 cm x 15.2 cm
Hauteur maximale de l'objet imprimé : 15,5 cm
Matériau utilisé : ABS en filament de 1,75 mm de diamètre

Découpeuse/graveuse laser EPILOG HELIX 75W :
Surface utile du plateau : 609 mm x 456 mm
Hauteur maximale de l'objet gravé ou découpé : 229 mm
Matériaux autorisés : papier, carton, bois, cuir, verre (gravure uniquement), plexiglas ...
Matériaux strictement interdits : matières contenant du chlore.
Le laser ne peut pas découper le métal.

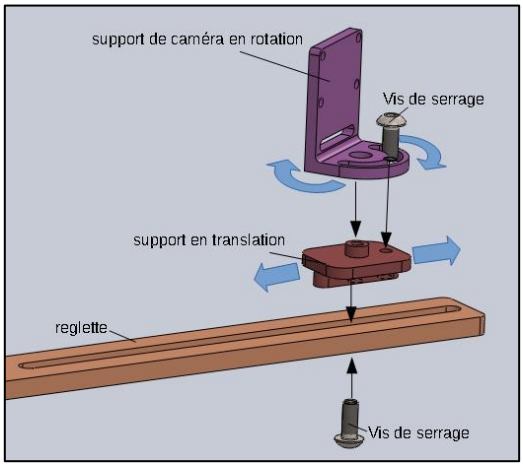
A. Conception de la structure

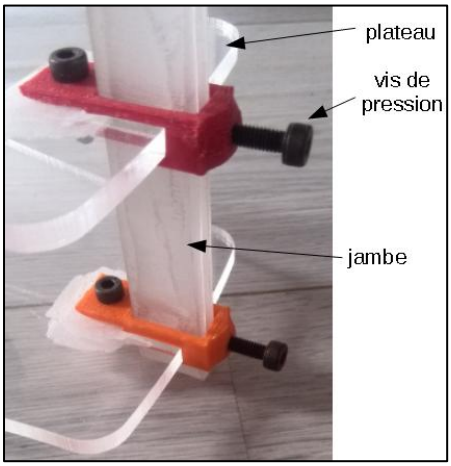
Après quelques croquis la structure est modélisée en 3D grâce au logiciel Solidworks.
Nous choisissons de réaliser les plateaux et les jambes avec la découpe laser. Les plateaux auront une épaisseur de 5mm, les jambes seront découpées dans des plaques de 10mm pour plus de solidité. Le plateau mobile aura un diamètre de 350mm. Les 3 jambes seront éloignées de 250mm du centre. La réglette qui porte le laser et la caméra sera éloignée de 275mm du centre pour garantir un bon angle de vue de la camera. Elle aura une largeur de 350mm pour garantir un angle important entre le laser et la caméra. Le scanner aura une hauteur totale de 375mm.



Sur la réglette la position du laser et de la caméra sont réglables. Une première pièce, qui s'emboite dans un trou oblong, a un mouvement de translation. Ceci permet le réglage de l'écart entre le laser et la caméra. Une deuxième pièce a un mouvement de rotation, cela permet de faire pointer le laser et la caméra vers le centre du plateau. La réglette sera découpée au laser. Les supports de caméra et laser seront imprimés en 3D.

Pour assembler toutes les pièces du châssis, dans un premier temps, nous choisissons de dessiner des encoches pour qu'elles s'emboitent entres elles. Cependant si les jambes sont emboitées aux deux plateaux nous seront obligé de démonter l'ensemble pour accéder au moteur, par exemple. Des équerres avec une vis de pression qui permettent plus de flexibilités sont dessinées. Ces pièces seront réalisées avec l'imprimante 3D.
Les jambes, de formes courbes, seront reliées entre elles en haut par une pièce imprimée en 3D aussi.

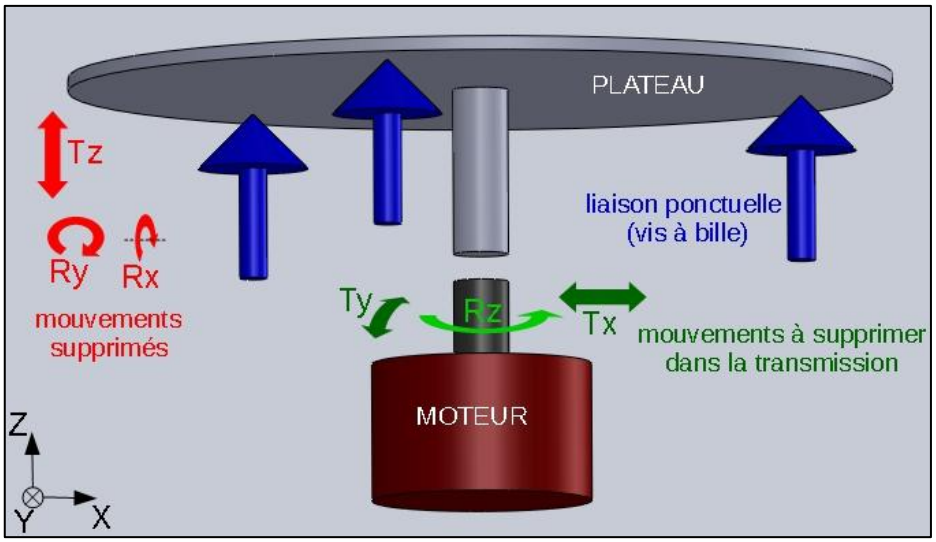


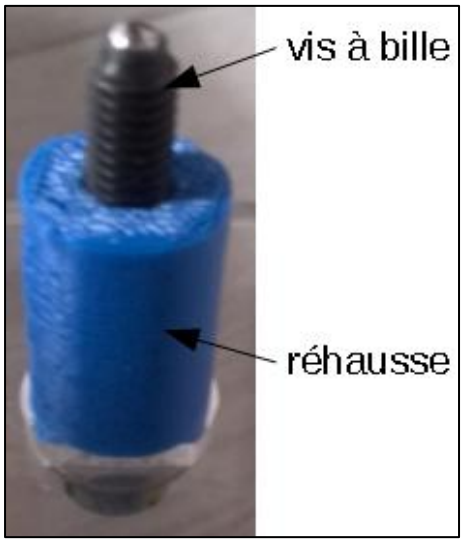


B. Accouplement moteur

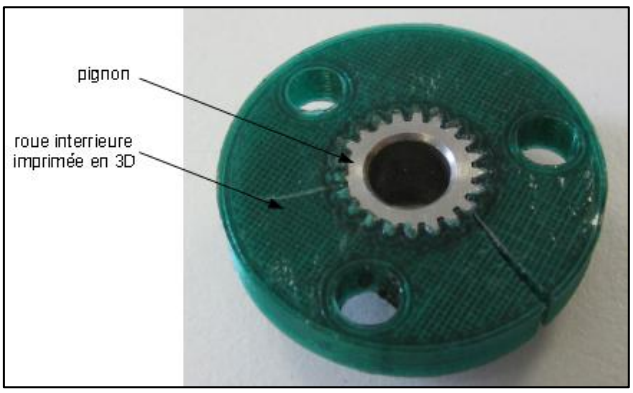
Le plateau rotatif à une grande surface. On ne peut pas le fixer directement à l'axe du moteur car il devra soutenir un poids trop important, et le plateau ne tournera pas de manière plane. Le moteur doit uniquement entrainer en rotation le plateau. Une liaison appui plan est réalisée grâce à trois appuis ponctuels à l'extérieur du plateau. Les degrés de rotation sur x et y sont supprimés, ainsi que la translation sur z. Pour réaliser ces liaisons ponctuelles nous choisissons d'utiliser des vis à billes, nous pouvons régler l'inclinaison du plateau en vissant ou dévissant chaque vis.

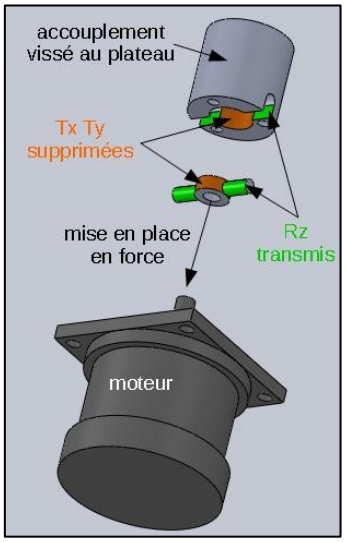
Avec l'accouplement, il reste deux translations selon x et y à supprimer, et il faut réaliser l'entrainement en rotation selon z.





Dans un premier temps, je choisis d'utiliser un pignon déjà monté et collé sur l'axe du moteur. Une roue à denture intérieure avec le même nombre de dents viendrait s'emmancher sur le pignon. Cependant, la roue intérieure imprimé en 3D est sortie plus étroite. Lorsqu'on essaye de l'emmancher, le pignon se décolle. En conséquence, je dessine un nouvel accouplement : une première pièce s'emmanche en force dans l'axe du moteur. Une deuxième pièce vient se centrer pour supprimer les translations. Deux doigts permettent l'entrainement en rotation. Ces doigts sont cylindriques pour que le plateau repose bien sur les vis à billes. Cet accouplement garantit un système non hyperstatique.





C. Chariot mobile

Comme expliqué dans le rapport de présentation, nous souhaitons rajouter une fonction :
La réglette serait mobile pour aller capturer des images du haut de l'objet à numériser. Deux solutions ont été imaginées : la réglette est fixée à un chariot qui roule sur une jambe du trépied, l'ensemble est tiré par une courroie. La deuxième solution est similaire, cependant la jambe est dentée, le chariot est entrainé par un pignon qui roule sur les dents. Le laser et la caméra doivent toujours rester à la même distance de l'origine (centre du plateau), la réglette doit donc forcément se déplacer sur un arc de cercle. La deuxième solution est choisie. Le système de courroie nous paraît compliqué à mettre en œuvre et le système de crémaillère est plus compacte, le petit moteur du kit Arduino paraît plus adapté. Je dessine les dents sur une jambe.
Pour la taille des dents, un module de 2 est choisi. C'est une taille assez élevée car lors de la découpe au laser, les dents vont demander moins de précision.
La jambe peut se voir comme le quart de cercle d'une roue intérieure.
Nous savons que $D = M \times Z$ D : diamètre en mm, M : module = 2, Z : nombre de dents
Les jambes sont éloignées de 250mm du centre du plateau, l'épaisseur sera de 10mm, le rayon intérieur est de 240mm. Le diamètre est donc de 480mm. D'où $Z = \frac{480}{2} = 240$
Nous obtenons donc 60 dents pour la jambe, ce qui correspond à un ¼ d'une roue complète. Les dents et la forme arrondie seront prolongées pour être sûr que le chariot arrive en position horizontale en bas et en position verticale en haut, il y aura donc 68 dents en tout.

Pour dimensionner la taille du pignon, nous décidons que le chariot doit monter l'arc de cercle en 15 secondes.
Pour un arc de cercle, on sait que $L = \alpha * R * \frac{\pi}{180}$ L : longueur de l'arc en mm, α : angle de l'arc en degré, R : Rayon de l'arc en mm.
$L = 90 * 240 * \frac{\pi}{180} \approx 377mm$
Le chariot doit donc parcourir 377mm en 15 secondes. Il a donc une vitesse de 25 mm/s environ.

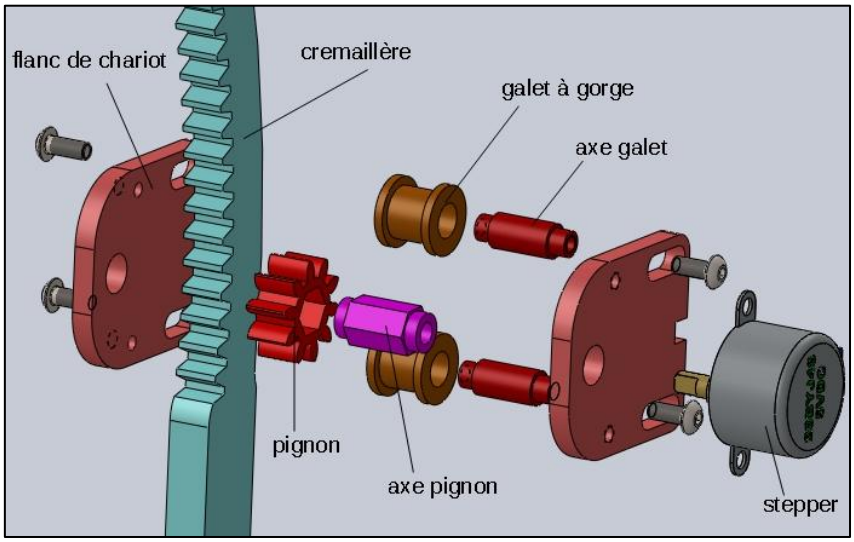
D'après la datasheet, le moteur pas à pas que nous allons utiliser dispose de 4 phases et d' un réducteur interne permettant d'obtenir 64 pas par tours. En alimentant les phases toutes les 10ms, nous obtenons une période de 40ms, donc une fréquence de 25Hz. La vitesse de rotation du moteur est de donc de :

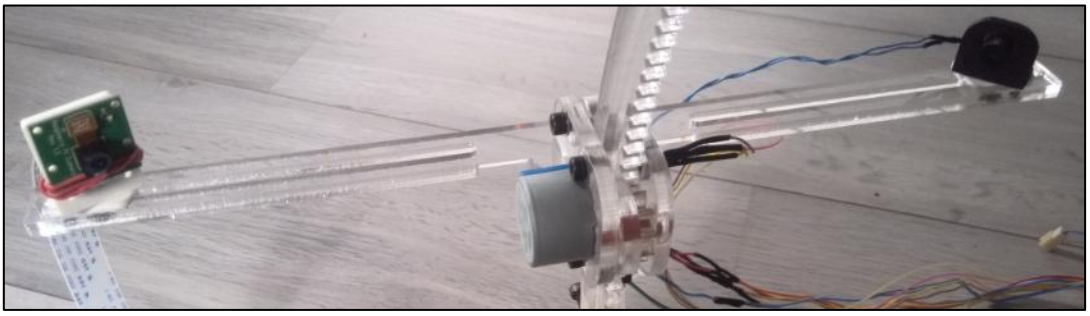
N= fréquence / pas par tour = 25/64 = 0,39 tr/s soit 2,51 rad/s.

Or $V = R \times \omega$ V : vitesse linéaire en mm/s, R : rayon en mm, ω : vitesse angulaire en rad/s

Donc $R = 25/2{,}51 \approx 10mm$. Ce qui nous donne un pignon de diamètre 20mm avec 10 dents et un module de 2.

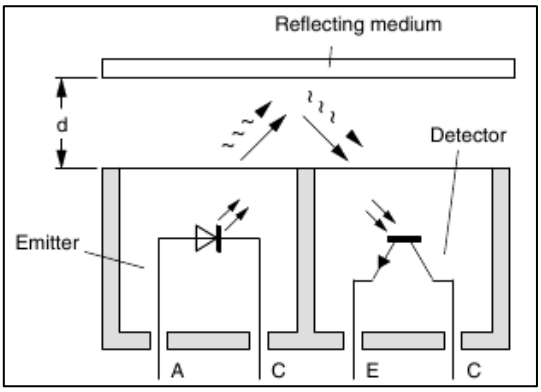
Un axe imprimé en 3D va permettre d'accoupler le moteur au pignon. Le chariot sera donc entrainé grâce aux dents sur la partie intérieure de la jambe. Deux galets à gorge reposent sur la partie extérieure de la jambe et vont permettre de maintenir le chariot. Les axes des galets viennent s'épauler aux flancs du chariot. Ils permettent d'assurer un écart entre le chariot et la jambe. Ces axes s'insèrent dans des lumières pour régler l'écart entre les dents de la crémaillère et le pignon. La réglette vient s'emboiter sur le chariot.

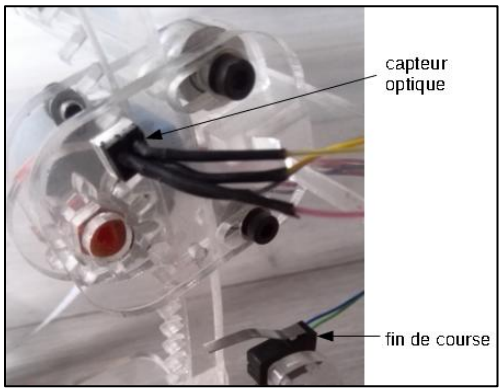




L'ensemble laser/camera peut maintenant rouler sur la crémaillère pour capturer des informations du dessus de l'objet à scanner. Lorsque nous allons piloter le moteur pas à pas, nous devons contrôler la position du chariot. La solution la plus simple consisterait à demander l'avancée d'un nombre de pas. Connaissant le nombre de pas par tours nous pouvons en déduire la distance parcourue par le chariot. Cependant il est possible que le moteur saute des pas, une dérive risque d'apparaitre à cause des jeux mécaniques. Dans un premier temps, nous

choisissons d'implanter un capteur fin de course pour contrôler la position zéro du chariot. Ensuite pour contrôler le déplacement nous avons choisi d'implanter une fourche optique, qui détecterait des marques (par exemple des perçages à 0°, 30°, 60°, 90°) sur la crémaillère. Le moteur devrait s'arrêter au niveau de ces marques. Finalement, une meilleure solution est apparue : un capteur optique réfléchissant va détecter le passage devant chaque dent de la crémaillère. Ce capteur est composé d'un émetteur : une diode infrarouge, et d'un récepteur : phototransistor. Si le signal est réfléchit (lorsqu'il il y a une dent), le phototransistor devient passant. Sinon (pas de dent) le transistor est bloqué.





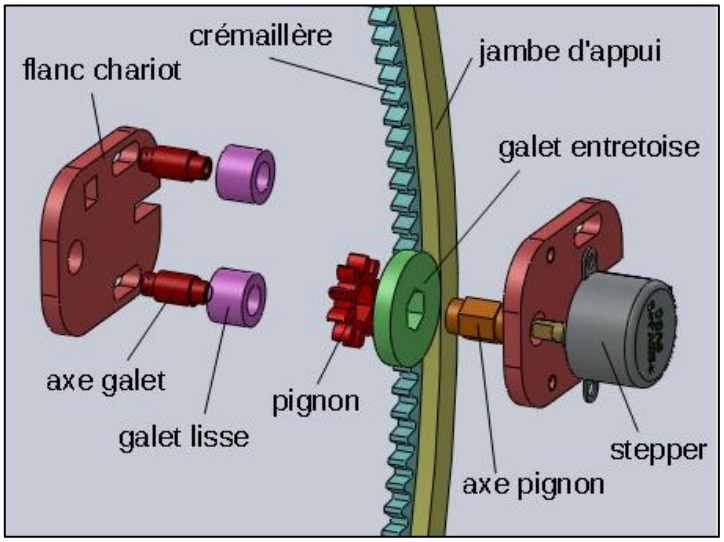
Grâce à cette méthode nous pourrons simplement, au niveau du programme, demander la montée ou la descente du chariot tant que la consigne n'a pas été atteinte.

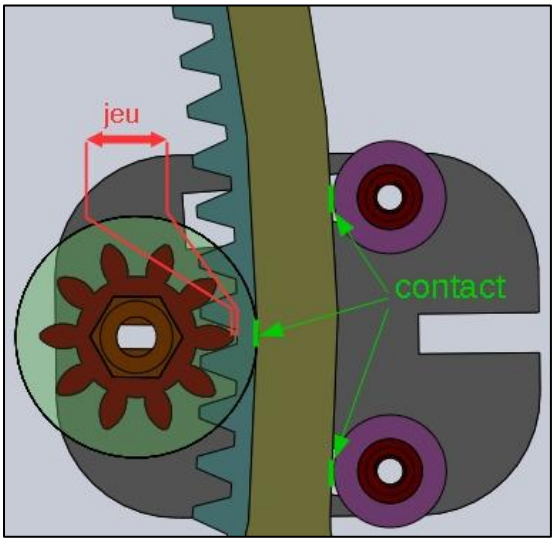
D. Réalisation

Nous rencontrons des problèmes lors de la réalisation des pièces imprimées en 3D : certaines pièces seront usinées (explication détaillée voir rapport Cirot).

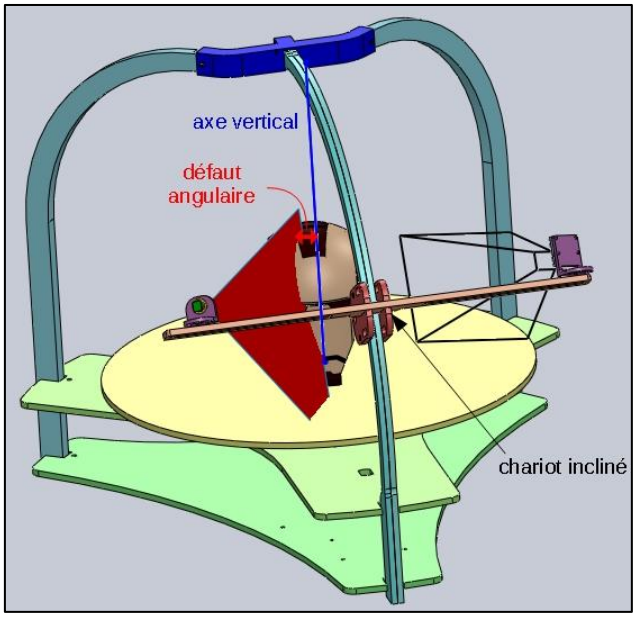
Au niveau du chariot, il y a trop de jeu, il se positionne en travers et se met en porte-à-faux. Si nous serrons les galets, le pignon s'enfonce trop dans les dents de la crémaillère et le moteur se bloque. Effectivement, dans un engrenage, la tête du pignon ne doit pas être en contact avec le pied de la crémaillère, il faut un jeu recommandé de 0,04 x module. Une solution est trouvée pour garantir ce jeu et un entraxe suffisant entre la crémaillère et le pignon. La crémaillère de 10mm d'épaisseur est divisée en 2 pièces de 5 mm. La première partie est conservée. Les dents sont supprimées sur la deuxième partie, elle est lisse. Sur l'axe moteur, une roue est ajoutée contre le pignon. Cette roue est en contact avec la partie lisse de la crémaillère, elle permet de garantir le bon entraxe entre la partie dentée de la crémaillère et le pignon.

Après modification, le système de chariot fonctionne mieux, de plus nous avons réduit le jeu transversal, les flancs du chariot sont plaqués contre la crémaillère.





Lors de la conception nous constatons un autre défaut : lorsque la réglette s'élève sur la crémaillère, elle est inclinée par rapport à l'horizontale car elle pointe vers le centre du plateau. Le laser génère une ligne verticale, mais si la réglette est inclinée, la ligne du laser n'est plus verticale. Le même problème apparaît pour la caméra, l'image ne sera pas capturée à la verticale.



En conséquence, le calcul des points sera erroné. Ce problème n'arrive pas lorsque le chariot est en position haute, à 90°, nous choisissons donc de conserver cette fonctionnalité. De plus nous pensons que ce déphasage angulaire est peut-être compensable grâce à un programme informatique. Il faudrait faire pivoter les images en fonction de l'angle du chariot.

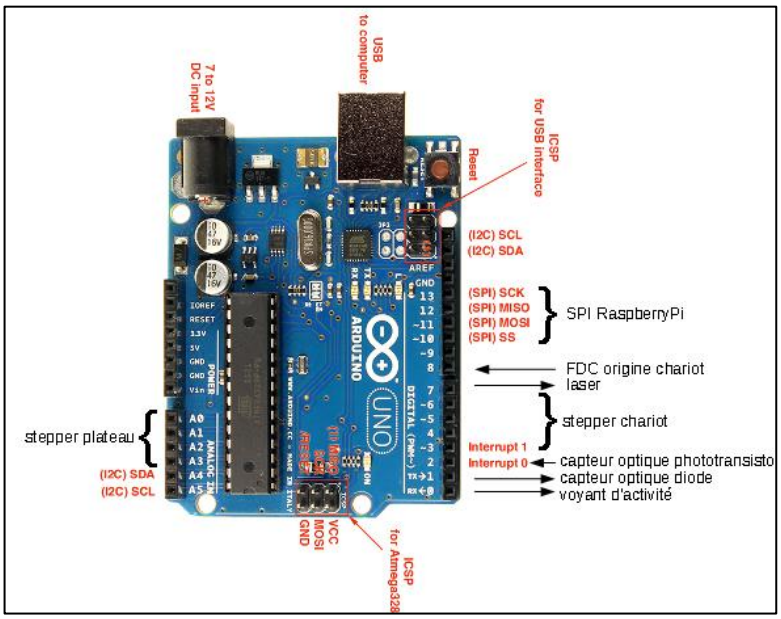
II. Partie électronique

Comme exposé dans le rapport de présentation, pour la partie contrôle/commande nous avons décidé de piloter les actionneurs grâce à un microcontrôleur Arduino. L'Arduino va recevoir des ordres depuis le micro-ordinateur RaspberryPi qui gère la caméra et qui exécute les algorithmes de traitement d'image.

Les actionneurs étant des composants de puissance nous ne pouvons pas les alimenter directement par l'Arduino. L'Arduino est conçu pour recevoir un shield (ou carte fille) et étendre ces fonctionnalités. Il faut donc concevoir un shield avec les broches pour le plugger sur l'Arduino. Ce shield sera composé de transistors pour piloter le laser et les moteurs pas à pas. Les actionneurs et les capteurs sont déportés, il y aura donc des connecteurs pour chaque actionneur. Le shield contiendra aussi l'arrivée d'alimentation avec un bouton et un voyant on/off directement relié à l'alimentation ainsi qu'un voyant d'activité relié à l'Arduino.
Enfin le shield permettra de relier l'Arduino à la RaspberryPi pour la communication entre les deux cartes.

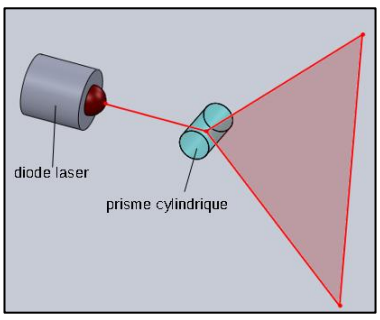
A. Conception du circuit électronique :

Entrées et sorties Arduino :



a) Laser

Le laser générateur de ligne est composé d'une diode qui émet de la lumière monochromatique (rouge). La diode émet un faisceau, en projetant cette lumière on obtient donc un point. En observant le laser on peut voir que le dispositif permettant de générer une ligne est en fait un prisme de forme cylindrique.

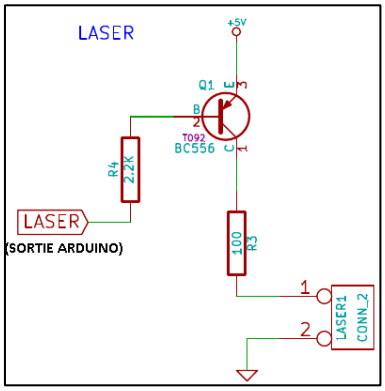


En alimentant la diode progressivement, la tension de seuil et le courant nominal sont déterminés :

Tension : 2.5V Courant : 30mA

La tension d'alimentation est de 5V, il faut donc insérer en série une résistance pour créer une chute de tension de 2.5V.

$R = \frac{U}{I} = \frac{2.5}{0.03} = 83.3\Omega$ Nous prendrons une résistance de 100ohms.



D'après la datasheet, le transistor à un gain de 150 pour Ic=30mA et Vce=5V

On sait que $Ic = Ib \times \beta$. Pour que le transistor sature il faut que $Ib \geq \frac{30}{150} \rightarrow Ib \geq 0.2mA$

La résistance à insérer pour limiter le courant de base est donc de $R = U/I = 5/0.0002 = 25k\Omega$. Nous avons testé ce montage avec une résistance de 22kΩ, cependant, comme nous l'a conseillé Eric Normandin, pour être sûr que le transistor sature nous prendrons une résistance plus faible : 2,2kΩ. Ib est donc égal à 2mA.

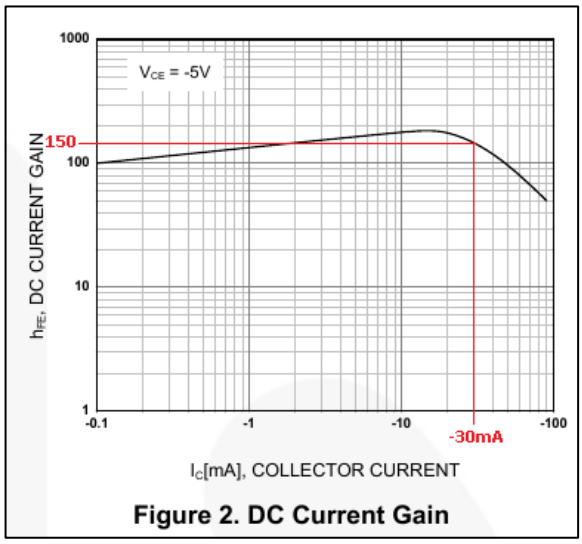


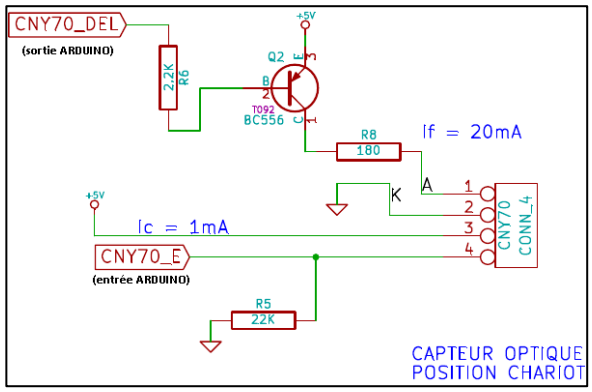
Figure 2. DC Current Gain

Dans ce montage, le transistor est saturé, donc passant, lorsque la sortie Arduino est à 0V, c'est un montage en logique négative, il faut mettre la sortie à 0V pour allumer le laser.

b) Capteur optique

Comme expliqué précédemment, le capteur optique est composé d'une diode et d'un phototransistor.

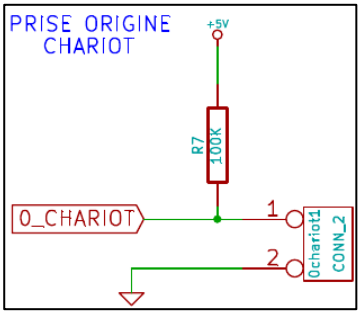
Pour alimenter la diode le même schéma que le laser est réemployé. Cependant la tension de seuil est différente. D'après la datasheet, il faut if ≥20mA.En prenant R = 180Ω, le courant est de $If = \frac{U}{R} = \frac{5}{180} = 27mA$.



Le phototransistor est alimenté sur le collecteur, et l'émetteur est directement relié à une entrée de l'Arduino. L'entrée 2 a été choisie car elle va permettre de gérer l'information du capteur comme une interruption au niveau du programme. Lorsque le signal est réfléchi, l'Arduino reçoit 5V. La résistance R5 permet de limiter le courant du collecteur. Elle permet aussi de garantir un niveau bas à 0V : lorsque le transistor est bloqué, l'entrée de l'Arduino est « en l'air » et pourrait détecter un niveau haut à cause de bruit électromagnétique.

c) Fin de course

Pour l'interrupteur fin de course, nous avons aussi ajouté une résistance de PULL-DOWN. Cependant, comme Eric Normandin nous l'a expliqué, la broche de l'Arduino est configurée en entrée (en open-drain, ou en weak pull-up). C'est donc plus fiable de mettre une résistance de PULL-UP. Au repos, lorsque l'interrupteur est ouvert, la broche est de l'Arduino est au niveau haut (5V). Lorsque le chariot vient enclencher l'interrupteur, l'Arduino détecte un passage à l'état bas (0V)



d) Communication

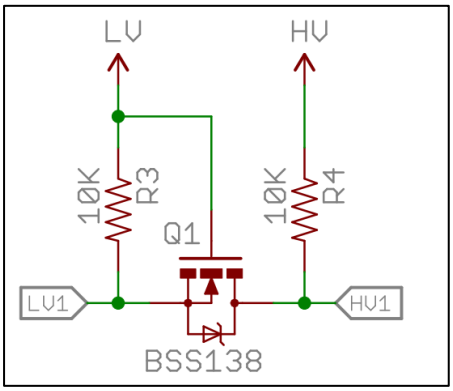
Pour faire communiquer les deux cartes il y a plusieurs choix : l'Arduino et la RaspberryPi supportent les liaisons séries UART, SPI et I2C.

Le bus SPI a été choisi car c'est une liaison full duplex, synchrone avec un fonctionnement maitre esclave. C'est la plus simple à mettre en œuvre du point de vue de la programmation: le maitre (ici RaspberryPi) active l'horloge pour transférer des données de la taille d'un octet. Si l'esclave (Arduino) veux donner une réponse, l'octet est envoyé au maitre dès que l'horloge est détectée par l'esclave.

Le bus I2C et UART ont un fonctionnement et un protocole plus compliqué. Pour transférer des donnés en I2C, la trame est composée d'un bit de start, de l'adresse de l'esclave, du sens de transfert, et d'un acquittement. En UART la trame est composée d'un bit de start, un bit de parité et un bit de stop. Pour la liaison SPI le maitre active l'horloge et les données sont directement transférées.

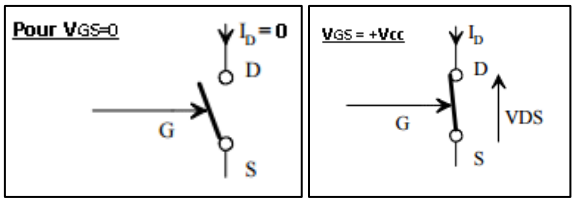
Sur la RaspberryPi l'envoi des données est très simplement programmé à l'aide de la bibliothèque WiringPi. Sur l'Arduino il existe aussi une librairie, par contre cette librairie permet d'utiliser l'Arduino en maitre uniquement. Pour un fonctionnement esclave, il faudra directement écrire ou lire dans le registre de mémoire correspondant.

L'arduino a un niveau logique bas à 0V et haut à +5V. La RaspberryPi a un niveau logique bas à 0V et un niveau logique haut à +3,3V. Si la Raspberrypi reçoit du +5V, elle risque de se détériorer. Il faut dont ajouter à notre shield un convertisseur de niveau logique bidirectionnel qui permet de transformer du +5V au +3,3V et inversement. Voici le montage existant sous licence libre (Open Hardware Creative Commons) que nous allons reproduire (source : www.sparkfun.com)



Fonctionnement :

Pour un transistor MOFSET en commutation, je sais que son schéma équivalent est le suivant :

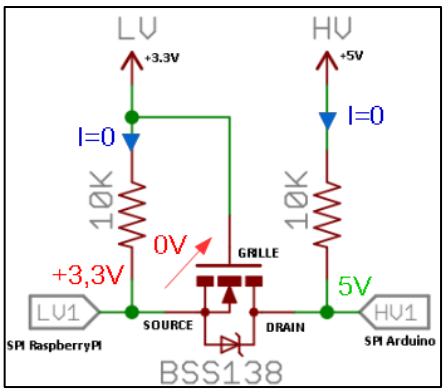


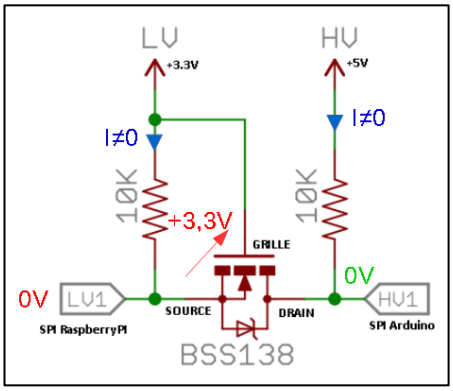
Ici d'après la datasheet la tension « gate threshold voltage » est égal à 1.3V, donc si V_{GS} est supérieur, le transistor est passant, sinon il est bloqué.

- Cas 1, le RaspberryPi envoie +3.3V:

V_{GS} = 0V, le MOSFET est bloqué, Id = 0

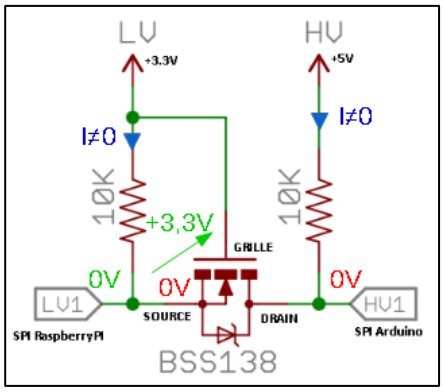
La résistance de PULL-UP met la broche de l'Arduino à +5V, c'est un 0 logique.





- Cas n°2, le RaspberryPi envoie 0V :

V_{GS} = 3.3V, le MOSFET est passant, tout le bus passe au niveau bas, l'Arduino détecte 0V, c'est un 1 logique.



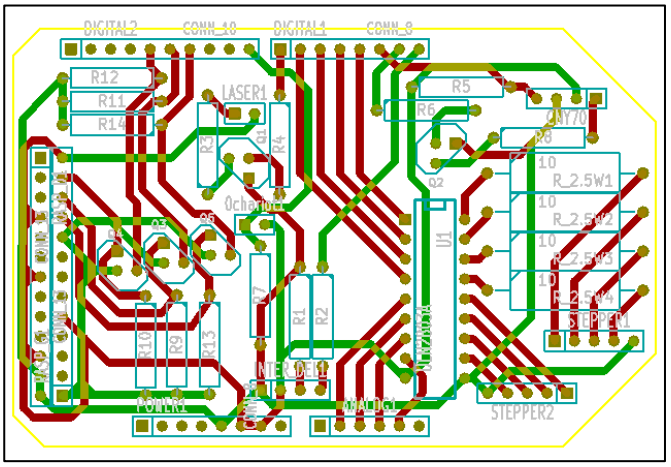
- Cas n°3, Arduino envoie 0V, et la RaspberryPi est au repos (3.3V) :

Lorsque l'Arduino passe à 0V et que le RaspberryPi était à 3.3V, la diode devient passante, la source est donc maintenant à 0V, la tension V_{GS} passe à 3.3V, le MOFSET est passant, la RapberryPi détecte un 1 logique envoyé par l'Arduino.

Annexe n°1 Schéma complet

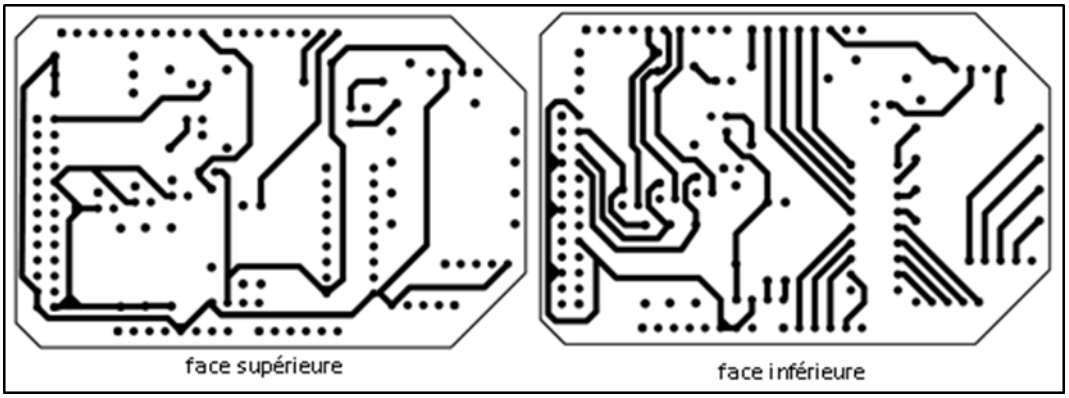
B. Rootage

Après la conception des circuits électriques et quelques tests, l'étape du routage débute. Il faut implanter les composants en fonction de leurs dimensions et de l'encombrement de la carte, puis dessiner les pistes en cuivre. Pour l'étape du routage le logiciel open source KiCad est utilisé. Après avoir implanté tous les composants et dessiné les pistes, la carte sera réalisée à l'UPPA.



Voici le procédé pour graver les pistes sur une carte électronique :

La carte est une plaque composée de 3 couches : Le support en époxy isolant et résistant mécaniquement au centre, une couche de cuivre qui est conductrice, et une pellicule protectrice de résine photosensible. Une carte double face sera utilisée, il y aura des pistes au-dessus et au-dessous, il y a donc deux couches de cuivre et de résine. Le logiciel KiCad permet d'imprimer un typon.

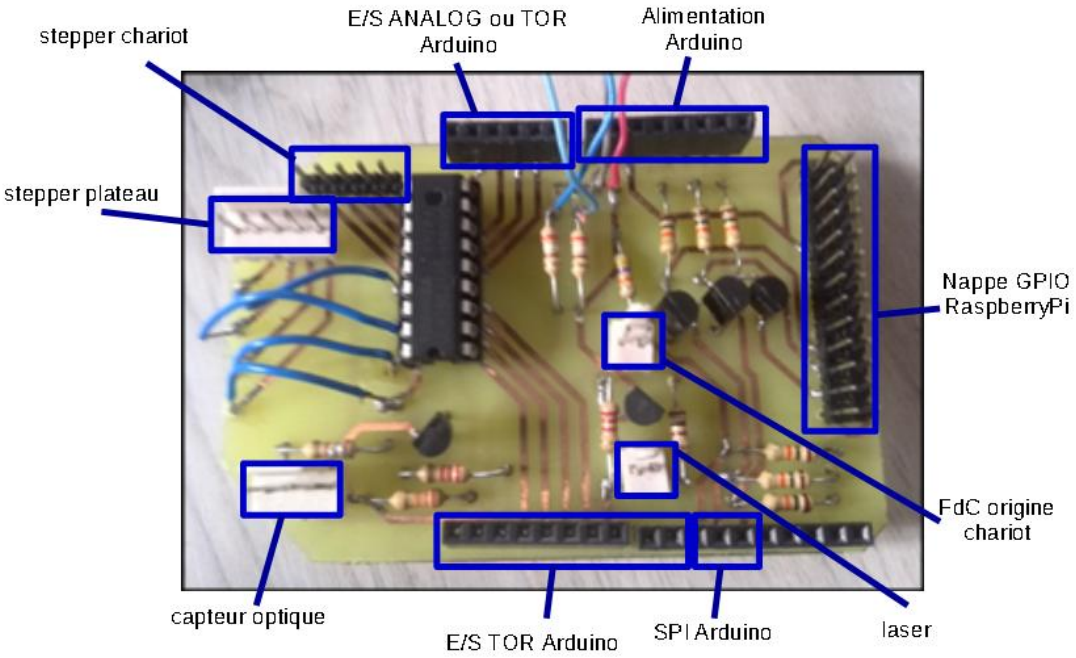


Le typon est le masque transparent sur lequel sont imprimées les pistes, dans une encre opaque aux ultraviolets. Le typon sert lors de l'insolation de la plaque pré-sensibilisée. Le typon est placé sur la résine de la plaque. On éclaire alors cette dernière à la lumière ultraviolette : là où le typon est opaque, la résine est laissée inerte et intacte. Derrière les zones transparentes, la résine est polymérisée par les ultraviolets, ce qui la rend soluble. On retire ensuite le typon et on trempe la plaque dans un révélateur : la résine polymérisée est arrachée de son support de cuivre, laissant ce dernier à nu, tandis que la résine inerte reste accrochée sur le cuivre et le protège. La plaque est gravée par trempage dans une solution corrosive qui dissout le cuivre, sauf aux endroits protégés par la résine encore présente. Il ne reste plus que les pistes de cuivre recouvertes par de la résine. Enfin la plaque est nettoyée avec de l'acétone pour retirer les derniers résidus de résine laissant à nu les pistes de cuivre sur lesquelles on pourra souder.

Quelques erreurs sont à noter :

- Les pastilles, à percer pour placer les broches des composants, sont trop petites. Elles risquent de se détacher lors du perçage, et la soudure sera compliquée.
- Sur certaines parties de la carte, les pistes sont du même côté que les composants. Normalement il faut souder le bout de la broche de l'autre côté du composant. Ici il faudra souder sous le composant ce qui est plus délicat.

Lors des tests nous remarquons qu'une résistance a été oubliée pendant la soudure.



III. Partie informatique

A. Prise en main du RaspberryPi

Au début du projet, nous avons commencé par apprendre à utiliser le port GPIO du RaspberryPi pour allumer le laser et le moteur. Les broches peuvent être directement pilotées depuis l'invité de commande linux grâce à l'instruction « gpio ».
Voici quelques exemples :

- gpio mode <pin> in/out/pwm/clock/up/down/tri

Choix du mode pour une broche (entrée, sortie, mli, horloge, pull up/down…), il faut remplacer <pin> par le numéro de la broche que l'on utilise.

- gpio write <pin> 0/1

Met une broche à l'état haut (+3,3V) ou l'état bas (0V).

- gpio read <pin>

Lit la valeur logique sur une broche.

Ensuite nous avons appris à élaborer des scripts Shell. Le script Shell est un fichier texte qui contient une suite d'instructions pour l'invité de commande Linux. Il n'a pas besoin de compilation. On lance le script dans depuis l'invité de commande grâce à l'instruction suivante :
./nomduscript.sh
Voici quelques exemples d'instructions:
ls : lister les fichiers d'un dossier
cd : changer de dossier
mkdir : créer un dossier
echo : afficher du texte à l'écran
read : demande une saisis pour la stocker dans une variable
Annexe n°2 exemple de script pour faire tourner le moteur pas à pas

Dans un script on peut aussi créer des variables. Cependant, les variables sont toutes des chaînes de caractères.
Le script peut être lancé avec des paramètres :
./nomduscript.sh param1 param2 param3
Pour réutiliser les paramètres, des variables sont automatiquement crées : $1 contient param1, $2 contient param2…
Les scripts ne sont pas fait pour piloter des actionneurs, mais plutôt pour automatiser une suite de d'instruction, manipuler des fichiers, lancer des programmes…

Ainsi, pour utiliser le port gpio la bibliothèque wiring pi a été installé, les programmes pour les actionneurs seront donc rédigés en langage C. Ces programme sont compilés, un fichier exécutable est créé, il peuvent être lancer depuis l'invité de commande grâce à l'instruction suivante : ./nomduprogramme.
Néanmoins, des scripts vont permettre de lancer les programmes d'actionneur de manière séquentielle.
Exemple de script :

```
#!/bin/bash
./allumer                    : lance le programme en c pour allumer le laser
raspistill -o photo.jpg      : instruction pour prendre une photo et l'enregistre
./eteindre                   : lance le programme en c pour éteindre le laser
```

Programme pour allumer le laser en langage C avec WiringPi :

```
#include <wiringPi.h>        : ajout de la bibliothèque
int main (void)              : création de la boucle principale
{
    wiringPiSetup();         : instruction propre à la bibliothèque pour configurer le numéro des broches
    pinMode (0, OUTPUT) ;    : broche n°0 configurée en sortie
    digitalWrite (0,HIGH);   : broche n°0 mise à l'état haut
    return 0;                : fin
}
```

Finalement, comme expliqué dans la présentation, nous choisissons d'utiliser un Arduino pour commander les actionneurs. Cependant cette structure, avec des scripts et des programmes en C, sera toujours utilisée pour envoyer des ordres à l'Arduino et réaliser les opérations de traitement d'image.

B. Traitement d'image

Comme expliqué dans la présentation du projet, pour obtenir la coordonnée d'un point sur la surface du sujet, il faut prendre une photo et mesurer l'écart du trait du laser par rapport au centre de la photo. Tout d'abord la photo va être capturée et enregistrée sur le RaspberryPi grâce au logiciel en ligne de commande « Raspistill ». Ensuite plusieurs programmes que j'ai élaboré en utilisant la bibliothèque de traitement d'image OpenCV vont permettre de :

- Seuiller l'image: tous les pixels rouges qui correspondent au trait du laser serons blanc, tous les autre pixels parasites seront noirs. Nous obtenons une image binaire.
- Sur l'image en noir et blanc nous pourrons calculer la distance recherchée en pixel.

Pour améliorer le procédé, dans un second temps, j'ai aussi élaboré des programmes qui permettent de recadrer l'image, modifier le contraste et la luminosité, et filtrer l'image grâce à des opérateurs morphologiques.

L'instruction qui permet de prendre une photo est la suivante :

raspistill -o nom_de_la_photo.jpg

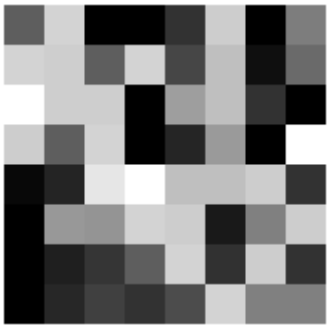
Plusieurs paramètres vont permettre d'optimiser la photo :

-w : largeur de l'image, -h: hauteur de l'image, -tl : mode time lapse (une photo toutes les <t> ms),

-co : contraste, -br : luminosité, -sa : saturation, -rot: rotation

En informatique, une image est en fait une matrice de pixel :

Image quelconque :



Matrice équivalente :

$$\begin{pmatrix} 94 & 211 & 0 & 0 & 50 & 205 & 0 & 125 \\ 211 & 206 & 94 & 211 & 70 & 191 & 15 & 106 \\ 255 & 206 & 206 & 0 & 158 & 191 & 50 & 0 \\ 205 & 94 & 211 & 0 & 37 & 154 & 0 & 255 \\ 8 & 36 & 230 & 255 & 191 & 191 & 205 & 50 \\ 0 & 152 & 148 & 211 & 205 & 25 & 128 & 205 \\ 0 & 31 & 53 & 94 & 211 & 49 & 205 & 50 \\ 0 & 39 & 64 & 50 & 76 & 211 & 128 & 128 \end{pmatrix}$$

Les images sont échantillonnées en un nombre fini de pixels. Une matrice à n lignes et p colonnes contient des scalaires. Dans la matrice le scalaire I(x,y), de valeur comprise entre 0 et 255, correspond au niveau de gris du pixel d'abscisse x et d'ordonnée y. 0 correspond au noir, 255 correspond au blanc. En programmation, ces matrices de scalaires seront en fait un tableau de variables de type UCHAR.

Pour les images en couleurs les matrices ont trois dimensions. Chaque pixel coloré est le mélange d'un pixel rouge avec un pixel vert et un pixel bleu qui ont des intensités lumineuses différentes. C'est le codage RGB (red, green, blue).

Couleur	Représentation RGB
Rouge	(255,0,0)
Vert	(0,255,0)
Bleu	(0,0,255)
Noir	(0,0,0)
Blanc	(255,255,255)
Jaune	(255,255,0)
Gris 50%	(127,127,127)

Les couleurs peuvent aussi être codé d'une autre façon : HSV (hue, saturation, value)

H correspond à la teinte (0° ou 360° : rouge, 60° : jaune, 120° : vert, 180° : cyan, 240° : bleu, 300° : magenta.)

S correspond à l'intensité de la couleur (de 0 à 100%)

V correspond à la brillance de la couleur (de 0 à 100%)

La bibliothèque OpenCV est très complète, nous allons utiliser des fonctions de bases uniquement. Voici les modules qui seront utiles:

core : fonctionnalités de base. Ce module permet de manipuler les structures de base, réaliser des opérations sur des matrices, dessiner sur des images, sauvegarder et charger des données dans des fichiers

imgproc : traitement d'image. Les fonctions et structures de ce module ont trait aux transformations d'images, au filtrage, à la détection de contours, de points d'intérêt…

highgui : entrées-sorties et interface utilisateur. Module pour ouvrir, enregistrer et afficher des images et des flux vidéo.

Pour le seuillage, voici la construction du programme :

- ouverture de l'image :

  - une matrice vide est créée.

  - le fichier image RGB (.jpeg), passé comme paramètre, est chargé dans la matrice

- seuillage :

  - création d'une matrice en HSV la fonction binarisation est plus efficace.

  - la matrice en RGB est convertit en HSV dans la nouvelle matrice.

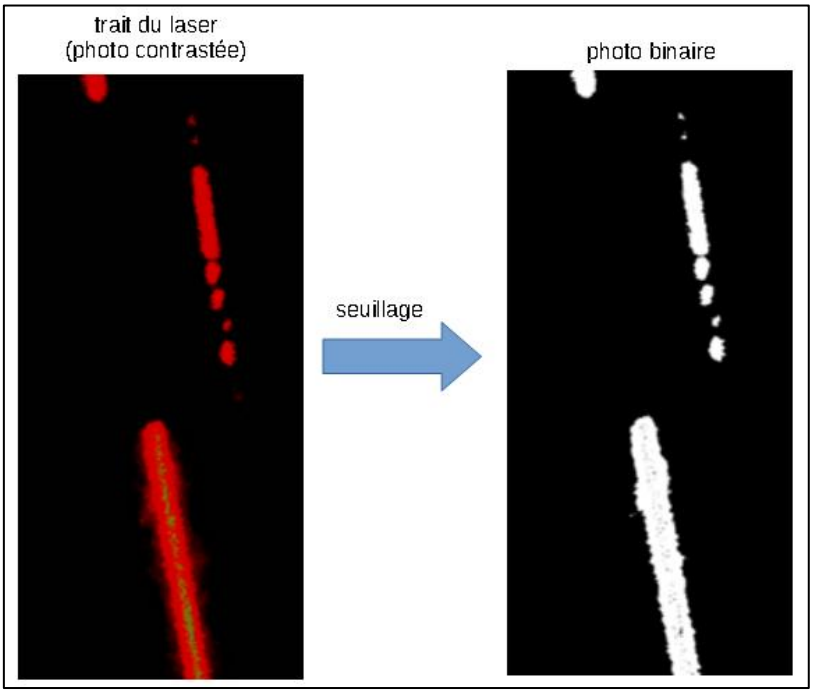
  - création d'une troisième matrice pour la binarisation : tous les pixels inférieur à un seuil seront noir, les pixels supérieurs au seuils seront blanc.

- sauvegardes

  - la matrice binarisée est sauvegardée comme une nouvelle image au format jpeg.

Plusieurs essais ont été réalisés avant d'arriver à ce procédé. Pour obtenir de bon résultats la photo initiale doit être la plus contrastée possible : tous le fond doit être le plus sombre possible et uniquement le trait du laser doit ressortir. Une telle image est obtenue en jouant sur les paramètres du logiciel qui prend la photo (Raspistill).

Annexe n°3 programme de seuillage.



Le seuillage, avec les bons paramètres, permet d'obtenir une image « propre ». On aurait pu charger la photo de base en niveau de gris, ou charger uniquement le canal rouge. Ces solutions ont étés testées cependant il y a toujours des pixels parasites qui sont de la même couleur que le laser. Le calcul de la position du trait est erroné.

Voici l'architecture du programme pour calculer la position de chaque pixel :

- chargement de l'image dans une matrice à une dimension (niveau de gris)
- on mémorise dans des variables le nombre total de lignes et de colonnes
- boucle "for" pour parcourir de toutes les lignes
    - boucle "for" pour avancer de colonne en colonne
        - test de la couleur du pixel sur lequel on se trouve
        - si il est blanc on sauvegarde la position dans un tableau et on passe à la ligne suivante
        - si il est noir on passe au pixel de la colonne suivante

Ce programme a été optimisé :

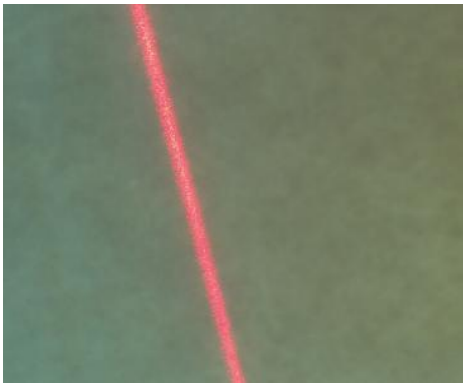
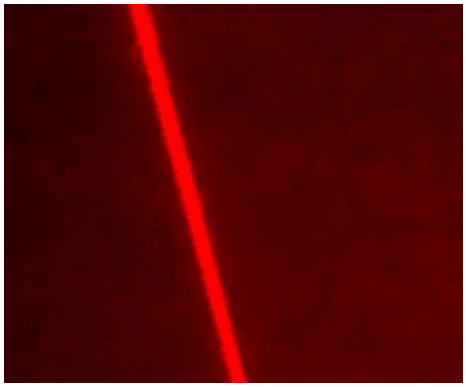
- le trait du laser à une certaine épaisseur, le premier pixel blanc depuis la gauche puis le premier pixel blanc depuis la droite seront recherché pour calculer la moyenne sur chaque ligne.
- les lignes entièrement noir sont détectés, l'objet n'a pas été éclairé par le laser, il y a une discontinuité.

Annexe n°4 programme calcul de la position

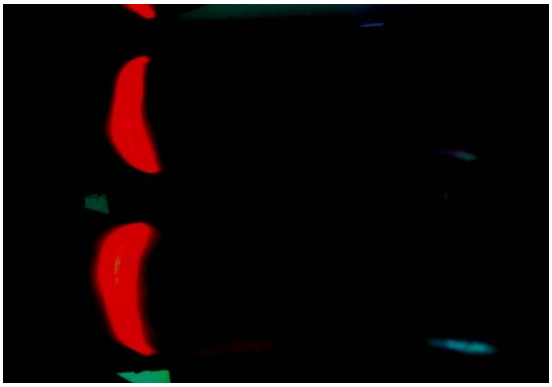
Ensuite d'autres programmes qui permettent d'optimiser la démarche ont été élaborés en s'inspirant d'exemples de la documentation de Open CV.

- lissage de l'image :


- modification du contraste et de la luminosité :

- rognage :

Annexe n°5 programme de lissage

Annexe n°6 programme de modification du contraste et de la luminosité

Annexe n°7 programme de rognage

C. Arduino

Au niveau de l'Arduino, j'ai créé une bibliothèque pour simplifier la programmation des moteurs pas à pas. Les bibliothèques Arduino sont composés de deux fichiers, un fichier d'en tête (extension .h) et un fichier source (extention .cpp). Le fichier d'en tête contient les définitions des fonctions disponibles et le fichier source contient l'implémentation du code. C'est à dire le code à l'intérieur des fonctions qui ont été définies dans le fichier d'en tête. Dans le dossier du logiciel Arduino se trouve un sous dossier nommé "libraries". Chaque dossier à l'intérieur de libraries représente une bibliothèque. Pour ajouter la nôtre à celles officielles, on crée un dossier au nom de notre bibliothèque. Lorsque la bibliothèque est créée, on pourra appeler les fonctions de cette bibliothèque, depuis le « main » grâce à l'instruction #include <ma bibliothèque>

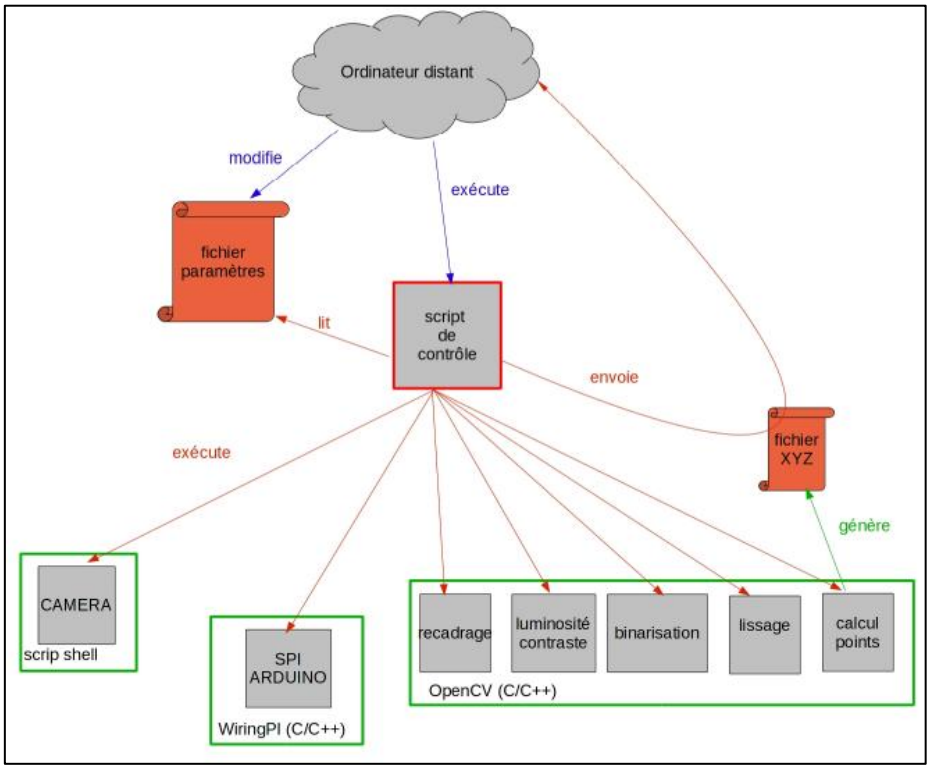
Annexe n°8 bibliothèque moteur pas à pas

D'autre part, j'ai élaboré un programme pour contrôler l'avance du chariot en fonction de l'information du capteur. Pour rendre le système réactif, le capteur a été géré en temps réel, comme interruption.

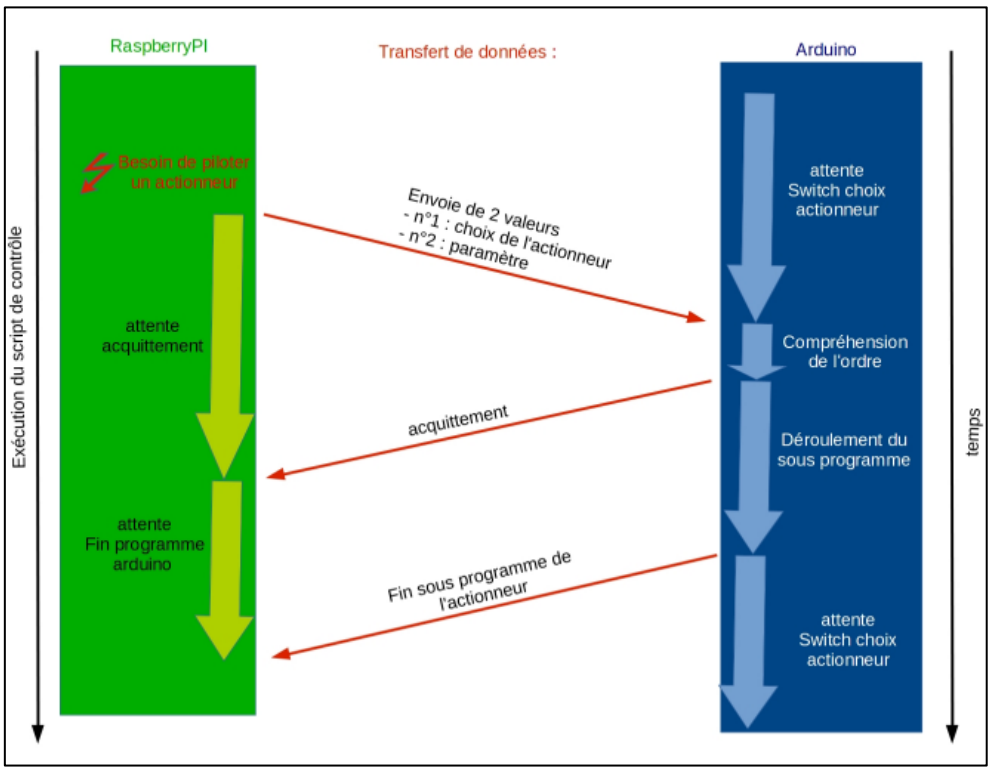
Annexe n°9 Programme capteur optique

D. Structure informatique du système et communication SPI

Suite à la conception des programmes séparément, voici la structure du système qui a été imaginée (vue depuis le maitre : RaspberryPi) :



Pour commander les actionneurs depuis la RaspberryPi voici le protocole qui a été imaginé :



Nous pouvons imaginer que si la RaspberryPi ne reçoit pas d'acquittement, l'ordre en renvoyé, au bout de X fois, un problème est signalé à l'utilisateur. Si la fin du sous-programme Arduino n'est pas reçue avant un temps T, un problème est signalé à l'utilisateur également.

Ce protocole n'a pas été entièrement mis au point par manque de temps. Voici ce qui a été testé :

- un programme qui envoie un ordre et un paramètre (deux octets) depuis la RaspberryPi

- un programme de scrutation dans l'Arduino.

- l'Arduino répond d'un acquittement lorsqu'il reçoit l'ordre

- dans l'Arduino, une boucle « swich » permet de piloter un actionneur ou un autre en prenant un paramètre en compte (exemple : tourner le stepper dans certain nombre de pas)

Annexe n°10 programme SPI RaspberryPi

Annexe n°11 SPI Arduino

VII. Conclusion :

Je suis satisfait par le travail réalisé pendant ce projet. Bien que le projet de prototype de scanner 3D ne soit pas terminé, j'ai appris à être autonome et autodidacte. Le travail réalisé va au-delà des connaissances acquises en cours. Nous avons perdu du temps à cause de problème d'organisation et d'accès aux différents lieux de travail (FabLab, Lycée et Fac). Néanmoins mener ce projet était très enrichissant et motivant. Ce projet était très complet, il regroupait tous les domaines de la mécatronique.

Le projet est terminé pour nous au niveau de nos études, cependant il sera ouvert à tout le monde au FabLab pour le terminer et apporter des améliorations. Nous pouvons par exemple imaginer un contrôle à distance depuis une interface dans un navigateur web. Nous pouvons encore imaginer une forme de e-learning, à partir d'un certain nombre de points capturés, le scanner pourrait reconnaître les formes géométriques en fonction d'une bibliothèque...

VIII. Référence bibliographiques

- Mécanique :

  Guide du Dessinateur Industriel, Hachette Supérieur

  http://www.hpceurope.com/fr/

  http://www.norelem.fr/

- Electronique / Programmation :

  http://www.arduino.cc/

  https://www.raspberrypi.org/documentation/

  http://www.raspbian.org/

  http://mchobby.be/wiki

  http://elinux.org

  Linux Magazine France, hors-série n°75 RaspberryPi niveau avancé

  http://mchobby.be/wikis-série n°75 RaspberryPi

  http://wiringpi.com/

  http://cppreference.com

  http://openclassrooms.com/

  http://www.mon-club-elec.fr/

  http://www.sparkfun.com

http://fr.rs-online.com/

- Traitement d'image :

OReilly Learning OpenCV (Gary Bradski and Adrian Kaehler)

http://www.laganiere.name/

http://docs.opencv.org/index.html

http://remisoft.info/wiki/doku.php?id=opencv:opencv

http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/opencv-intro.html

http://www.shervinemami.info/imageTransforms.html

http://www.geckogeek.fr/tutorial-opencv-isoler-et-traquer-une-couleur.html

http://fr.wikipedia.org/wiki/Traitement_d%27images

http://en.wikipedia.org/wiki/Range_imaging

- Exemples de scanner 3D :

http://telefab.fr/2013/06/14/scanner-3d-kinect/

https://projets-labinfo.he-arc.ch/projects/magritte/wiki/Kinect

http://diablotronic.free.fr/Pyscan3d/pyscan3d.html

http://www.123dapp.com/catch

http://reprap.org/wiki/Scanning

http://www.simple3d.com/

http://www.makerscanner.com/

http://wiki.labomedia.org/index.php/Laser_Scanner_3D_SkanDal

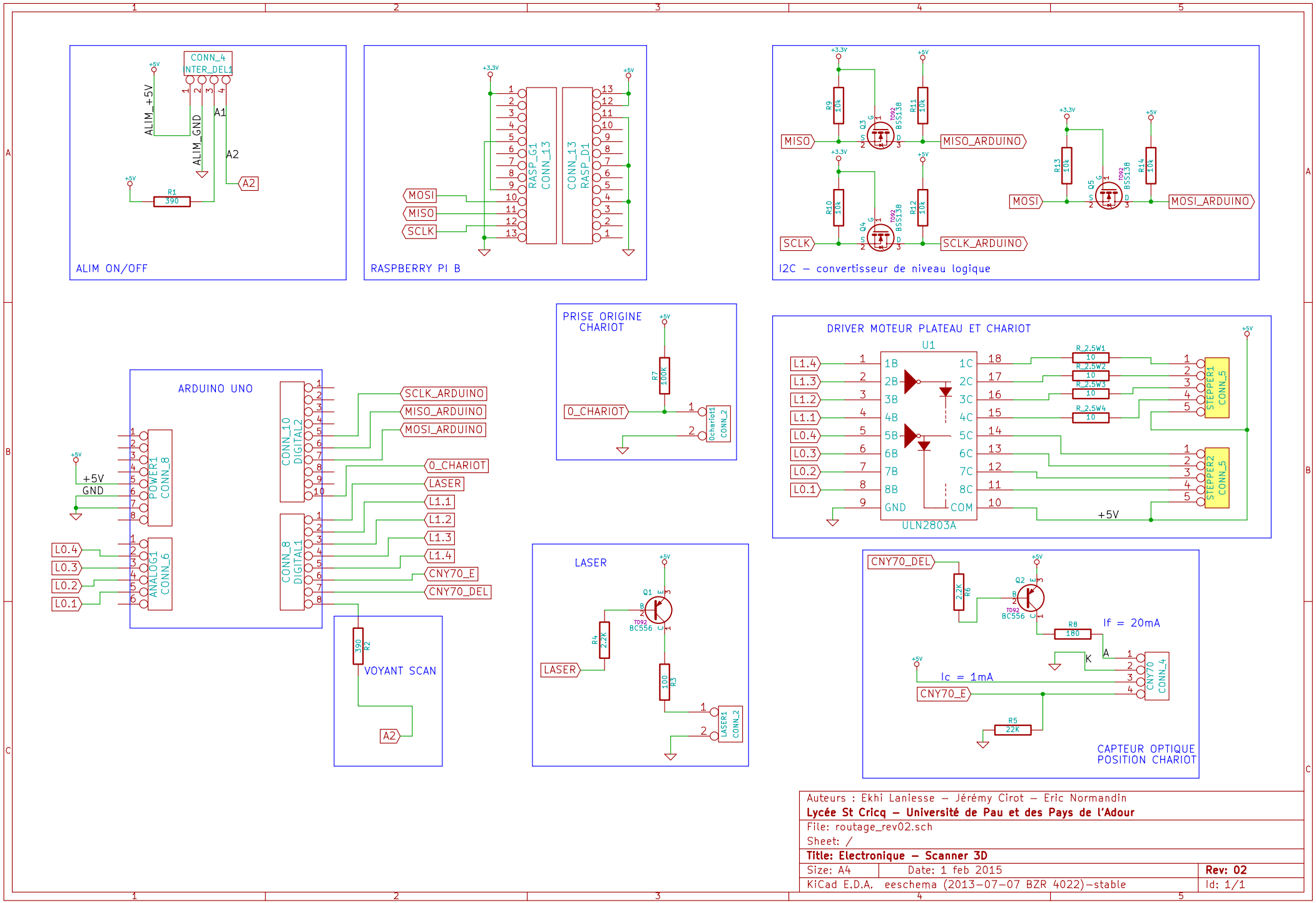
http://www.ensci.com/blog/fablab/category/scanner3d/

http://hci.rwth-aachen.de/FabScan

http://fr.wikipedia.org/wiki/Scanner_tridimensionnel

IX. Annexes

Annexe 1: schéma du schield



Auteurs : Ekhi Laniesse – Jérémy Cirot – Eric Normandin		
Lycée St Cricq – Université de Pau et des Pays de l'Adour		
File: routage_rev02.sch		
Sheet: /		
Title: Electronique – Scanner 3D		
Size: A4	Date: 1 feb 2015	**Rev: 02**
KiCad E.D.A. eeschema (2013-07-07 BZR 4022)-stable		Id: 1/1

Annexe n°2 exemple de script pour faire tourner le moteur pas à pas

```
#!/bin/bash                    //invité de commande qui va executer le script

//paramétrage des broches en sortie
gpio mode 1 out
gpio mode 0 out
gpio mode 2 out
gpio mode 3 out

//cycle des 4 pas pour faire un tour entier
while true; do
   gpio write 1 1
   gpio write 0 1
   gpio write 2 0
   gpio write 3 0
   gpio write 1 0
   gpio write 0 1
   gpio write 2 1
   gpio write 3 0
   gpio write 1 0
   gpio write 0 0
   gpio write 2 1
   gpio write 3 1
   gpio write 1 1
   gpio write 0 0
   gpio write 2 0
   gpio write 3 1
done
```

Annexe n°3: programme de seuillage

```
//inclusion des bibliothèques

#include <stdio.h>
#include <stdlib.h>
#include <opencv/highgui.h>
#include <opencv/cv.h>

//valeur de la constante CV_BGR2HSV pour transformer une image RGB
#define CV_BGR2HSV 40

//debut main
int main (int argc, char* argv[])
{
  IplImage *imagergb = NULL;
//variable tolerance
    double tolerance=0;
    printf("entrer la tolerance");
    scanf("%lf", &tolerance);

  /* Vérification: au moins 3 arguments doivent être passés au programme.*/
  if (argc < 4)
  {
    fprintf (stderr, "usage: %s entrer une IMAGE en paramètre \n", argv[0]);
    return EXIT_FAILURE;
  }


  /* Chargement de l'image passée en argument */
  imagergb = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);

  if (!imagergb)
  {
    fprintf (stderr, "couldn't open image file: %s\n", argv[1]);
    return EXIT_FAILURE;
  }

//convertir l'image en hsv
  IplImage *imagehsv;
  imagehsv = cvCloneImage(imagergb);
  cvCvtColor(imagergb, imagehsv, CV_BGR2HSV);

//créer un masque de la même taille que l'image
  IplImage *imagebin;
  imagebin = cvCreateImage(cvGetSize(imagehsv), imagehsv->depth, 1);

//sauvegarder
  cvSaveImage(argv[2], imagehsv, 0);
//mise à l'échelle de la tolerance
    tolerance = 255*tolerance/100;

//binariser l'image hsv
  cvInRangeS(imagehsv,cvScalar(0, 255-tolerance, 230-tolerance,0),cvScalar(0+tolerance, 255,
  230,0),imagebin);

//sauvegarder l'image binarisée
  cvSaveImage(argv[3], imagebin, 0);


  /* Libération de la mémoire */
```

```
  cvReleaseImage (&imagergb);
  cvReleaseImage(&imagehsv);
  cvReleaseImage(&imagebin);

  return EXIT_SUCCESS;
}
```

Annexe n°4 programme calcul de la position

```
#include <stdio.h>
#include <stdlib.h>
#include <opencv/highgui.h>

int main (int argc, char* argv[])
{
  IplImage* img = NULL;

  /* Vérification: au moins un argument doit être passé au programme.*/
  if (argc < 2)
  {
    fprintf (stderr, "usage: %s IMAGE\n", argv[0]);
    return EXIT_FAILURE;
  }

  /* Chargement de l'image passée en argument */
  img = cvLoadImage(argv[1], CV_LOAD_IMAGE_GRAYSCALE);

  if (!img)
  {
    fprintf (stderr, "couldn't open image file: %s\n", argv[1]);
    return EXIT_FAILURE;
  }

  int y=0;
  int x=0;
  int xMax=img-> width; //largeur de l'image
  int yMax=img-> height; //hauteur de l'image
  uchar varpixel=0; //variable = couleur pixel
  uchar *pixel=&varpixel; //pointeur sur varpixel
  int tab[yMax][3]; //tableau xgauche xdroite xmoyen

//detection toutes les lignes suivantes

for (y=0; y<yMax; y++)//on parcourt toutes les lignes
{
  for (x=0; x<xMax; x++) //on avance colonne par colonne depuis la gauche
    {
        pixel = cvPtr2D (img, y, x, NULL); //on mesure la couleur de chaque pixel
        if (*pixel > 100) //1er pixel blanc trouvé
        {
            tab[y][0]=x; //on sauvegarde le pixel
            x=xMax; //on sort de la boucle for
        }
        else if (*pixel < 100 && x==xMax-1)//si uniquement pixel noir jusqu'au bout
        {
            tab[y][0]=0;//on met 0 pour cette ligne
            x=xMax;//on sort de la boucle for
        }
    }//trouve le premier pixel blanc de chaque ligne depuis la gauche

    for (x=xMax-1; x>0; x--)
    {
        pixel = cvPtr2D (img, y, x, NULL);
        if (*pixel > 100)
        {
            tab[y][1]=x;
            x=0;
        }
```

```
          else if (*pixel < 100 && x==1)
        {
            tab[y][1]=0;
            x=0;
        }
    }//trouve le premier pixel blanc de chaque ligne depuis la droite

//calcul de tous les xmoyen
    tab[y][2] = tab[y][0] + tab[y][1];
    tab[y][2] = tab[y][2] / 2;
//affichage des données
    printf( "y: %d  ",y);
    printf( "xGauche: %d  ",tab[y][0]);
    printf( "xDroite: %d  ",tab[y][1]);
    printf("moyenne: %d\n",tab[y][2]);
}

  /* Libération de la mémoire */
  cvReleaseImage (&img);

  return EXIT_SUCCESS;
}
```

Annexe n°4 programme calcul de la position

```
#include <stdio.h>
#include <stdlib.h>
#include <opencv/highgui.h>
#include <opencv/cv.h>


int main (int argc, char* argv[])
{
  IplImage* image = NULL;

  /* Vérification: au moins 2 arguments doivent être passés au programme.*/
  if (argc < 3)
  {
    fprintf (stderr, "usage: %s entrer une IMAGE en paramètre \n", argv[0]);
    return EXIT_FAILURE;
  }

  /* Chargement de l'image passée en argument 1*/
  image = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);

  if (!image)
  {
    fprintf (stderr, "couldn't open image file: %s\n", argv[1]);
    return EXIT_FAILURE;
  }

IplConvKernel *kernel; //creation d'un kernel (zone d'interet)
kernel = cvCreateStructuringElementEx(20, 20, 2, 2, CV_SHAPE_ELLIPSE, NULL);
//dimensionnement du kernel
cvDilate(image, image, kernel, 1); //dilatation
kernel = cvCreateStructuringElementEx(25, 25, 2, 2, CV_SHAPE_ELLIPSE, NULL);
//redimensionnement du kernel
cvErode(image, image, kernel, 1); //erosion
//les amas de pixels blancs sont agrandis puis les arretes vive sont lissées

//sauvegarder
  cvSaveImage(argv[2], image, 0);

  /* Libération de la mémoire */
  cvReleaseImage (&image);

  return EXIT_SUCCESS;
}
```

Annexe n°6 programme de modification du contraste et de la luminosité

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;

double alpha; /**< Simple contrast control */
int beta;  /**< Simple brightness control */

int main( int argc, char** argv )
{
/// Read image given by user
Mat image = imread( argv[1] );
Mat new_image = Mat::zeros( image.size(), image.type() );

/// Initialize values
std::cout<<" Basic Linear Transforms "<<std::endl;
std::cout<<"-------------------------"<<std::endl;
std::cout<<"* Enter the alpha value [1.0-3.0]: ";std::cin>>alpha; //1.5
std::cout<<"* Enter the beta value [0-100]: "; std::cin>>beta;  //-100


/// Do the operation new_image(i,j) = alpha*image(i,j) + beta
for( int y = 0; y < image.rows; y++ )//avance de ligne en ligne
{
    for( int x = 0; x < image.cols; x++ )//avance de colone en colone
    {
            new_image.at<Vec3b>(y,x)[0] = 0;//le canal bleu est mis à 0
            new_image.at<Vec3b>(y,x)[1] = 0;//le canal vert est mis à 0
            new_image.at<Vec3b>(y,x)[2] = saturate_cast<uchar>( alpha*( image.at<Vec3b>(y,x)[
            2] )+beta);//le canal rouge est saturés
    }
}
//les pixel rouges seront encores plus intenses, les autre teinte sont supprimées

imwrite(argv[2],new_image); //sauvegarde

return 0;
}
```

Annexe n°7 programme de rognage

```
#include <stdio.h>
#include <stdlib.h>
#include <opencv/highgui.h>
#include <opencv/cv.h>



int main (int argc, char* argv[])
{
    IplImage* bigimage = NULL;

    /* Vérification: au moins 2 arguments doivent être passés au programme.*/
    if (argc < 3)
    {
    fprintf (stderr, "usage: %s entrer une IMAGE en paramètre \n", argv[0]);
    return EXIT_FAILURE;
    }


    /* Chargement de l'image passée en argument */
    bigimage = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED);

    if (!bigimage)
    {
    fprintf (stderr, "couldn't open image file: %s\n", argv[1]);
    return EXIT_FAILURE;
    }

    int bigXmax=bigimage-> width; //longeur de l'image à redim
    int bigYmax=bigimage-> height; //largeur de l'image à redim
    int xdroite=100, xgauche=400 ,yhaut=100 ,ybas=400; //redim de chaques cotés

    //dimensions du Region Off Interest plus petit que l'image de base
    int smallXmin=xgauche, smallXlong=bigXmax-(smallXmin+xdroite), smallYmin=yhaut,
    smallYlong=bigYmax-(yhaut+ybas);

    //création du ROI
    cvSetImageROI(bigimage,cvRect( smallXmin, smallYmin, smallXlong, smallYlong));

    //création nouvelle image a partir du ROI
    IplImage *smallimage = cvCreateImage(cvGetSize(bigimage), 8, 3);

    //copie de la grande(uniquement le ROI) dans la petite
    cvCopy(bigimage, smallimage, NULL);

    //sauvegarde
    cvSaveImage(argv[2], smallimage, 0);


    /* Libération de la mémoire */
    cvResetImageROI(bigimage);
    cvReleaseImage (&bigimage);
    cvReleaseImage(&smallimage);


    return EXIT_SUCCESS;
}
```

Annexe n°8 bibliothèque moteur pas à pas - header

```
#ifndef Stepmot_h
#define Stepmot_h

#include "Arduino.h"

class Stepmot
{
  public:
   Stepmot(int pin1, int pin2, int pin3, int pin4, int time);
   void Plateau(int cdeP);
   void Chariot(int cdeC, int sensC);
  private:
   int _pin1;
   int _pin2;
   int _pin3;
   int _pin4;
   int _time;
   int _cdeC;
   int _cdeP;
   int _sensC;
   void Step1();
   void Step2();
   void Step3();
   void Step4();
   void Stop();
};

#endif
```

```
#include "Arduino.h"
#include "Stepmot.h"

Stepmot::Stepmot(int pin1, int pin2, int pin3, int pin4, int time)
{
  pinMode(pin1, OUTPUT);
  pinMode(pin2, OUTPUT);
  pinMode(pin3, OUTPUT);
  pinMode(pin4, OUTPUT);
  _pin1=pin1;
  _pin2=pin2;
  _pin3=pin3;
  _pin4=pin4;
  _time=time;
}

void Stepmot::Step1()
{
  digitalWrite (_pin1, HIGH);
  digitalWrite (_pin2, HIGH);
  digitalWrite (_pin3, LOW);
  digitalWrite (_pin4, LOW);
}

void Stepmot::Step2()
{
  digitalWrite (_pin1, LOW);
  digitalWrite (_pin2, HIGH);
  digitalWrite (_pin3, HIGH);
  digitalWrite (_pin4, LOW);
}

void Stepmot::Step3()
{
  digitalWrite (_pin1, LOW);
  digitalWrite (_pin2, LOW);
  digitalWrite (_pin3, HIGH);
  digitalWrite (_pin4, HIGH);
}

void Stepmot::Step4()
{
  digitalWrite (_pin1, HIGH);
  digitalWrite (_pin2, LOW);
  digitalWrite (_pin3, LOW);
  digitalWrite (_pin4, HIGH);
}

void Stepmot::Stop()
{
  digitalWrite (_pin1, LOW);
  digitalWrite (_pin2, LOW);
  digitalWrite (_pin3, LOW);
  digitalWrite (_pin4, LOW);
}

void Stepmot::Plateau(int cdeP)
{
  _cdeP=cdeP;
```

```
int x=0;
int numstep=1;
if(_cdeP>0)
{
  for(x=0; x<=_cdeP; x++)
  {
    switch (numstep)
    {
      case 1:
        Step1();
        delay (_time);
        numstep++;
       break;
      case 2:
        Step2();
        delay (_time);
        numstep++;
       break;
      case 3:
        Step3();
        delay (_time);
        numstep++;
       break;
      case 4:
        Step4();
        delay (_time);
        numstep=1;
       break;
    }
  }
}
else if(_cdeP<0)
{
  for(x=0; x>=_cdeP; x--)
  {
    switch (numstep)
    {
      case 1:
        Step4();
        delay (_time);
        numstep++;
       break;
      case 2:
        Step3();
        delay (_time);
        numstep++;
       break;
      case 3:
        Step2();
        delay (_time);
        numstep++;
       break;
      case 4:
        Step1();
        delay (_time);
        numstep=1;
       break;
    }
  }
```

```
  }
  else if (_cdeP==0)
  {
    Stop();
  }
  Stop();
}

void Stepmot::Chariot(int cdeC, int sensC)
{
    _cdeC=cdeC;
    _sensC=sensC;
    if (_cdeC!=0)
    {
      if (_sensC==1)
      {

      }
      else if (_sensC==0)
      {

      }
    }
}
```

Annexe n°8 bibliothèque moteur pas à pas - main

```
#include "Stepmot.h" //inclusiuon de la bibliothèque

Stepmot moteur(A1, A2, A3, A4, 100); //definition de la fonction

void setup()
{
}

void loop()
{
 moteur.Plateau(500); //rotation de 500 pas
}
```

```
volatile int interrupt = 0;
int pozition=0;

void fctinterrupt()
{
  interrupt=1;
}

void setup()
{
  attachInterrupt(1, fctinterrupt, RISING);
  Serial.begin(9600);
  pinMode (4,OUTPUT);
  digitalWrite(4,HIGH);
}

void loop()
{

  if (interrupt==1)
  {
    pozition++;
    Serial.print("INTERRUPTION, pozition=");
    Serial.print(pozition);
    Serial.print("\n\r");
    interrupt=0;
  }
  else
  {
     Serial.print("pas d'interruption\n\r");
     delay (1000);
  }

}
```

```
#include <iostream>
#include <wiringPiSPI.h>

unsigned char * message;
unsigned char number1[2]={0,0};

int main(int argc, char* argv[] )
{
  wiringPiSPISetup(0, 5000); // channel 0, speed 5000
  std::cout << "entrer la lettre 1\n";
  std::cin >> number1[0];
  std::cout << "entreer la lettre 2\n";
  std::cin >> number1[1];
  int answer = wiringPiSPIDataRW(0, number1, 2);
  std::cout << number1  [0] << " "<< number1 [1] << "\n";
  //wiringPiSPIDataRW (int channel, unsigned char *data, int len) ;
  //int wiringPiSPIGetFd  (int channel) ;
}
```

```
#include <SPI.h>

unsigned char donnee[2]={0,0};
int x=0;

void setup()
{
   pinMode(MISO, OUTPUT);
   SPCR |= _BV(SPE);
   Serial.begin(9600);
}


void loop()
{
    while ((SPSR & _BV(SPIF)))
    {
        donnee[x]=SPDR;
        x++;
        x>2? x=0 : x=x;
    }
    Serial.print(donnee[0]);
    Serial.print("\n\r");
    Serial.print(donnee[1]);
    Serial.print("\n\r");
    delay(500);
}
```