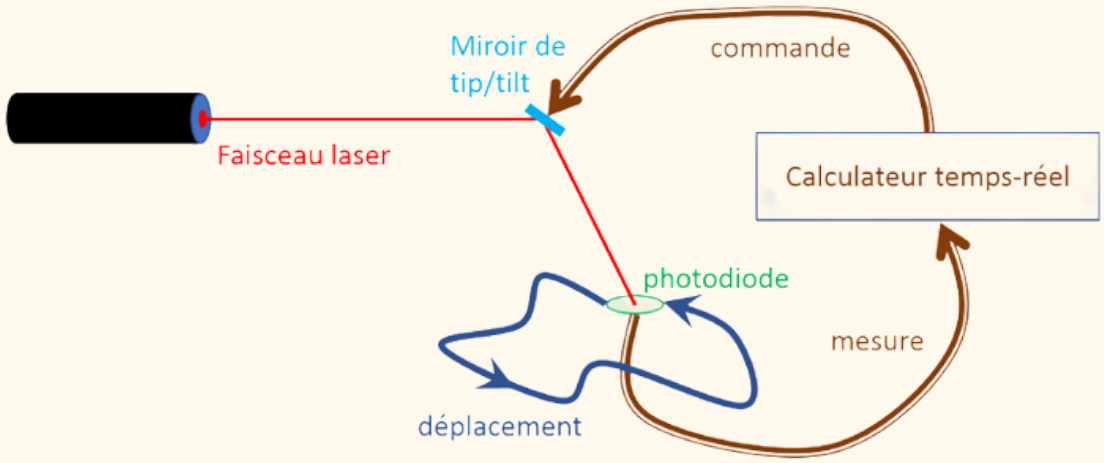
PROTIS

BANC OPTIQUE POUR L'ASSERVISSEMENT D'UN LASER

Par Alice Galbrun et Mathilde Faure



INTRODUCTION

Dans de nombreuses applications il est nécessaire d'avoir un laser dont le faisceau doit être pointé sur une cible avec une grande précision. SOLEC veut proposer une solution d'asservissement de la position du laser à une dimension. On utilise deux servomoteurs dont un est asservi pour pointer le faisceau sur une barrette CCD située sur le deuxième servomoteur. On utilisera pour l'asservissement un correcteur proportionnel intégral.

Table des matières

INTRODUCTION..... 1
Table des matières.....2
Schéma fonctionnel.....4
Détail des missions.....5
Contraintes.....5
Cahier des charges.....6

LES COMPOSANTS ET LEUR UTILISATION..... 7
Servomoteur..... 7
Fonctionnement..... 7
Présentation..... 7
Réalité..... 8
Utilisation..... 8
Branchement..... 8
MBED..... 8
Générateur de fréquence..... 9
Barrette CCD..... 10
Fonctionnement..... 10
Présentation..... 10
Utilisation..... 11
Signaux de la barrette CCD..... 11
MBED..... 12
Etape de vérification..... 13
Asservissement..... 15
Présentation..... 15
Correcteur proportionnel intégrateur dérivé (PID)..... 15
Utilisation..... 15
Correcteur proportionnel..... 15
Correcteur intégral..... 18
Moyenne glissante..... 19
Carte Nucleo..... 20
Présentation..... 20
MBED..... 21

BILAN DU PROJET..... 22
Bilan du projet..... 22
Performances obtenues..... 22
Possibles améliorations..... 23
Retour de l'expérience..... 23
Photos de l’équipe..... 25
Photos du prototype..... 25

Schéma fonctionnel

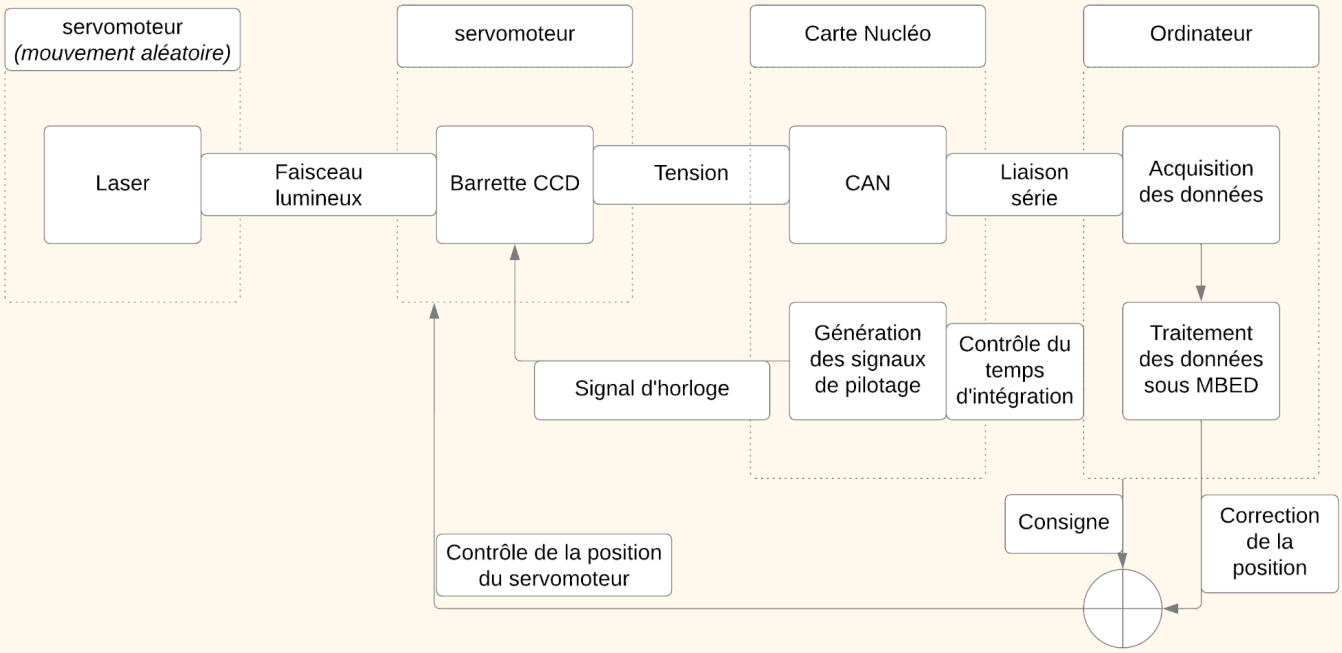


Schéma fonctionnel du système de banc optique pour l'asservissement d'un laser

Notre système est composé :

- D'un servomoteur qui porte le laser dont le mouvement est aléatoire
- D'un servomoteur contrôlé par la carte Nucleo qui porte la barrette CCD
- D'une carte Nucleo qui récupère les signaux de la barrette et contrôle le déplacement du servomoteur
- D'un PC qui traite toutes les données

Ainsi, le servomoteur portant le laser bouge avec un mouvement aléatoire contrôlé par un générateur de fréquence. La barrette CCD reçoit le faisceau laser. Celle-ci transmet une tension à la carte nucleo qui à l'aide qui convertisseur analogique numérique transmet enfin cette information au PC. Le traitement des données reçues se fait sur MBED. Depuis l'ordinateur on contrôle deux choses, tout d'abord la lecture des données de la barrette CCD qui se fait à l'aide de plusieurs fonctions temporelles et ensuite, on asservi la position du servomoteur pour suivre le faisceau laser.

Détail des missions

Les différentes missions à mettre en place seront donc les suivantes:

- Réussir à **contrôler le servomoteur,** à la fois celui qui porte le laser et qui aura donc un mouvement aléatoire, mais aussi celui qui porte la barrette CCD et qui aura donc une position asservie.
- Réussir à **contrôler la barrette CCD** et recevoir les informations provenant de celle-ci. Comme nous le détaillerons par la suite, cette mission s'avère plus difficile qu'il n'y paraît.
- Réussir à **asservir la position du servomoteur** portant la barrette CCD pour que celle-ci suive en continu le laser.

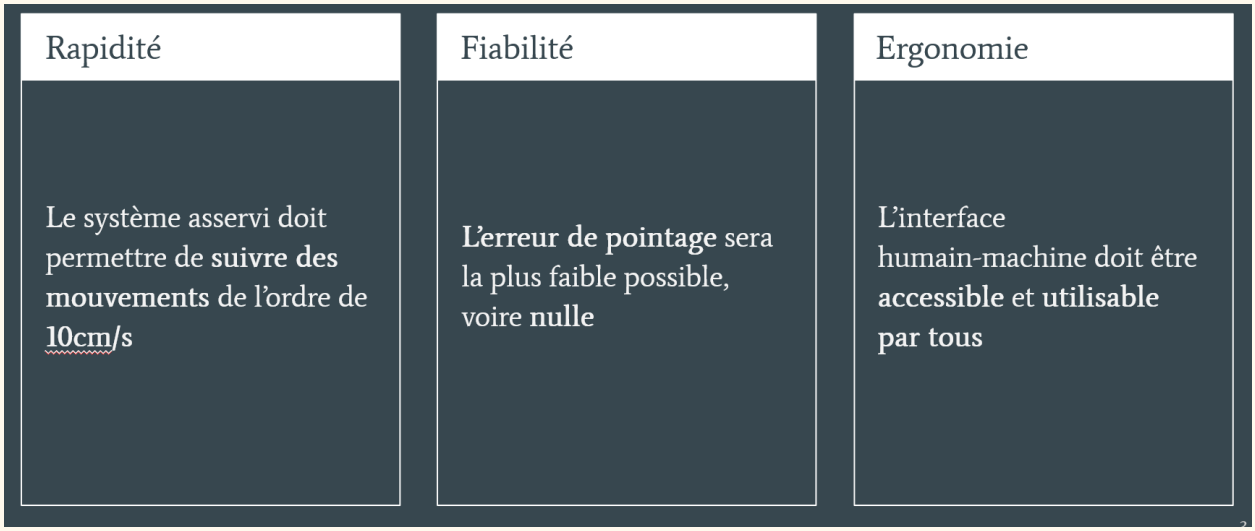
Contraintes

Voici les principales contraintes données par SOLEC:

Asservissement	Servomoteur	Coefficients correction
L'asservissement et la correction se feront numériquement, par le biais d'un PID numérique.	Les servomoteurs utilisés seront des servomoteurs classiques.	Les coefficients du correcteur devront pouvoir être modifiés en temps réel.

Contraintes fournis par l'équipe SOLEC

Cahier des charges



Cahier des charges fournis par l'équipe SOLEC

LES COMPOSANTS ET LEUR UTILISATION

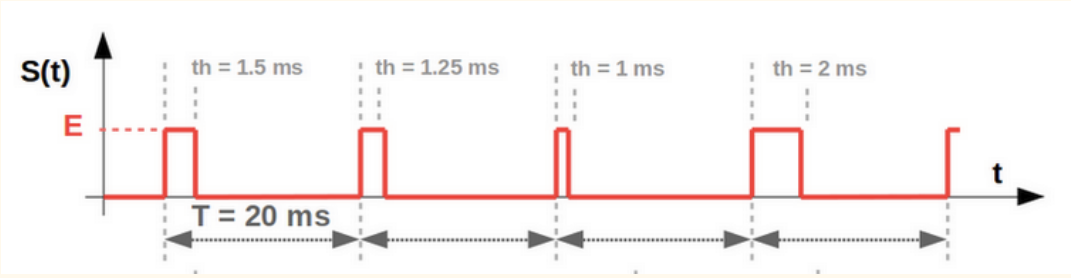
Servomoteur

Fonctionnement

Présentation

Le servomoteur est un organe essentiel de notre prototype. Comprendre son fonctionnement et l'étendue des ses capacités est essentiel pour la réussite de notre projet.

Le servomoteur est un organe servant à diriger le mouvement d'un moteur, d'un engin. Il est contrôlé par un signal créneau de largeur variable et de **période fixe de 20 ms**.



signal de commande du servomoteur

Pour une largeur d'impulsion de 1.5ms le servomoteur va rester immobile. Pour une largeur inférieure à 1.5ms il bougera dans un sens et pour une largeur supérieure à 1.5ms il bougera dans l'autre.

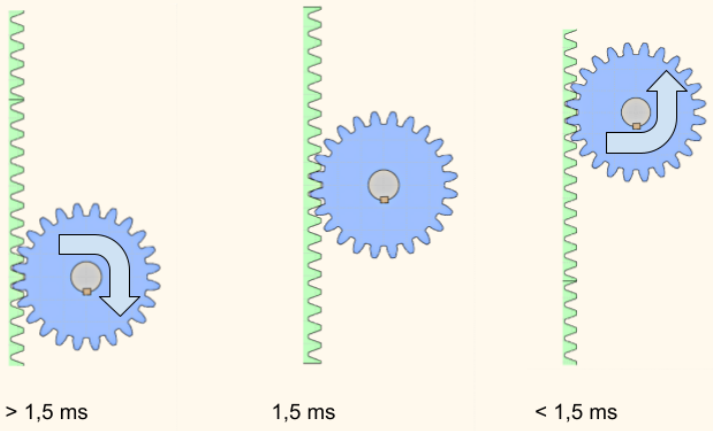


Schéma du fonctionnement du servomoteur en fonction de la largeur d'impulsion

Réalité

Dans les faits, il a fallu calibrer notre servomoteur pour nous assurer de la valeur pour laquelle il restait parfaitement immobile, 1.530ms dans notre cas. Nous avons aussi dû déterminer les valeurs maximale et minimale de notre largeur d'impulsion afin que le servomoteur ne sorte pas de son rail, entre 1ms et 2ms dans notre cas.

Dans les faits il suffit de tester différentes largeurs d'impulsion avec mbed ou avec le générateur de fréquence, et d'observer les mouvements du servomoteur.

TIPS: Chaque servomoteur fonctionne relativement de la même manière, mais ils ont tout de même leurs spécificités et il ne faut pas hésiter à tester les différents paramètres avant de continuer dans le projet.

Utilisation

Branchement

Le servomoteur à besoin d'être alimenté par un courant continu pour fonctionner. L'alimentation continue (souvent de 5V ou 6V) doit être branchée sur le fil rouge. Une masse doit être ajoutée sur le fil noir (ou marron). La commande du servomoteur doit être branchée sur le fil jaune (ou vert, blanc selon les cas).

MBED

La commande du servomoteur doit être envoyée sur une sortie de type **Pwout.** La fonction de contrôle de la période du servomoteur doit ensuite être fixée à 20 ms grâce à la fonction **servo_mot.period_ms()** (servo_mot étant le nom de la sortie de type Pwout fixée au préalable). Enfin, il est possible de contrôler le mouvement du servomoteur grâce à la fonction qui contrôle la largeur de l'impulsion envoyée au servomoteur : **servo_mot.pulsewidth_us()**.

Exemple:

Voici un exemple de code Mbed permettant au servomoteur de se déplacer dans un sens, avec un pas régulier (de 10 us), toutes les 50 ms:

```
#include "mbed.h"

PwmOut servo_mot(D10); // déclaration de la sortie de contrôle du
servomoteur
int i;

int main() {

    servo_mot.period_ms(20); // On fixe la période du servomoteur à 20ms
    servo_mot.pulsewidth_us(1530); // On initialise sa position au centre
    i=0;

    while(1) {

      servo_mot.pulsewidth_us(1500+i); // Modification de la largeur
d'impulsion du signal permettant le mouvement du servomoteur

      i=i+10; // on augmente le pas

      wait_us(50000); // on attends 50ms avant de bouger le servomoteur à
la position suivante

}
}
```

Générateur de fréquence

Il est possible de contrôler le servomoteur grâce à un générateur de fréquence (GBF). Pour se faire il suffit d'envoyer au servomoteur un signal de type **pulse**. La **période** doit ensuite être fixée à **20 ms,** période de fonctionnement du servomoteur. Le **hi-level** doit être fixé à **5V** et le **lo-level à 0V**. Enfin la largeur de l'impulsion, appelée **width**, doit être modifiée entre le minimum et la maximum de temps déterminé au préalable. Dans notre cas, celle-ci variait entre 1ms et 2ms.

En résumé il faut:

Paramètres	Valeurs
mode	pulse
période	20 ms
hi-level	5V
lo-level	0V
width	[1ms; 2ms]

Tableau récapitulatif des paramètres nécessaire pour contrôler le servomoteur grâce au GBF

Contrôler le servomoteur grâce au GBF va être principalement utilisé ici pour le servomoteur portant le laser qui possède un mouvement aléatoire.

Barrette CCD

Fonctionnement

Présentation

Les capteurs CCD sont des capteurs de lumière utilisés dans de nombreux systèmes de prise de vue numérique. Le principe de fonctionnement est le suivant: la photodiode va venir capter des électrons et par effet photoélectrique celle-ci va générer des charges. Les charges sont ensuite collectées dans des puits de potentiels contenus dans chaque pixel. Dans le cas d'une CCD les charges sont ensuite transférées et convertis en tension.

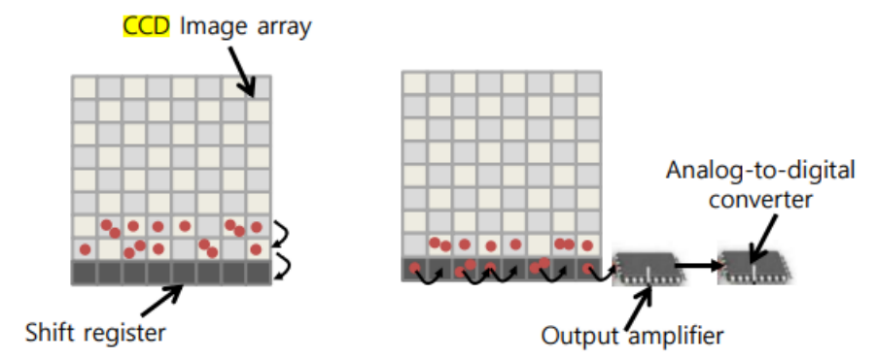


Schéma explicatif du fonctionnement d'une barrette CCD

Dans le cas d'une barette CCD l'architecture est composée de lignes de pixels qui se remplissent et se vident pas à pas. Dans le cas de notre barrette, celle-ci va être composée de 64 pixels par ligne. On remplit ainsi pas à pas les 64 pixels d'une ligne que l'on vide ensuite pour passer à la ligne suivante.

Utilisation

Le modèle de barrette que nous allons ici utiliser est le **tsl201r.** Sa notice d'utilisation détaillée est disponible dans l'onglet suivant:

https://html.alldatasheet.fr/html-pdf/102326/ETC/TSL201R/54/1/TSL201R.html

Signaux de la barrette CCD

Pour faire fonctionner la barrette CCD, et pouvoir lire les informations fournies par celle-ci, il est nécessaire de lui envoyer deux signaux via une carte nucléo:

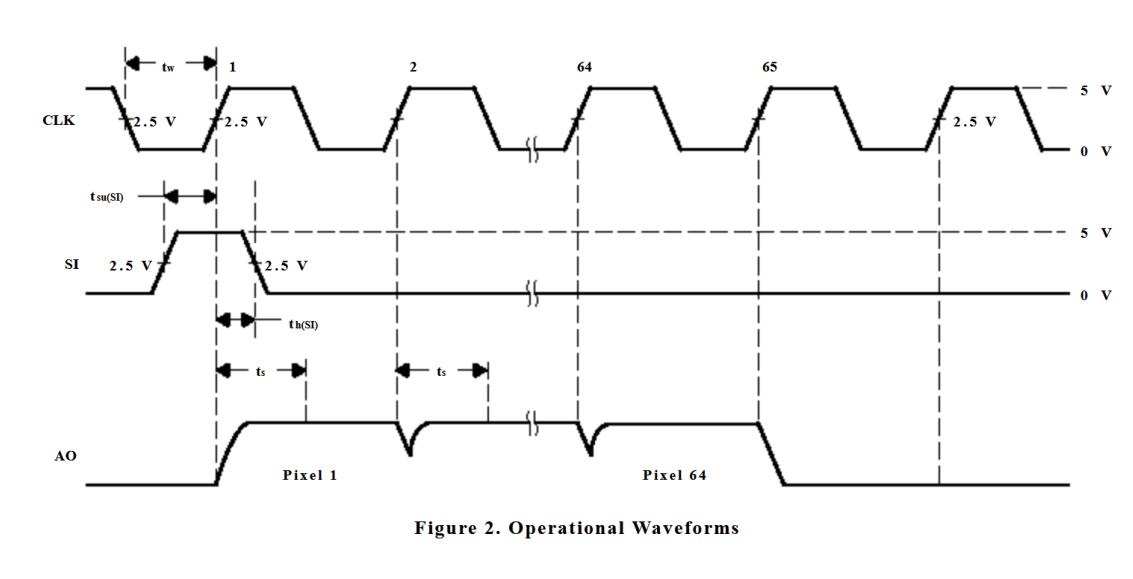


Figure 2. Operational Waveforms

Allure des signaux envoyés et reçu par la barrette CCD

- Le premier signal, appelé signal horloge ou **clock,** est un signal créneaux qui va nous permettre d'effectuer le remplissage des 64 pixels. Celui-ci doit être composé de 64 créneaux dont la période doit être choisie minutieusement afin qu'à la fois, on ne sature pas les pixels et qu'il soit possible d'obtenir assez de signal pour en tirer des informations.

TIPS: Afin de vous assurer que votre période à bien été choisie il faut tout d'abord que vous vous référez à la notice d'utilisation de la barrette CCD. Dans notre cas, la fréquence du signal horloge devait être entre 5 et 5000 kHz. Comme vous pouvez le constater c'est un intervalle de valeur assez large, il ne faut donc pas hésiter à regarder à l'oscilloscope l'allure du signal de sortie de votre barrette CCD.

- Le second signal, appelé **SI** ou SI1, est un signal composé d'un unique créneau qui apparaît au bout de 64 créneaux du signal clock. Celui-ci va permettre de passer à la ligne de pixel suivante de la barrette CCD. Il permet donc de passer à un nouveau cycle de lecture.

  MBED

Afin de générer les signaux horloge et SI nous avons utilisé une fonction d'interruption qui était déclenchée toute les **600ns,** soit à une fréquence de **1700 kHz.** Pour écrire les créneaux nous avons utilisé un compteur conditionné sur sa parité:

compteur pair → créneau montant

compteur impair → créneau descendant

Dans les faits le code MBED est le suivant:

Déclaration:

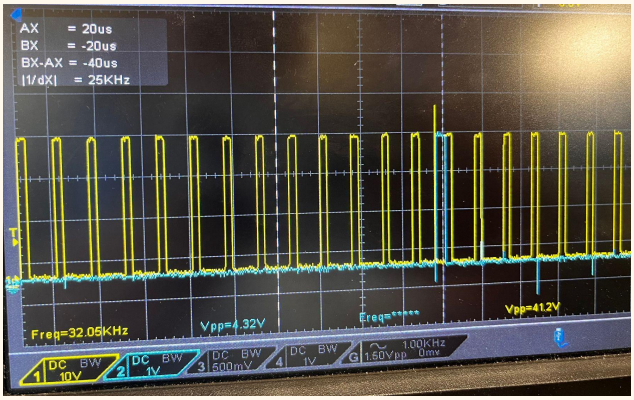
```
AnalogIn analog_in(A0); //signal de sortie de la barrette ccd

DigitalOut digital_out(A2); //signal envoyé sur la ccd pour déclencher le nouveau cycle (SI)

DigitalOut horloge(D13); // signal envoyé sur la ccd pour le remplissage des 64 pixels (horloge)
```

Fonction d'interruption:

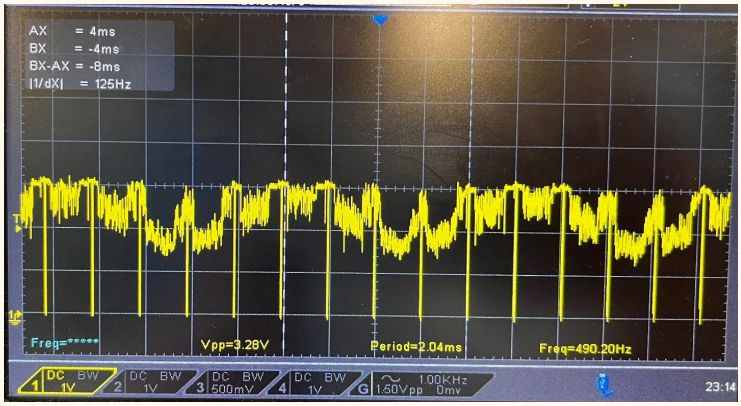
```
void toggle_horloge() // fonction d’interruption

{

// Début du cycle de 64 pixels
if (cpt!=N){

    if (cpt & 1){ //si impair
      horloge.write(0); // créneau horloge descendant
      digital_out.write(0); // valeur nulle pour le signal SI
      cpt=cpt+1;

    // Lecture du signal provenant de la barrette CCD
       if(j < 64)  { t[j]=analog_in.read();
            j=j+1; }

    }
    else {
       horloge.write(1); // créneau horloge montant
       digital_out.write(0); // valeur nulle pour le signal SI
        cpt=cpt+1;

    }
}
// si on arrive à 64 on déclenche le signal SI

if(cpt==N){
    digital_out.write(1); // déclenchement du signal SI

    // on réinitialise les compteurs à zéro pour le nouveau cycle
    cpt=0;
    j = 0;
}
```

Etape de vérification

Comme nous l'avons vu précédemment, il ne faut pas hésiter tout au long du projet à vérifier l'allure des signaux obtenus à l'oscilloscope. Nous avons ainsi obtenu les signaux suivants lors de nos tests:



Allure du signal horloge (en jaune) et SI (en bleu)



Allure du signal de sortie de la barrette CCD

⚠ ATTENTION: La vérification des valeurs obtenues en sortie de la barrette CCD n'est cependant pas possible avec la fonction **printf.** En effet, la fonction printf qui permet l'affichage en direct des données sur le console Teraterm n'est ici pas utilisable directement dans le fonction d'interruption. Cela vient du fait que la fonction printf est une fonction très gourmande en temps, or comme nous l'avons expliqué précédemment, la fonction d'interruption est très sensible temporellement parlant. Si l'on vient ajouter un printf dans cette fonction, l'on vient alors perturber tout le fonctionnement de notre système. Les printf ne doivent donc être utilisés que dans le main() de notre code.

Asservissement

Présentation

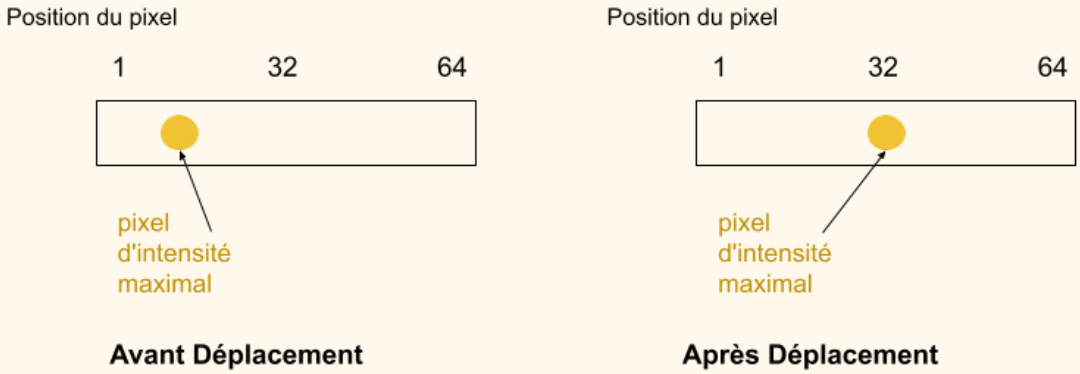
Correcteur proportionnel intégrateur dérivé (PID)

Afin d'asservir la position du servomoteur, pour que la barrette CCD puisse suivre la position du laser, il a fallu mettre en place un correcteur proportionnel intégrateur dérivé, aussi appelé PID. Ce correcteur est un système de contrôle permettant d'améliorer les performances d'un asservissement.

Utilisation

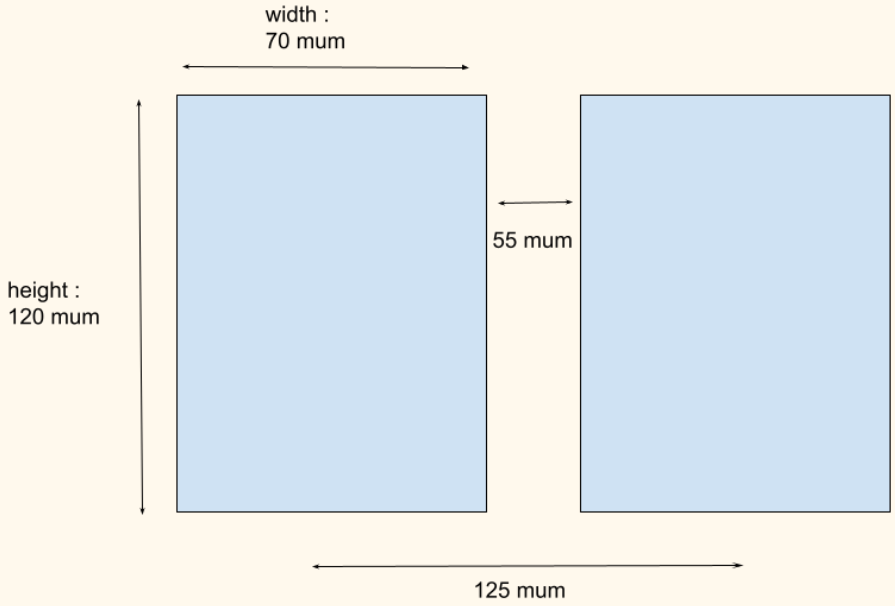
Correcteur proportionnel

La première étape du correcteur est d'obtenir une consigne proportionnelle au mouvement du servomoteur. Pour ce faire nous avons décidé de repérer la position du pixel d'intensité maximal de notre barrette et de nous déplacer de la distance nécessaire pour que ce pixel d'intensité maximal se retrouve au centre de notre barrette. Cela équivaut à ramener le spot laser au centre de la barrette.



Correction de la position du laser au centre de la barrette après déplacement

Afin de déterminer la valeur du déplacement nécessaire, il a fallu s'intéresser aux dimensions de notre barrette ccd spécifiées dans la notice d'utilisation:



Dimension des pixels de notre barrette CCD

Grâce à ces dimensions, on sait maintenant que le déplacement nécessaire pour remettre le laser au centre de notre barrette est de:

(32-position du pixel maximum)*125um

Il faut maintenant retranscrire cette consigne de position en consigne de largeur d'impulsion pour pouvoir asservir la position du servomoteur. Pour ce faire, nous avons calculé les déplacements induits en position pour différentes largeurs d'impulsion. Nous avons ainsi obtenu un déplacement de 1,8 cm pour un écart de largeur d'impulsion de 0,5 ms. Tout cela nous a ainsi permit d'obtenir la valeur du coefficient proportionnel de notre correcteur:

Kp=125*500/(1,8e4)=3,47 us

Au final notre consigne d'asservissement proportionnel est donc la suivante:

Uc= (32-position du pixel maximum)*Kp [us]

Recherche du pixel de valeur maximale

La rédaction du code débute avec la recherche de la position du pixel d'intensité maximale. Cette recherche est effectuée par un balayage classique de la liste de valeurs.

TIPS: Si vous souhaitez optimiser au maximum votre système, il est possible de rechercher la position du pixel d'intensité maximal par une méthode de barycentre (recherche du point d'équilibre). Celle-ci sera plus précise et plus rapide et permettra d'éviter le problème causé par la saturation ainsi que ceux causés par le bruit. Cette méthode est cependant plus longue à mettre en place.

Voici le code Mbed associé:

```
while(1) {

       m=0;// initialisation de la valeur maximale

      // Recherche de la position du pixel de valeur maximale

        for(int k = 0; k < 64; k++){ // boucle qui balaye la liste de
valeur des pixels
            if (t[k]>m){
                m=t[k];
                position_max=k; // la position correspond au rang du pixel
maximal
             }

        }
```

Asservissement proportionnel

L'asservissement proportionnel passe par deux étapes:

- On calcule l'écart à la position centrale (32) de la position du pixel maximal. Cet écart sera appelé **delta** dans notre code.
- On modifie la largeur de l'impulsion envoyée au servomoteur d'un facteur **delta*Kp**

Le code associé est le suivant:

```
// asservissement proportionnel de la position

delta=32-position_max; // Ecart à la position centrale

if (t[position_max]>6000){ //seuillage
     position_servo=1530-delta*Kp; // modification de la largeur
d'impulsion
     servo_mot.pulsewidth_us(position_servo);  }
```

TIPS: Il est intéressant d'envisager un seuillage lors de l'asservissement de la position du servomoteur pour éviter qu'il ne se déplace alors qu'il ne capte que du bruit ou de la lumière ambiante. Pour se faire n'hésitez pas à observer la valeur obtenue lorsque la barrette est positonnée dans le noir. Dans notre cas, la valeur de seuillage est fixée à 6000.

Correcteur intégral

Même si vous optimisez au maximum votre correcteur proportionnel, vous observerez que dans les faits votre servomoteur oscille autour de la position du laser. Cela provient du fait que l'erreur statique de votre système n'est pas nulle. Pour la réduire au maximum, un intégrateur est nécessaire.

L'intégrateur est mis en place en calculant en continu la moyenne des positions des pixels de valeur maximale. Cette erreur est ensuite prise en compte dans la valeur de delta et donc dans la correction de la largeur de l'impulsion du servomoteur.

Au niveau du code, il suffit d'ajouter une ligne dans le calcul de la position du pixel de valeur maximale afin de calculer la moyenne en continue de l'écart à la position centrale:

Calcul de l'erreur intégrale:

```
while(1) {

       m=0;// initialisation de la valeur maximale

      // Recherche de la position du pixel de valeur maximale

        for(int k = 0; k < 64; k++){ // boucle qui balaye la liste de
valeur des pixels
            if (t[k]>m){
                m=t[k];
                position_max=k; // la position correspond au rang du pixel
maximal
             }
      //Calcul de l'erreur statique

           erreur_inte=(erreur_inte+32-position_max)/cpt_inte;// moyenne
de   l'écart à la position centrale
            cpt_inte=cpt_inte+1; // nombre de cycle effectué

        }
```

Prise en compte dans le correcteur (delta):

```
// asservissement proportionnel et integral de la position

delta=32-position_max-erreur_inte; // Ecart à la position centrale

if (t[position_max]>6000){ //seuillage
      position_servo=1530-delta*Kp; // modification de la largeur
d'impulsion
      servo_mot.pulsewidth_us(position_servo);  }
```

Moyenne glissante

Nous avons décidé dans notre équipe d'ajouter une étape en plus dans notre asservissement, celle de la moyenne glissante. Cette moyenne glissante va fonctionner de la même manière que celle présentée précédemment dans la partie correction intégrale, sauf que celle-ci ne s'effectue que sur quelques cycles au lieu d'être en continue.

Cette moyenne glissante effectuée sur N cycles permet d'asservir la position un peu plus précisément que sur un cycle comme c'est le cas avec le correcteur proportionnel. On va ici calculer la position moyenne du pixel d'intensité maximale sur N cycles. La valeur de N doit être adaptée en fonction de la rapidité du mouvement du temps de détection, de calcul etc.

Le code associé est le suivant:

Moyenne glissante sur N boucles

```
// calcul de la position moyenne du max sur N boucles
        if(cpt_moy<N_moy){
            t_pos[cpt_moy]=position_max;// on remplit en continu en
tableau avec les positions des pixels maximals
            moy=0;

            for(l=0;l<N_moy;l++) {
                 moy=moy+t_pos[l]/N_moy;} // on calcule la moyenne du
tableau
                 cpt_moy=cpt_moy+1;
        }
        wait_us(300);

        if(cpt_moy==N_moy){ //on remet le compteur à zéro arrivé à N
            cpt_moy=0;

        }
```

Prise en compte dans le correcteur (delta):

```
// asservissement de la position

delta=32-position_max-erreur_inte+(32-moy); // Ecart à la position
centrale

if (t[position_max]>6000){ //seuillage
     position_servo=1530-delta*Kp; // modification de la largeur
d'impulsion
     servo_mot.pulsewidth_us(position_servo);  }
```

Carte Nucleo

Présentation

La carte de fonctionnement que nous allons utiliser est une carte Nucléo-L476RG. Cette carte est constituée de plusieurs broches entrées ou sorties analogiques ou numériques. Voici un schéma récapitulatif du type des différentes entrées et sorties.

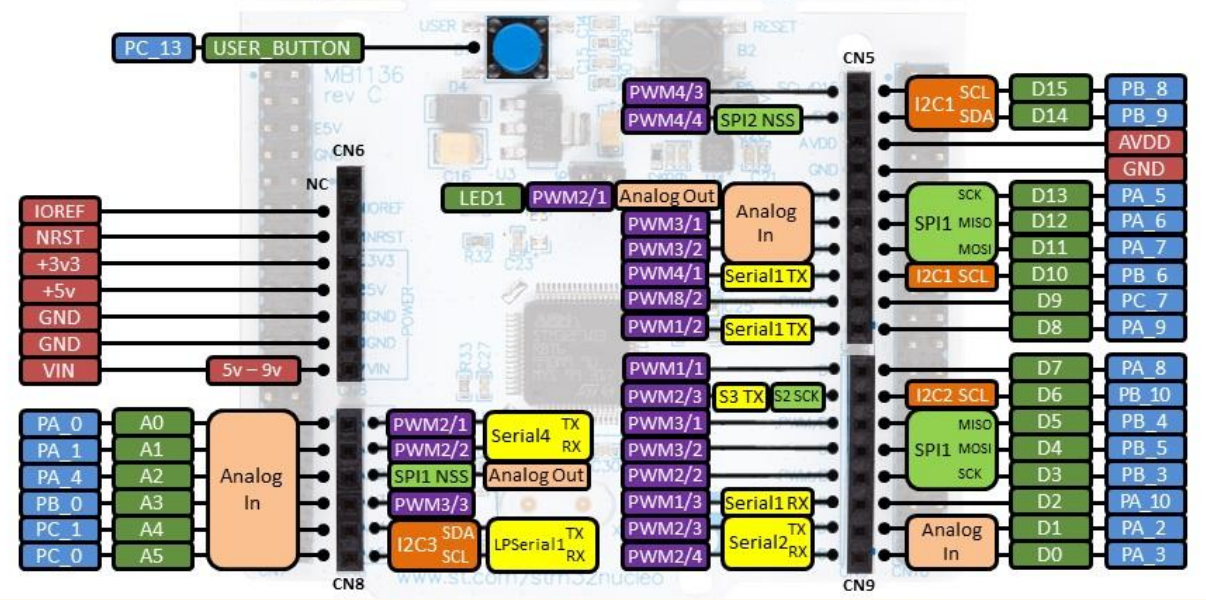


Schéma du type des broches de la carte Nucleo

MBED

Pour la barrette CCD voici les entrées et sorties à définir. Seul le type défini est important, donc avant de connecter une broche il faut toujours vérifier son type. Voici l'entête de notre code:

```
AnalogIn analog_in(A0); //signal de sortie de la barrette ccd
DigitalOut digital_out(A2); //signal envoyé sur la ccd pour déclencher la lecture
DigitalOut horloge(D13); // signal envoyé sur la ccd pour le remplissage des 64 pixels
```

Pour le servomoteur, voici la commande à définir.

```
PwmOut servo_mot(D10); // commande du servomoteur
```

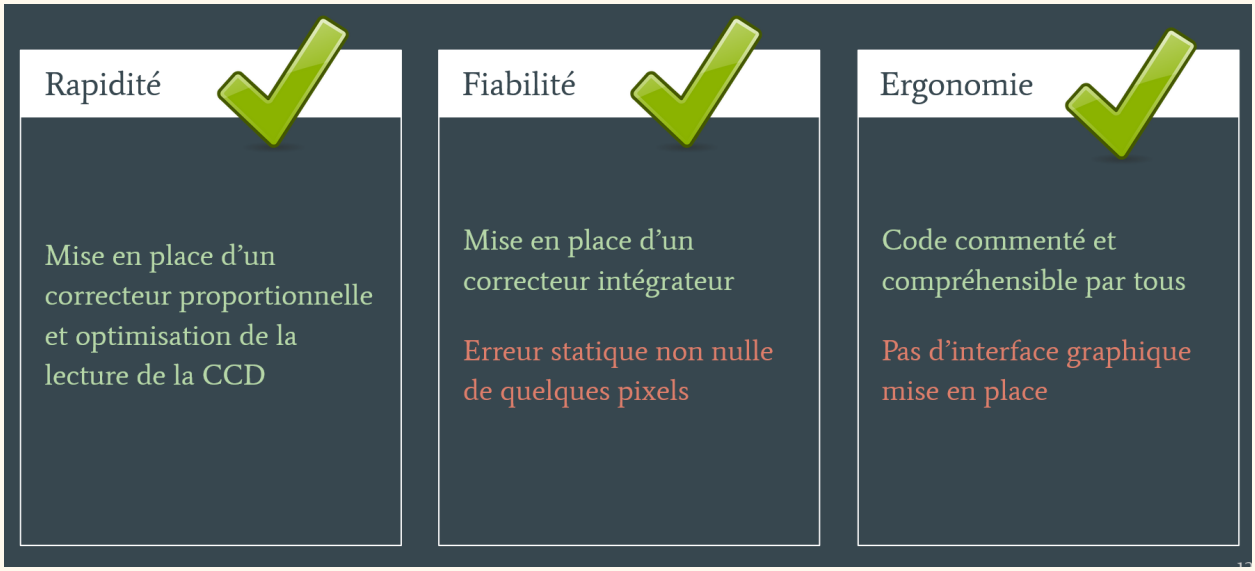
ATTENTION: Si teraterm vous affiche des symboles d'origine non-identifié, il est probable que le type de la broche ne soit pas le bon, ou alors que la vitesse d'affichage ne soit pas la bonne. A vérifier dans le programme **app.johnson**.

BILAN DU PROJET

Bilan du projet

Performances obtenues

Notre équipe est relativement fière des résultats obtenus pour notre prototype. Le servomoteur suit effectivement le laser sur toute la longueur du rail. Celui est relativement rapide et fiable grâce à la mise en place d'un correcteur proportionnel et intégral et grâce à l'optimisation du temps de lecture de la CCD. La prise en main du code est relativement simple grâce aux nombreux commentaires que nous avons ajoutés.



Bilan des performances obtenues pour notre prototype

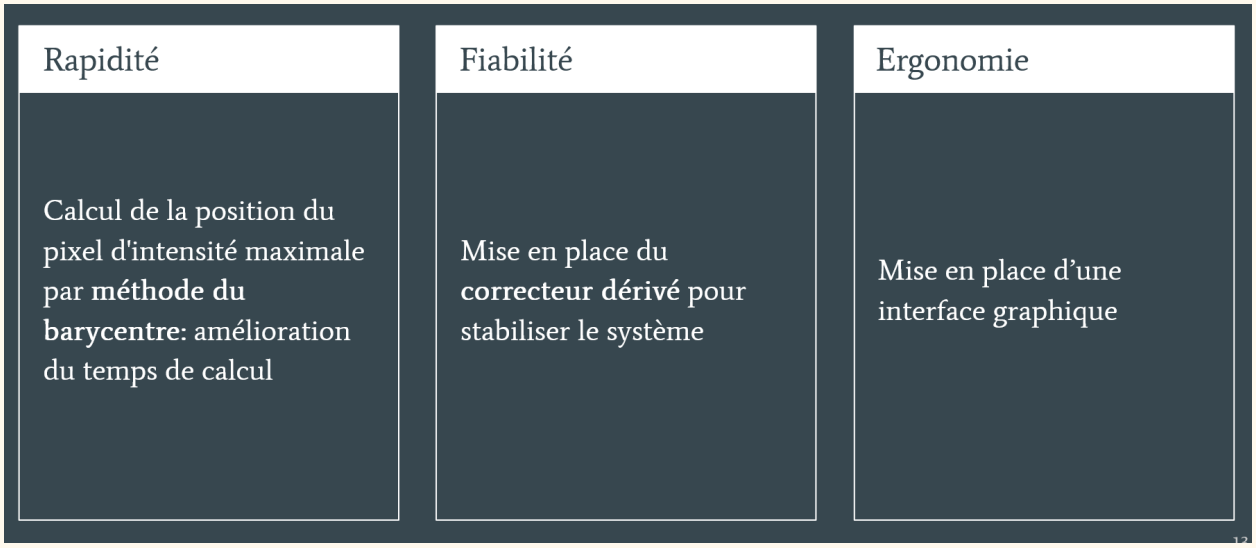
Comme vous pouvez l'observer sur le bilan des performances obtenues, la plupart des points du cahier des charges ont donc été validés mais il reste tout de même quelques points pas totalement accomplis, voire non traités.

Tout d'abord nous avons pu observer que notre erreur statique n'était pas totalement nulle car notre barrette CCD oscillait de quelques pixels autour de la position du laser. L'erreur statique nulle n'était pas forcément demandée dans notre cahier des charges mais il était demandé de la réduire au maximum, voire de la rendre nulle. Nous avons donc fourni notre maximum, celui-ci ne s'avère pas nul.

Le second point, cette fois-ci non traité, est la mise en place d'une interface graphique. Faute de temps et compte tenu du nombre de membres dans l'équipe (2), nous n'avons pas eu le temps de traiter cette partie. Nous avons cependant cherché à optimiser au maximum notre code et à le commenter au maximum pour que celui-ci soit le plus compréhensible.

Possibles améliorations

Etant donné qu'il faut garder à l'esprit que ce projet risque d'évoluer après notre passage, nous avons souhaité vous transmettre quelques pistes de possibles améliorations du prototype:



Possibles améliorations du prototype

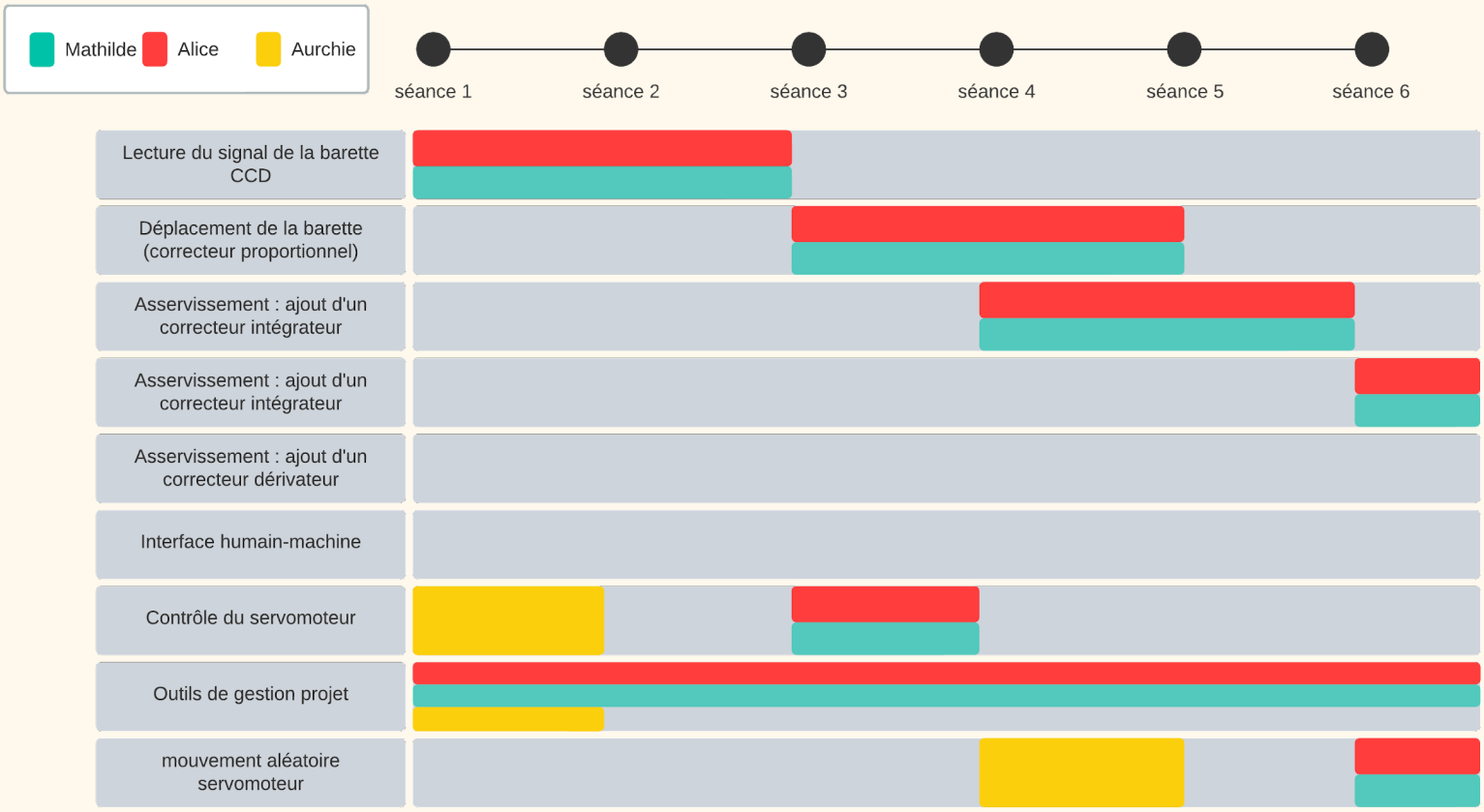
Comme nous vous l'avons déjà conseillé au préalable dans ce compte-rendu, une première piste d'amélioration du prototype est le calcul de la position du pixel maximum via la méthode du barycentre. Cette méthode est censée fournir un résultat de manière plus précise et plus rapide.

Une seconde piste d'amélioration est la mise en place d'un correcteur dérivé. Celui-ci est censé pouvoir prédire les futures évolutions du système et donc pouvoir stabiliser celui-ci.

Enfin au niveau de l'ergonomie, la possible amélioration est plutôt évidente ici, il s'agit de la mise en place d'une interface graphique.

Retour de l'expérience

En ce qui concerne notre retour sur l'expérience. Tout d'abord nous sommes ravies que toute l'équipe soit satisfaite du projet et fière de ce qui a été accompli. L'un des premiers critères de base émis au tout début du projet était: "Le projet est réussi si toute l'équipe est contente du travail fourni". Être fière du travail accompli est un point mis à l'honneur depuis le début du projet.



Calendrier des missions de notre équipe

En ce qui concerne notre calendrier, celui-ci a été réparti en fonction des missions à effectuer. La plupart des missions ont été effectuées à deux et dans l'ordre car pour notre projet il était relativement compliqué de séparer les missions et de ne pas les effectuer dans un certain ordre. L'asservissement n'a de sens qu'une fois que la lecture de la ccd et que le servomoteur ont été contrôlés. De même l'interface ne doit être réalisée qu'une fois le projet fini. Les missions ont donc été réalisées en duo a chaque fois et cela nous a permis à toutes les deux de pouvoir toucher un peu à tous les aspects du projet.

Photos de l’équipe

Photos très qualitatives d’une équipe plus que qualitative

Photos du prototype

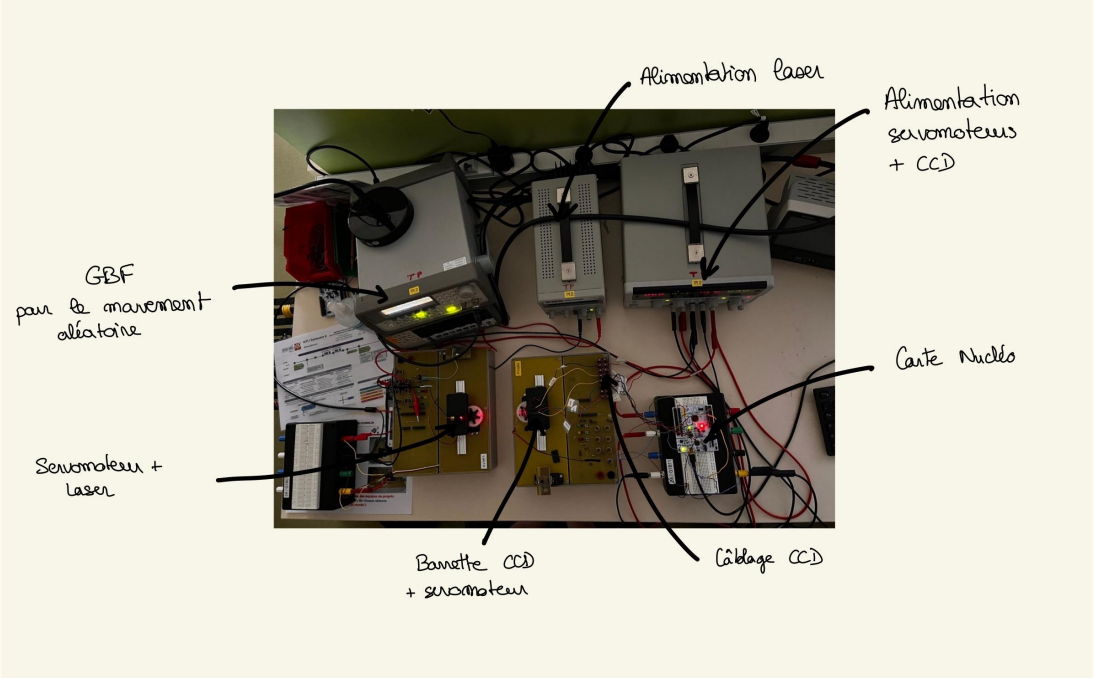


Photo de notre prototype