HAGUI Bacem - Département IMA - Année universitaire 2017/2018

Rapport de mi-parcours de Projet de Fin d'Études

Projet de marionnette déformable

Encadrants : SANZ-LOPEZ Mario
DEQUIDT Jeremie

INRIA Lille
Du 26/09/2017 aux 24/02/2018

Remerciement

Je souhaite remercier tout d'abord l'ingénieur recherche et développement de l'équipe, Mario Sanz Lopez, de m'avoir accueilli, encadré et aidé durant ce projet. Je voudrais aussi remercier M. Jeremie Dequidt de m'avoir présenté le projet et avoir répondu à mes questions pour mieux choisir mon sujet de projet de fin d'études. Je remercie également M. Félix Vanneste de son aide et conseille durant le projet. Enfin, je remercie le directeur de l'équipe, Christian Duriez, et toute l'équipe DEFROST de m'avoir accueilli parmi eux et pour l'atmosphère de travail unique et confortable crée.

Sommaire

Introduction

Lors de la cinquième année nous sommes amenés à travailler sur un projet de fin d'études dans l'école ou dans un organisme extérieur. J'ai choisit de participer à un projet dans le laboratoire INRIA (Institut National de Recherche en Informatique et en Automatique) au sein de l'équipe DEFROST.

L'Institut National de Recherche en Informatique et en Automatique est un établissement public à caractère scientifique et technologique (EPST) qui participe à nombreux domaines de recherche. Notamment la robotique, programmation, sécurité et beaucoup d'autres. L'équipe DEFROST a pour thème de recherche la robotique déformable. Je participe donc à un projet qui a pour but de réaliser une marionnette déformable utilisé pour faire des démonstrations lors des conférences ou éventements INRIA.

Dans ce rapport, je présenterai dans un premier temps le projet et le cahier des charges envisagé. Ensuite je vais décrire le travail que j'ai pu effectuer durant le projet ainsi que le travail restant.

I Présentation générale du projet

Description

Pour ce projet il s'agit de la mise en place d'un démonstrateur interactif dans lequel l'utilisateur pilotera une marionnette déformable à travers d'un LeapMotion. Le LeapMotion est utilisé pour capturer la forme de la main et ses mouvements. Ces informations sont ensuite traitées pour déterminer ce que l'utilisateur voudrais faire de la marionnette. Une simulation temps réel de la marionnette pilotera la version réelle du robot déformable grâce au logiciel simulateur SOFA.

Le contrôle de la marionnette (ici une pieuvre) est actuellement fait grâce à 12 moteurs monté sur une plaque fixé sur le dessus de la scène, cf photo 2. Deux moteurs sont dédiés à chacune des 5 tentacules, un moteur fait monter ou descendre le tentacule et l'autre moteur est responsable de soit boucler ce dernier ou non. Les deux moteurs restants ont pour but de contrôler les deux yeux du pieuvre. La liaison entre les moteurs et le corps de la marionnette est fait à travers des fils de pêche.

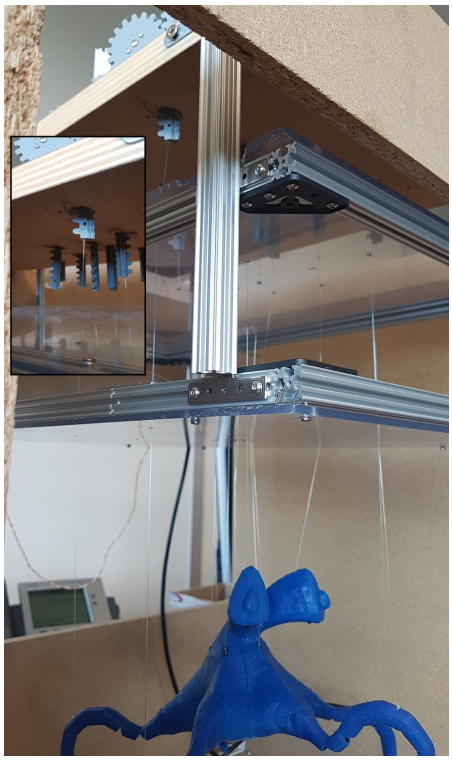

Photo 1: Démonstrateur complet.

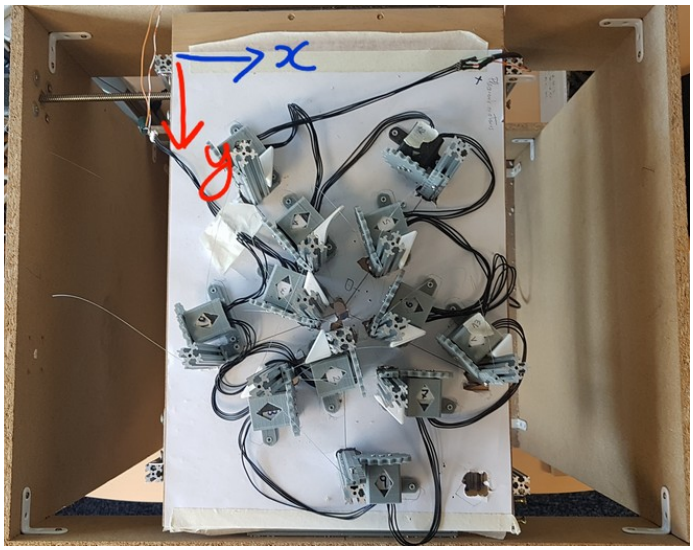

Photo 2: Plaque de moteurs.

Objectifs

L’objectif principale de ce projet est de rendre la scène plus modulaire et facile à manipuler. L’ingénieur recherche et développement de l’équipe m’as expliqué que la procédure de changement de marionnette prenait plusieurs heures et ensuite venait la calibration manuelle des actionneurs qui elle aussi prenait pas mal de temps. Le robot étant un démonstrateur interactif, il est indispensable d’avoir un mécanisme facile à gérer et de pouvoir facilement changer de marionnette.

La première étape est d'améliorer la version actuelle de la plate-forme d'actionnement et de trouver une meilleure disposition de moteurs permettant un accès facile à chacun d’entre eux. Ensuite il faudrait ajouter un système d'actionnement contrôlé pour faire coulisser la plate-forme dans le plan X dans un premier temps et potentiellement l’axe Y également, cf photo 2. Enfin, un choix de carte de contrôle doit être fait. Une carte Arduino est actuellement utilisée mais il est envisagé de passer à une carte plus puissante, Teensy 3.2 ou même 3.5 si besoin du à des soucis de vitesse de communication par port série.

Calendrier prévisionnel

Avec l'ingénieur recherche et développement de l'équipe, nous avons pu établir le calendrier prévisionnel suivant. Ce calendrier pourrait être mis à jour par la suite en fonction de l'avancement du projet et des éventuelles circonstances.

- **Début Novembre:** Mettre à jour l'organisation des moteurs sur la plaque et avoir la nouvelle plaque des moteurs de prêt.
- **Début Décembre:** Formation sur le logicielle de simulation et contrôle des corps déformable SOFA. Ainsi que se familiariser avec la scène contenant la marionnette et les capteurs utilisé pour détecter la main, sa forme ainsi que sa position.
- **Mi-décembre:** Prendre une décision concernant le micro-contrôleur à utiliser par la suite, soit continuer avec Arduino ou changer vers Teensy.
- **Début Janvier:** Équiper la plaque de moteurs pour avoir dans un premier temps un mouvement sur l'axe X, cf photo 2.
- **Fin Janvier:** Mapping de la main, c'est à dire d'associer la main et les doigts à certains parties de la marionnette pour pouvoir la contrôler et de prendre en compte un mouvement dans l'axe X.
- **Février:** Prospectif d'ajouter un mouvement de la plaque dans l'axe Y. Tester le système et faire des améliorations où c'est possible.

II Travail réalisé

Nouvelle plaque de moteurs

J'ai commencé ma contribution au projet par faire une nouvelle disposition de moteurs qui permet un accès plus facile à chacun pour les manipulations nécessaires. Cela dans le but de faciliter la calibration du démonstrateur ainsi que le câblage des moteurs. La nouvelle disposition devait aussi prendre moins de place que l'ancienne pour permettre de réduire la taille de la plaque si besoin. Ma plaque est de la même taille que l'ancienne actuellement mais les moteurs sont bien espacés, cela sera réduit par la suite si besoin. J'ai proposé la disposition illustrée en photo 3.

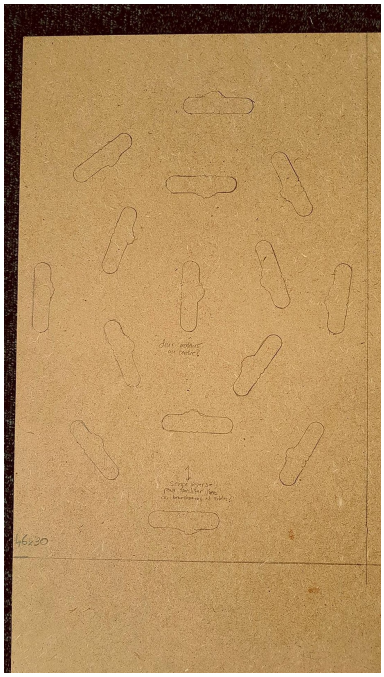

Photo 3: Proposition de nouvelle disposition.

Photo 4: Nouvelle plaque de moteurs.

Après réflexion, j'ai choisi d'opter pour une disposition circulaire. Cette disposition me permet d'avoir un accès facile à chaque socle moteur sans avoir un contact gênant entre les engrenages, cf photo 1. J'ai aussi ajouté un moteur central qui a pour but dans le cas de la pieuvre par exemple de la tenir en place, monter ou descendre entièrement. N'ayant pas de place idéale pour positionner ce moteur dans l'ancienne plaque, il a été remplacé par un

fil qui attache la marionnette à un point fixe de la scène, par conséquence, la hauteur de la marionnette était non-modifiable. Cela sera corrigé avec la nouvelle plaque.

Après une discussion avec l'ingénieur R&D, ma proposition a été accepté et j'ai pu poursuivre la démarche pour faire ma nouvelle plaque. J'ai dessiné sur une planche de bois les emplacements des moteurs pour s'assurer que la plaque pourrait tenir 13 moteurs avec les dimensions 46cm*30cm. La plaque été ensuite coupé avec les dimensions précisées par le service responsable.

J'ai ensuite eu accès aux modèles 3D des pièces qui seront utilisé pour fixer les moteurs sur la plaque. Après avoir suivi une formation sur l'utilisation des imprimantes 3D du laboratoire, j'ai effectué plusieurs impressions pour compléter les pièces déjà imprimé auparavant. Une fois que j'avais 13 exemplaires de chaque pièce, j'ai monté les moteurs pour finaliser la nouvelle plaque, cf photo 4. Ensuite j'ai démonté le démonstrateur pour remplacer l'ancienne plaque et faire les modifications nécessaires.

Formation SOFA et utilisation de la marionnette

Pour cette partie j'ai tout d'abord eu besoin de l'aide de la part des membres de l'équipe pour compiler le logiciel SOFA et avoir une courte formation d'utilisation. Apprendre comment réaliser une scène ou une simulation ne fait pas partie du projet, je me suis seulement intéressé aux bases du logiciel c'est-à-dire comment démarrer la simulation de la marionnette actuel (pieuvre), cf photo 5.

Un deuxième élément clé du projet est l'interface et librairie LeapMotion, cf photo 5 à droite. Nous pouvons voir la main représentée ainsi que les points clés en violet. Nous avons ici l'interface graphique qui nous aide à confirmer que les éléments LeapMotion sont correctement installé ainsi que calibrer ce-dernier pour avoir des mesures plus précises. Il faut également installer un package (SDK) qui nous aidera à extraire les informations qui seront utilisé dans le pilotage de la marionnette, comme par exemple les coordonnées des doigts par exemple.

Une fois toutes les outils nécessaire préparées, j'ai commencé à étudier le code de la première version de la marionnette. J'ai pu comprendre l'architecture globale du code ainsi qu'identifier les parties du code que j'aurai à modifier pour implémenter les nouvelles fonctionnalités de mon cahier des charges.

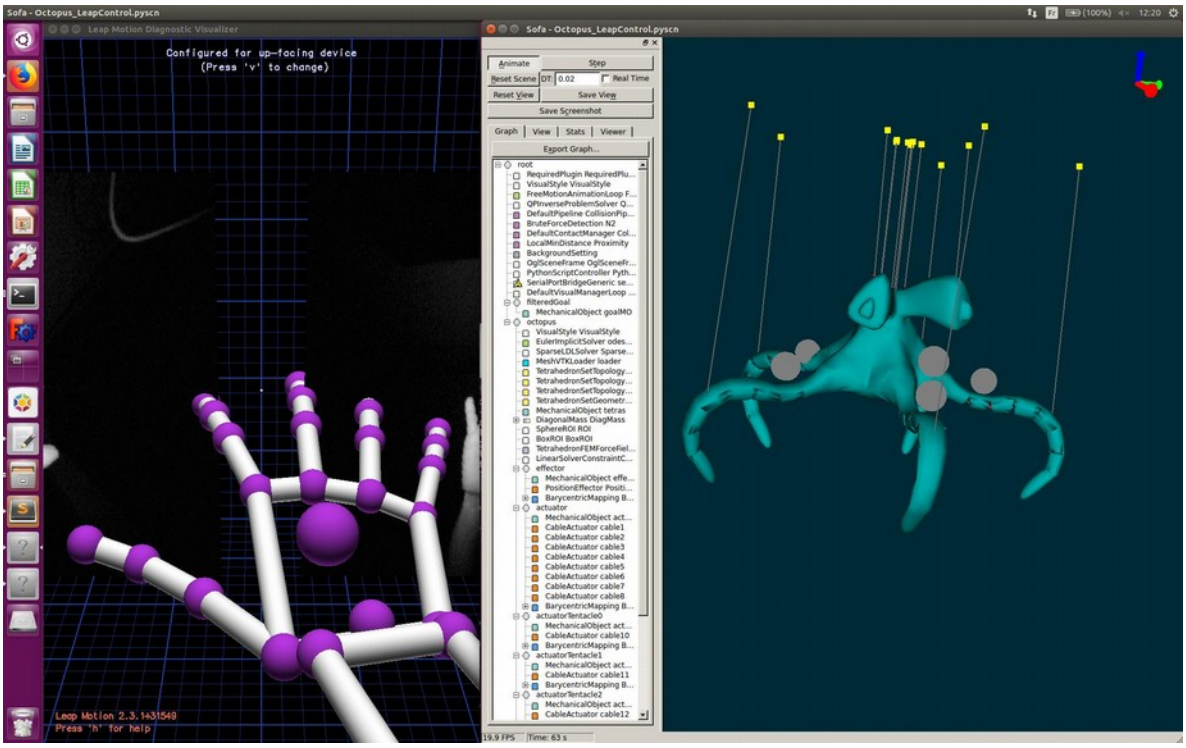

Photo 5: Interfaces LeapMotion et SOFA

Décisions pour la suite du projet

Après ma mi-soutenance j'ai eu quelques conseils et suggestions au niveau du code et choix de matériel pour améliorer certains aspects de performance du projet. Le premier conseil était d'utiliser une carte Arduino Mega au lieu de la carte Uno actuel pour éviter d'utiliser un port série software. Deuxièmement, de programmer l'Arduino en langage C et non avec l'IDE pour avoir une meilleure performance et éviter des éventuels problèmes d'efficacité. Et enfin de tenir compte du nombre de pas que le moteur pas-à-pas effectue pour estimer sa position au lieu d'utiliser le capteur ultrason HC-SR04. Lors de mon retour au laboratoire j'ai pu discuter avec l'ingénieur recherche et développement pour lui communiquer ces informations et savoir comment il voudrait poursuivre.

Concernant le choix entre un Arduino Mega ou Uno, l'ingénieur R&D a préféré rester sur une carte Uno pour la simple raison que dans le laboratoire ils ont beaucoup plus de cartes Uno que de cartes Mega. Il a voulu continuer avec une carte Uno pour tester ses limites et si cela n'était pas à la hauteur de nos attentes, une carte Mega sera utilisé à la place. Toujours dans le sujet de l'Arduino, l'ingénieur R&D a voulu continuer de le programmer avec l'IDE vu que le coût de changement de code sera trop élevé.

Pour le moteur pas-à-pas j'ai commencé par tester la méthode conseillée. Vous trouverez ci-dessous la partie du code de cette méthode qui nous poseras un problème si on utilise cette méthode.

```
steps=stepspermm*(positions[14]-stepperposition);
if (steps){
  analogWrite(PINstep,periode>>1);
  for (long int i=1; i<abs(steps);i++){
    delayMicroseconds(periode);
  }
analogWrite(PINstep,0);
}
stepperposition+=(steps/stepspermm);
```

La variable s*tepspermm* est égale à *200* pour le moteur que nous avons en disposition et *positions[14] est* la position souhaité par l'utilisateur dans l'axe X. Pour des grandes distances à parcourir, c'est-à-dire un grand nombre de *steps*, le temps de délai sera trop élevée et va empêcher le programme d'envoyer des commandes aux smart actuators. C'est pour cette raison la qu'il était décidé d'utiliser le capteur ultrason pour positionner la plaque de moteurs.

En conclusion, nous avons décidé de continuer avec une carte Arduino Uno programmé avec l'IDE et d'utiliser un moteur pas-à-pas avec un capteur ultrason pour détecter la position de la plaque mobile. Nous avons aussi décidé de mesurer les différents temps de réponse des différentes commandes d'actionnement de la bibliothèque Herkulex pour nous aider à mieux prendre une décision concernant les commandes à utiliser.

Identification des moteurs existants

Chacun des douze moteurs existants est lié à une partie de la pieuvre pour réaliser une action spécifique ; le moteur 1 par exemple doit faire plier le premier tentacule et le moteur 8 déplier ce dernier. Il est donc indispensable d'avoir dans chaque socle le moteur réalisant la tâche appropriée et que ce dernier soit connecté à la bonne partie de la pieuvre. Pour la procédure d'identification, j'ai écrit le code Arduino ci-dessous.

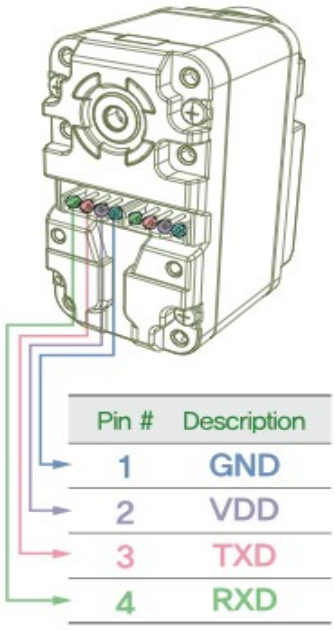

Photo 6: Branchement des Smart Actuators

```
#include <Herkulex.h>
int i;
void setup() {
  Serial.begin(115200); // set the baud rate
  Serial.println("Ready"); // print "Ready" once
  Herkulex.begin(57600, 12, 11);
}
void loop() {
  if(Serial.available()){
    Serial.println("Serial available");
    int i = Serial.read()-96;
    Serial.print("Value read i = ");
    Serial.println(i);
    Herkulex.reboot(i);
    delay(500);
    Herkulex.initialize();
    Herkulex.moveOne(i, random(0,1023), 2000, LED_BLUE);
  }
  delay(100); // delay for 1/10 of a second
}
```

Le programme commence par attendre que l'utilisateur entre un caractère dans le moniteur série d'Arduino. Le code recevant des valeurs ASCII, je considère que l'utilisateur doit entrer les lettres d'alphabet (a-z) et ensuite je soustrais 96 pour que la lettre « a » donne le numéro « 1 », « b » le numéro « 2 » et ainsi de suite. Ce numéro correspond à un numéro de moteur et est stocké dans la variable « i » pour actionner le moteur correspondant. Il faut ensuite connecter les moteurs un par un, comme illustré sur la photo 6, et tester pour quel identifiant ils répondent pour pouvoir les identifier. Après le redémarrage et initialisation du moteur, la commande **moveOne** de la bibliothèque Herkulex prends en paramètre l'identifiant du moteur, la position à atteindre, le temps d'exécution en Millisecondes (ici 2000ms) et la couleur que la LED devrait avoir.

Programmation du treizième moteur

Une des fonctionnalités à ajouter était de pouvoir contrôler la hauteur de la pieuvre. Pour cela il a fallu ajouter un treizième moteur attaché au centre de la marionnette. Un nouveau smart actuator a deux paramètres que nous devons changer avant de s'en servir, notamment son identifiant et sa vitesse de communication. L'identifiant par défaut est 253 avec un baud rate de 115200. Cette vitesse de communication n'est pas stable avec un Arduino Uno et il faut donc la changer à 57600 qui est plus adapté. Pour faire ces

modifications nous avons besoin d'un Arduino Mega pour sa disposition de plusieurs ports série et la possibilité de pouvoir utiliser le baud rate par défaut des moteurs. Une fois le moteur bien branché, nous avons tout d'abord changé son ID grâce à la fonction **void set_ID(int ID_Old, int ID_New)** qui prends en paramètre l'ancien et nouveau identifiants. L'ancien étant 253 et le nouveau 13. Ensuite nous utilisons la fonction **void writeRegistryEEP(int servoID, int address, int writeByte)** qui se chargera de changer la valeur du registre à l'adresse précisé. Dans notre cas nous avons à changer le registre du Baud Rate situé à l'adresse 4 et de changer sa valeur de 0x10 à 0x22 (les valeurs hexadécimaux signifient 115200 et 57600 respectivement). L'appel de fonction est donc le suivant : writeRegistryEEP(13, 4, 0x22). Le moteur est maintenant prêt à être utilisé. Ci-dessous est une photo démontrant l'état actuel de la nouvelle plaque de moteurs.

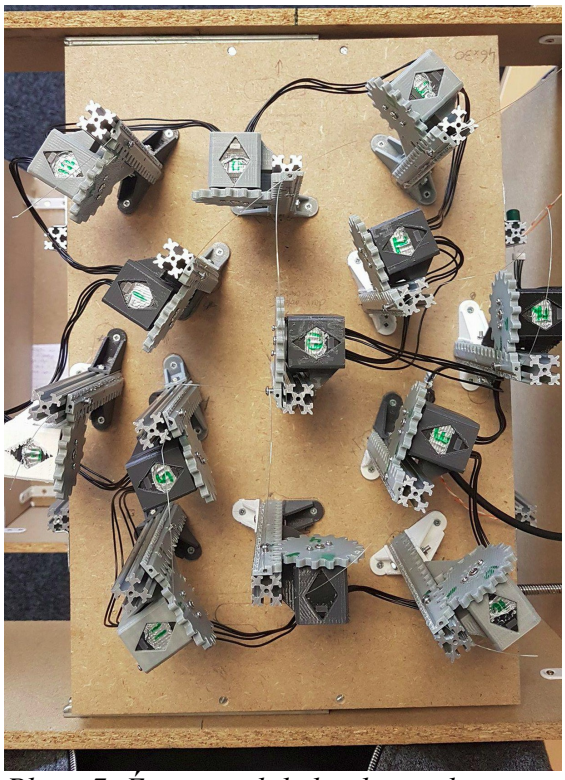

Photo 7: État actuel de la plaque de moteurs.

Test des temps de réponse des moteurs

Dans le but de tester la possibilité de continuer avec un Arduino Uno, la programmation avec l'IDE ainsi qu'ajouter un moteur pas-à-pas pour le mouvement dans l'axe X par la suite, nous avons voulu tester les temps de réponse de deux commandes d'actionnement de la librairie Herkulex. Les commandes étant : **Herkulex.moveOne(int servoID, int Goal, int pTime, int iLed)** et **Herkulex.moveAllAngle(int servoID, float angle, int iLed)** qui prépare plusieurs moteurs à un mouvement synchrone suivi par **Herkulex.actionAll(int pTime)** pour les actionner. Pour cela j'ai écrit un script python qui envoie un message par le port série pour lancer le test et ensuite reçois et enregistre les temps de réponse dans un fichier nommé **log.txt**. Vous trouverez ci-dessous le script python ainsi que le code Arduino pour tester la commande moveOne.

Code Arduino :

```
#include <Herkulex.h>
unsigned long time;
int positions[13];
void setup() {
  Serial.begin(9600); // set the baud rate
  Herkulex.begin(57600, 12, 11);
  Herkulex.initialize();

  for (int n=0; n<13; n++){
    positions[n]=random(0,1023);
  }
}
void loop() {
  if (Serial.available() > 0){
    Serial.read();
    for (int n=0; n<13; n++){
      Serial.println("Sending");
      time = millis();
      Serial.println(time);
      Herkulex.moveOne(n+1, positions[n], 2000, LED_BLUE);
      time = millis();
      Serial.println(time);
      Serial.print("Command sent to motor ");
      Serial.println(n+1);
    }}}
```

Script Python :

```
from time import sleep
import serial
ser = serial.Serial("/dev/ttyACM1", 9600) # Establish the connection on a specific port
sleep(2)
ser.write("1")
f = open('log.txt','w')
while True:
        f.write(ser.readline()) # Read the newest output from the Arduino
ser.close()
f.close()
```

Résultats :

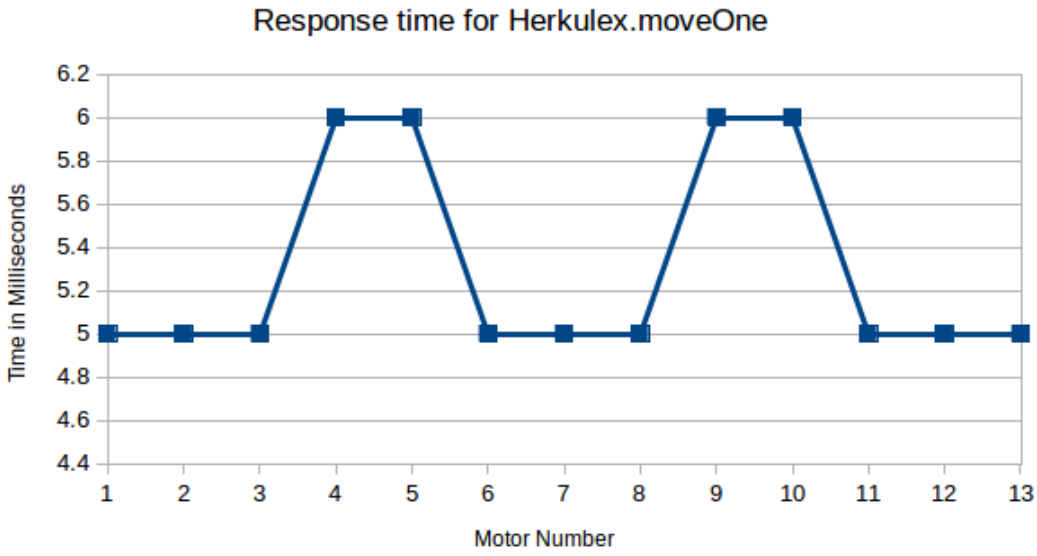

Photo 8: Un total de 69 Millisecondes pour actionner les treize moteur avec moveOne.

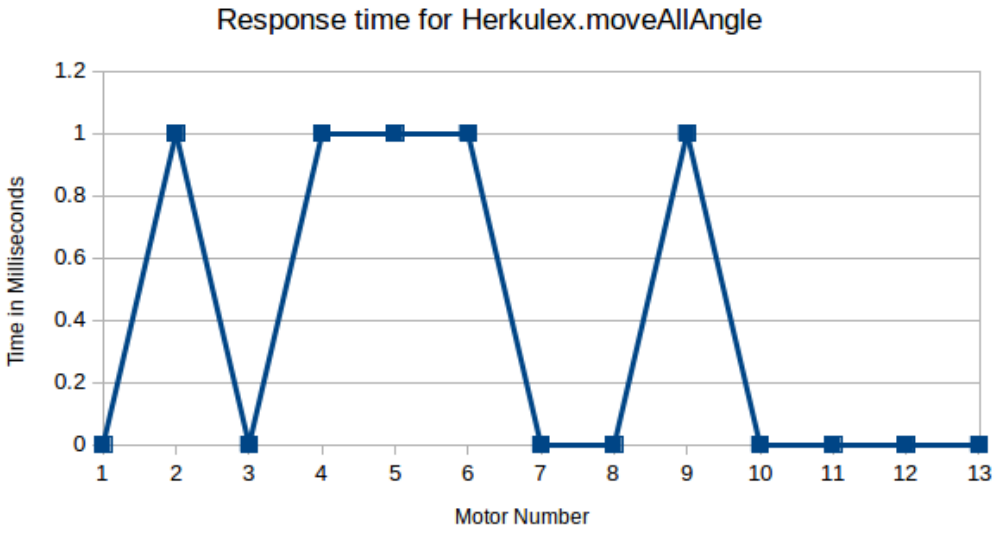

Photo 9: Un total de 5 millisecondes pour préparer les treize moteurs avec moveAllAngle.

Pour que l'actionnement soit fluide à l'utilisateur nous souhaitons avoir un actionnement de la marionnette dans un temps qui ne dépasse pas 40ms. Nous souhaitons avoir au moins un "Frame Rate" de 24fps, et donc chaque "Frame" ne doit pas dépasser 40ms. J'ai transmis les résultats du log dans un fichier Excel pour mieux interpréter les résultats qui sont les suivantes. La commande moveOne prends entre 5 et 6ms pour actionner un moteur, ayant treize moteurs cela donne 69ms et donc cette commande n'est pas utilisable dans notre cas. En revanche, la commande moveAllAngle prends entre 0 et 1ms pour **préparer** un moteur pour un actionnement synchrone avec les autres. Cela dit, moveAllAngle doit être suivi par la commande actionAll qui elle prend environs 10ms pour actionner tous les moteurs au même temps. Dans les pires des cas ou chaque moteur prends 1ms pour être préparer, la totale de temps consommé est de 23ms ce qui nous laisse assez de marge pour l'actionnement de la marionnette dans l'axe x avec un moteur pas-à-pas et de toujours respecter la contrainte de 24 fps minimum.

Simulation

Pour commencer j'ai effectué une simulation avec les douze moteurs maintenant bien identifié et placé dans les bons socles. Comme illustré en photo 5, il faut démarrer tout d'abord la LeapMotion et ensuite lancer la simulation Sofa. Il faut tout de même s'assurer que le port série indiqué dans la scène Sofa, fichier avec l'extension pyscn, est bien celui sur lequel l'Arduino est connecté (ACM0 ou ACM1 par exemple) avant de lancer la simulation.

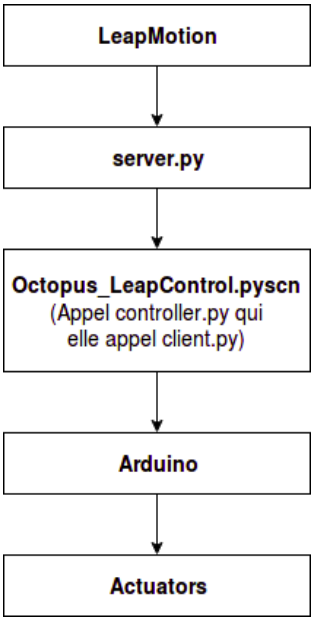

Photo 10: Schéma de fonctionnement

Ci-dessus est le schéma décrivant les liens entre les différents fichiers de code utilisés. Tout d'abord le fichier server.py récupère, de la LeapMotion, les informations représentant la position ainsi que la forme de la main. Ces information sont représenté sous forme de distances, dans les trois axes (x,y,z), entre l'origine qui est le centre du capteur LeapMotion et les points de référence de la main. Les différents points de la main reconnaissable par le capteur sont représenté sur la photo 5. La scène Sofa qui est lancée en parallèle fait appel au script controller.py qui lui fait appel à client.py pour récupérer les informations du server.py et les renvois à Sofa pour pouvoir faire les calculs nécessaires à la simulation. Une fois ces calculs sont faits, les valeurs à envoyer aux smart actuators sont récupéré par controller.py avant de les envoyer à l'Arduino. L'Arduino reçois un tableau de 14 valeurs par le port série qu'il traite ensuite pour déplacer chacun des 13 servomoteurs qui lui sont maintenant connectés. La dernière valeur est celle destinée au moteur pas-à-pas

contenant la distance à atteindre entre 4 et 22cm. Le démonstrateur est équipé d'un capteur ultrasons. La distance mesurée, entre le capteur et la plaque de moteurs, est utilisée pour activer ou non le moteur pas-à-pas. La vitesse du moteur est proportionnelle à la différence entre la position désirée et la position actuelle de la plaque.

Mouvement dans l'axe Z

La distance entre la LeapMotion et la main est exprimée en millimètres. Le code ci-dessous traite cette information comme suivant ; de 0 à 150mm nous considérons que l'utilisateur voudrais la position la plus basse de la marionnette. Ce qui lui permettra d'avoir assez d'espace entre la main et le capteur pour plier les doigts et contrôler la marionnette. À partir de 400mm la position la plus haute est choisie pour éviter que l'utilisateur lève trop la main et sort de du champ de captation. Dans le cas contraire, une hauteur intermédiaire est choisie pour notre marionnette. Un filtre exponentiel est utilisé pour éviter des mouvements brusques afin de garder la stabilité de la marionnette dans l'espace.

```
global interZ
global lastValueZ
if vectors[10][1] <= 150:
    interZ = 0
elif vectors[10][1] >= 400:
    interZ = 250
else:
    interZ = vectors[10][1]-150
outputVector[12] = lastValueZ * (1-alpha) + interZ * alpha
lastValueZ = outputVector[12]
```

La valeur de ouputVector[12] est ensuite ajoutée à la valeur des douze autres moteurs pour permettre à la marionnette d'être levé et de conserver la forme simulée, à une certaine limite bien sûr.

```
for i in range(0,12):
    if outputVector[i] < 0:
        outputVector[i] = 0
    outputVector[i] = 255/116*outputVector[i]
    #adding an offset to compensate for lifting the robot
    outputVector[i] += (outputVector[12])
    if outputVector[i] > 250:
        outputVector[i] = 250
```

Mouvement dans l'axe X

Pour tenir compte de la distance entre les bords du démonstrateur et la plaque de moteurs, nous avons utilisé un capteur ultrasons. Avec le capteur installé nous avons mesuré une valeur minimale de 4cm et une valeur maximale de 22cm.

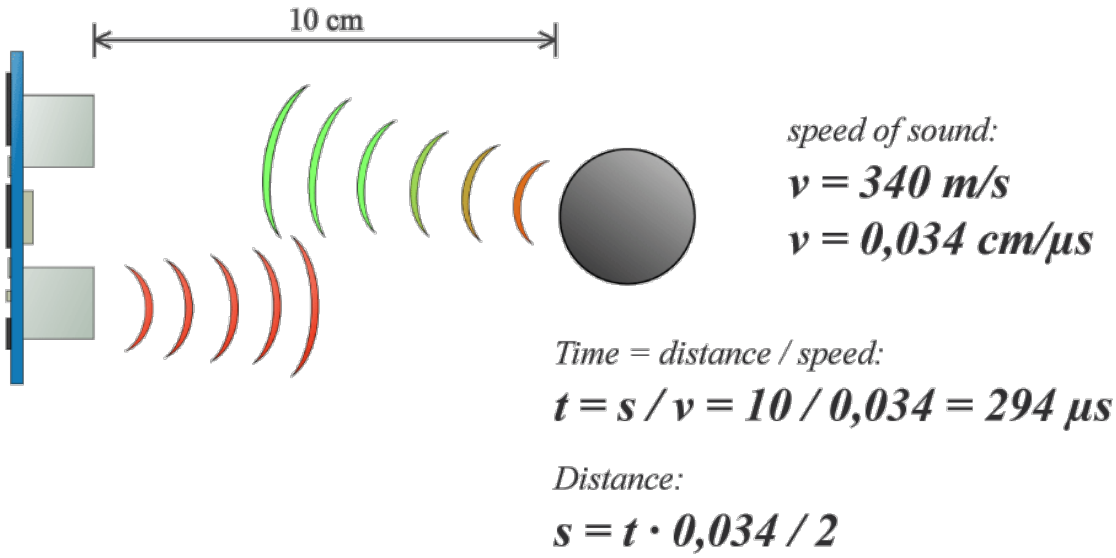

Photo 11: Fonctionnement du capteur ultrason

N'ayant que 18cm à manipuler dans l'axe X, nous avons donc limité les mesures de la LeapMotions entre -200mm et 200mm. Lorsque la mesure respecte cette condition, elle sera transformée en une valeur entre 4 et 22cm.

```
if vectors[10][0] > -200 and vectors[10][0] < 200:
    outputVector[13] = (vectors[10][0]+200)*18/400+4
elif vectors[10][0] < -200:
    outputVector[13] = 4
else:
    outputVector[13] = 22
```

Le moteur pas-à-pas responsable de réaliser le mouvement sur cet axe est connecté à un driver DRV8825 de la façon suivante.

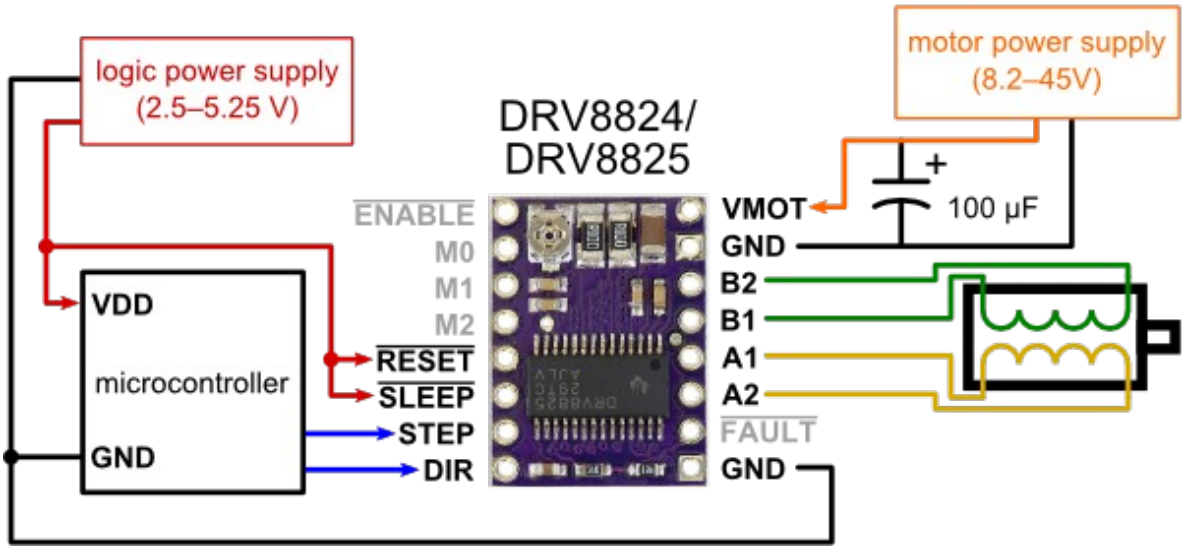

Photo 12: Branchement du moteur pas-à-pas.

Arduino

L'Arduino Uno reçois un tableau de 14 valeurs par le port série software. Ces valeurs sont ensuite traité pour déplacer chacun des 13 servomoteurs qui lui sont connectés.

```
for (int i = 1; i < 14; i++){
	Herkulex.moveAllAngle(i, map(positions[i], 0, 255, -150, 130 ), LED_GREEN);
}
Herkulex.actionAll(200);
```

La dernière valeur est celle destinée au moteur pas-à-pas. Elle contient une distance à atteindre entre 4 et 22cm. Le moteur est contrôlé avec un signal PWM avec une fréquence variable dans le but de varier sa vitesse.

```
OCR1A = periode;
TCCR1B = TCCR1B & 0b11111000 | 0x12;
```

La période nous permet de régler la fréquence du signal PWM et la deuxième ligne ci-dessus de choisir le mode PWM phase-correct. Ce mode nous permet d'avoir une PWM symétrique ainsi que régler la fréquence plus finement. Les registres contrôlent les paramètres de timer1 d'Arduino.

Photo 13: Signal PWM avec une période de 70µs.

La distance mesurée par le capteur ultrasons, est utilisée pour activer ou non le moteur pas-à-pas. La vitesse du moteur est proportionnelle à la différence entre la position désirée et la position actuelle de la plaque. Cela est rendu possible en modifiant la période, OCR1A, avec la fonction map. À partir d'une différence de 12cm la vitesse maximale à 20KHz est choisie.

```
if (distance-positions[14] > 1 && distance > 4 && positions[14]!=0){
    OCR1A = map(distance-positions[14],2,12,70,50);
    digitalWrite(PINdir, HIGH);
    analogWrite(PINstep,30);
  }
   else if (positions[14]-distance > 1 && distance < 22 && positions[14]!
=0){
    OCR1A = map(positions[14]-distance,2,12,70,50);
    digitalWrite(PINdir, LOW);
    analogWrite(PINstep,30);
  }
  else{
    analogWrite(PINstep,0);}
```

Le démonstrateur aujourd'hui

Vous trouverez ci-dessous quelques photos pour représenter l'état courante du démonstrateur.

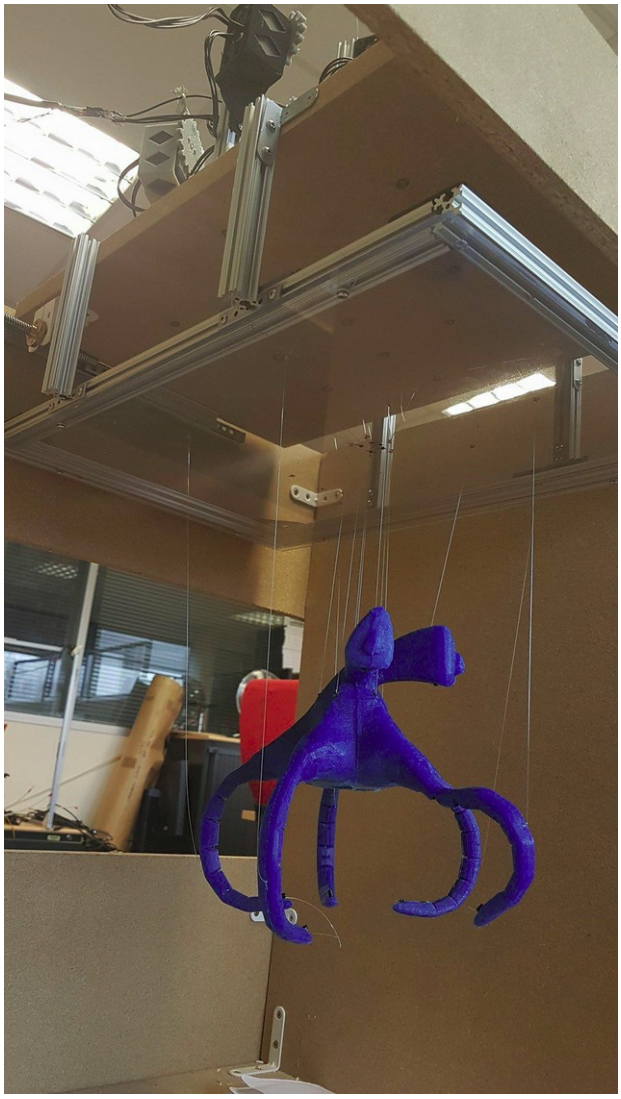

Photo 14: Marionnette et démonstrateur.

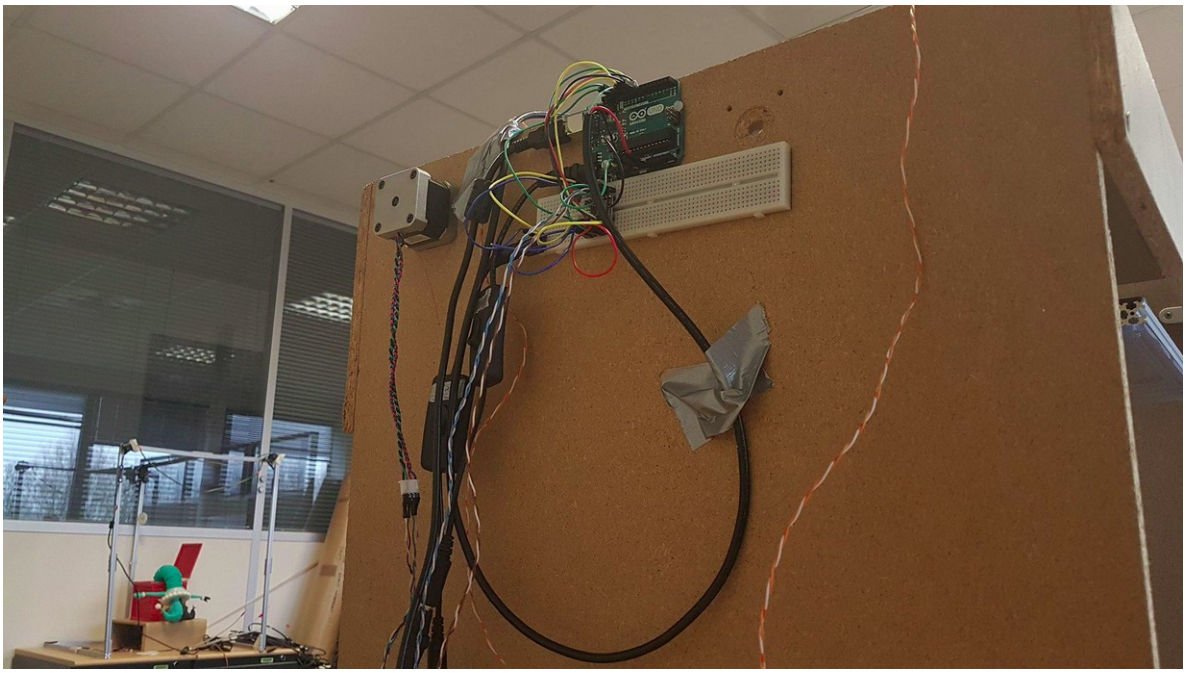

Photo 15: Arduino, driver et moteur pas-à-pas.

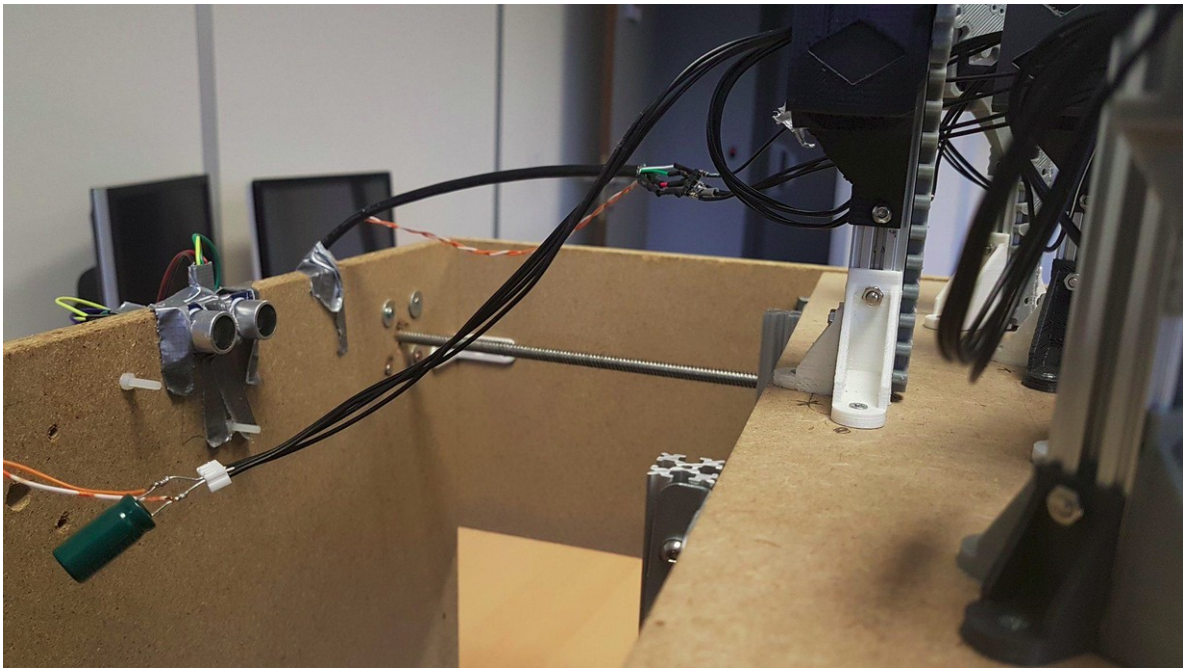

Photo 16: Capteur ultrason.

III La suite

- Prospectif d'ajouter un mouvement de la plaque dans l'axe Y.
- Passer par la simulation pour les mouvements dans les axes X et Z, dans le cas où l’environnent réel seras représenté dans la simulation.
- Meilleure gestion des câbles, avec une carte PCB.

IV Bibliographie

- http://robottini.altervista.org/dongbu-herkulex-arduino-library-2
- http://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
- https://developer.leapmotion.com/documentation/python/index.html
- https://www.robotshop.com/media/files/pdf2/_eng_herkulex_manual_20140218.pdf
- https://github.com/bmhagui/pfe_marionnette