Rapport de projet R&D

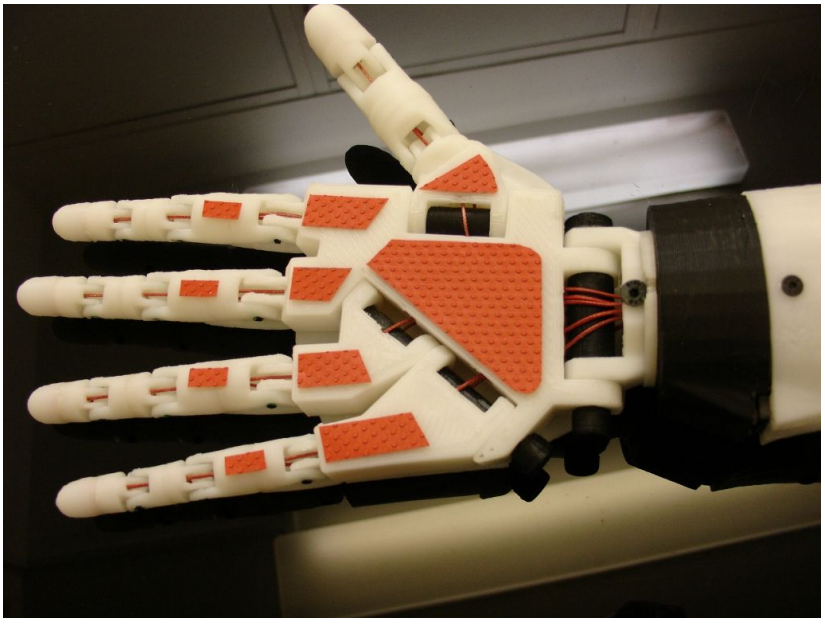
Projet Leap Motion et main InMoov

1. Présentation du problème

1.1. Inmoov

Le robot Inmoov est un robot humanoïde à taille réelle intégralement imprimable en 3D. Il a été créé en 2012 par Gaël Langevin sous forme d'un projet open source.

La main du robot est constituée de 5 doigts biomimétiques actionnés chacun en flexion et en extension par un servomoteur par le biais de câbles. Les doigts ont soit trois (pouce, index, majeur) soit quatre (annulaire, auriculaire) degrés de liberté en rotation et sont donc sous actionnés. Il est également possible d'ajouter un servomoteur pour permettre la rotation du poignet.



Main du robot InMoov (inmoov.fr)

Les servomoteurs que nous utilisons sont de marque Parallax et sont contrôlés par une Arduino Uno.

1.2. Leap Motion

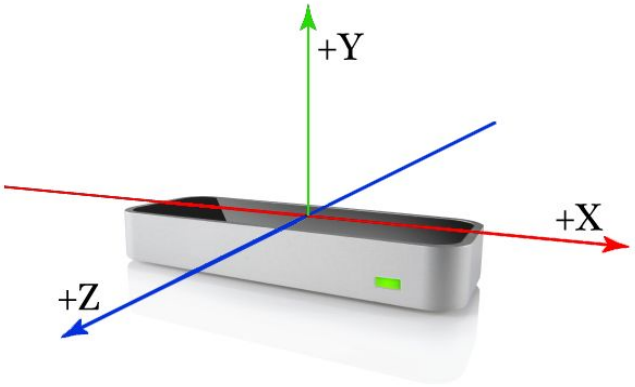
Le système Leap Motion est un dispositif de détection et de suivi des mains et des doigts. Le dispositif fonctionne dans une proximité intime avec une grande précision et rapporte des positions et des mouvements discrets.

Le contrôleur Leap Motion utilise des capteurs optiques et de la lumière infrarouge. Les capteurs sont dirigés le long de l'axe y - vers le haut lorsque le contrôleur est dans sa position de fonctionnement standard - et ont un champ de vision d'environ 150 degrés. La portée effective du Leap Motion Controller s'étend d'environ 25 à 600 millimètres au-dessus de l'appareil.

La vue du contrôleur Leap Motion des mains (inmoov.fr)

Le logiciel Leap Motion combine ses données de capteur avec un modèle interne de la main humaine pour aider à faire face à des conditions de suivi difficiles.

Le système Leap Motion utilise un système de coordonnées cartésiennes droitier. L'origine est centrée au sommet du Leap Motion Controller et les axes sont définis sur le schéma ci-dessous avec l'axe z ayant des valeurs positives croissant vers l'utilisateur.



Système de coordonnées du Leap Motion (inmoov.fr)

2. Etat de l'Art partiel

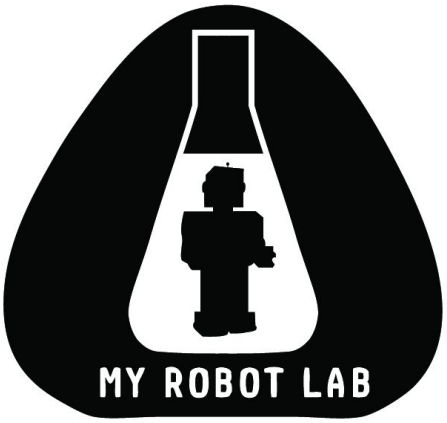
Une première recherche de solutions existantes nous a orienté vers le framework MyRobotLab qui est référencé par inmoov (http://inmoov.fr/how-to-start-myrobotlab/)

2.1. MyRobotLab

Le framework MyRobotLab regroupe de nombreux services pour la robotique et le contrôle de machine. En effet, il inclut des services tels que des composants de vision avec OpenCV, des éléments de reconnaissance de parole, de contrôle moteur et servomoteur. La partie qui nous intéresse particulièrement est la gestion des communications avec les microcontrôleurs.

MyRobotLab inclut de plus un service pour le robot inMoov qui permet de contrôler le robot complet grâce à une interface graphique. Tout d'abord nous avons considéré la combinaison de ce service avec la gestion des communications avec l'Arduino (protocole MRL) comme une solution très probante. Cependant, après avoir voulu comprendre tous les processus impliqués dans ces services, nous avons remarqué l'utilisation de protocoles complexes qui allaient nécessiter un lourd travail de compréhension et d'adaptation pour effectuer nos propres tests et récupérer les données que nous souhaitions.

La découverte de ce framework nous aura en tout cas dirigé vers la recherche d'un protocole de communication entre Python et l'Arduino. Après une courte recherche, nous avons pu trouver des protocoles plus clairs ayant une syntaxe proche des syntaxes utilisées dans les codes Arduino.



Logo My Robot Lab (myrobotlab.org)

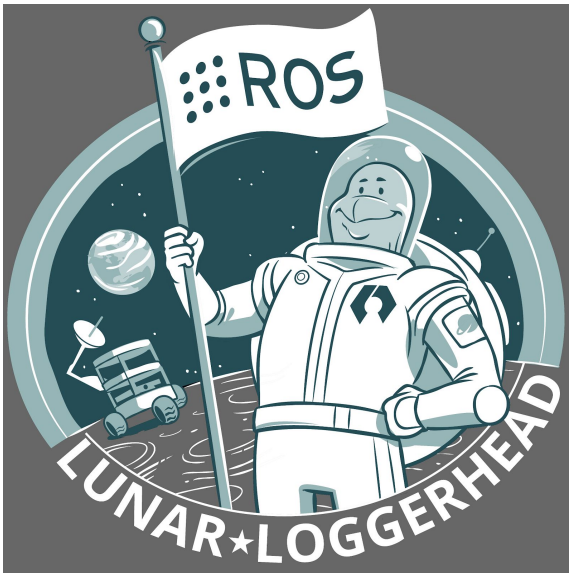
2.2. Utilisation de ROS

Une autre possibilité que nous n'avions pas découverte lors de notre première recherche est l'utilisation du Robot Operating System (ROS). Il existe en effet un package ROS permettant de l'interfacer avec le Leap Motion.

ROS est un framework flexible donnant accès à de nombreux outils dans le domaine de la robotique. Il permet notamment d'interagir entre plusieur système (Ordinateur, carte microcontrôleur…) dans plusieurs langages (python, C++ notamment).

Dans le cadre de notre projet, l'utilisation de python pour récupérer les données du leap motion n'est pas impacté. Cependant, ce framework nous aurait permis de coder directement les fonctions devant actionner les servomoteurs sur l'arduino avec la librairie rosserial_arduino. En effet, le point fort de ROS est l'utilisation de Publisher et Subscriber qui masque l'utilisation d'un protocole de transmission des messages. Lorsque l'on veut envoyer un message, il suffit de stocker notre donnée dans une variable ROS et de l'envoyer à l'aide d'un Publisher sur un topic (sujet de communication). Sur les systèmes où l'on souhaite recevoir des données, il suffit de souscrire au topic afin de recevoir les messages qui transitent. Des fonctions peuvent être appelées à la réception des données pour les traiter de la façon que l'on souhaite.

Au delà des protocoles de transmission, ROS propose de nombreux affichages graphiques tels que rviz ou gazebo sur lesquels il serait possible de créer une simulation afin de se soustraire des contraintes mécaniques de la construction de la main en 3D.



Logo de la version actuelle de ROS (ros.org)

3. Solution proposée

3.1. Détection de la main avec le SDK Leap Motion et Python

3.1.1. Etat Actuel

3.1.1.1 Installation du SDK et des bibliothèques nécessaires

Un certain nombre de bibliothèques sont nécessaires pour faire fonctionner le Leap avec Python. Ces librairies sont disponibles en ligne ainsi que dans l'archive de notre projet GitHub (voir bibliographie).

3.1.1.2 Connection au contrôleur et récupération d'une image

On va tirer parti des objets Controller et Frame fournis par le SDK Python du Leap motion. On ne se sert que de méthodes basiques ici.

On peut ainsi se connecter au leap motion avec la commande :

```
controller = Leap.Controller()
```

On va ensuite pouvoir accéder à l'image vue par le Leap Motion avec la fonction :

```
frame = controller.frame()
```

3.1.1.3 Récupération des mains et des doigts ainsi que leurs positions

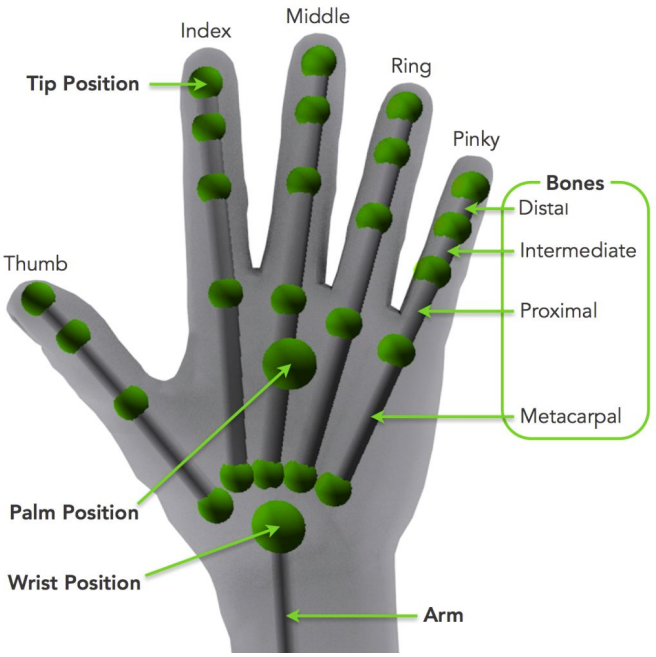
Les liens vers les descriptions complètes des classes Hand et Finger utilisées dans ce paragraphe sont disponibles dans la bibliographie.

La fonction **frame.hands** donne un vecteur contenant toutes les mains détectées dans l'image par le Leap. Il ne nous en faut qu'une seule et puisqu'on travaille sur une main droite, on décide arbitrairement de ne conserver que la main la plus à droite de l'image avec la fonction **hand = frame.hands.rightmost.**

Avec cet objet et la fonction **hand.palm_position**, on peut récupérer la position du repère attaché à la paume de la main. C'est un vecteur de trois composantes contenant la distance du centre du repère associé à la main par rapport à celui lié au Leap selon les trois axes.

La fonction **hand.fingers** donne un vecteur contenant les cinq doigts correspondants à la main en question. Il est alors possible de récupérer la position de la dernière phalange de chaque doigt avec **fingers[indice].joint_position(3)** où indice est le numéro associé au doigt considéré (de 0 pour le pouce à 4 pour l'auriculaire), retourne un vecteur à trois

composantes contenant la distance du centre du repère associé à la dernière phalange du doigt considéré au repère lié au Leap, selon les trois axes.



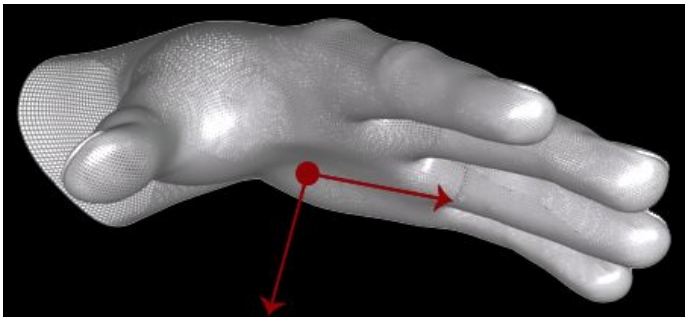
Carte des points que le Leap Motion peut détecter et tracker (blog.leapmotion.com)

On n'a ensuite plus qu'à calculer la distance en norme entre le centre du repère lié à la main et le centre du repère associé à la dernière phalange du doigt.

On va ensuite pouvoir transformer cette distance en une commande pour les servomoteurs.

3.1.1.4. Angle du poignet

Pour améliorer notre système nous avons ajouté la commande de l'angle du poignet en plus des 5 doigts. Pour ce faire on utilise la fonction **hand.palm_normal** qui donne un vecteur normal à la paume de la main. En utilisant la fonction **hand.palm_normal.roll** on obtient directement l'angle que l'on veut commander.



Le repère associé à la main (inmoov.fr)

3.1.2. Améliorations possibles

Les limites auxquelles on se heurte à ce stade sont principalement dues à la précision du capteur. En particulier, lorsque la main est repliée, le capteur est peu précis sur la position des doigts.

Illustration du problème de détection en configuration “main fermée” (blog.leapmotion.com)

On pourrait imaginer différentes solutions à ce problème.

La plus évidente serait de changer de capteur pour un capteur plus performant.

Une seconde idée pourrait être d’utiliser une commande redondante, c’est-à-dire détecter plus d’informations que nécessaire pour commander chaque doigt. Chacun des doigts n’a qu’un seul actionneur et donc besoin que d’une seule entrée de commande, la distance entre la paume et la dernière phalange devrait donc suffire. Néanmoins on pourrait par exemple aussi récupérer les angles entre les différentes phalanges et s’en servir pour déterminer la distance entre la paume et la phalange (en utilisant le modèle géométrique direct d’un robot 3R plan). On aurait alors deux valeurs de cette distance, une mesurée et une calculée, qu’on pourrait alors moyenner pour se rapprocher de la valeur réelle. Cependant il est très probable que dans ce cas ci on rencontre également rapidement les limites du capteur.

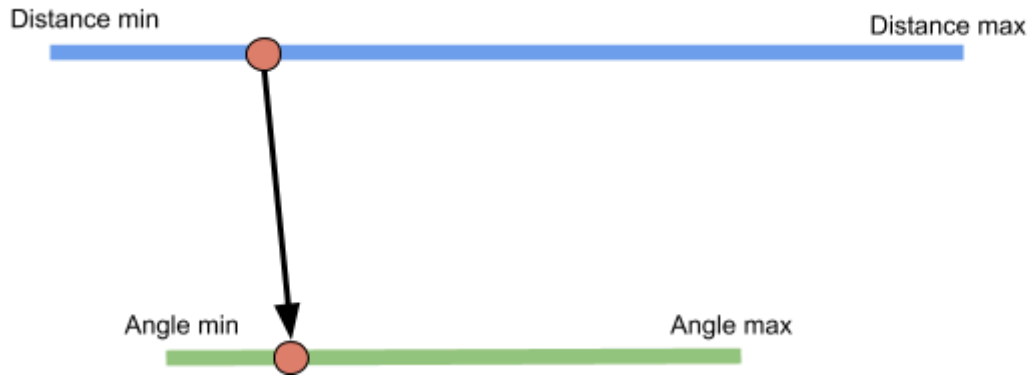
Une dernière option pourrait être d’anticiper les positions à venir pour chaque doigt. En utilisant un historique des positions de chaque doigt on peut aisément calculer leur vitesse et leur accélération et essayer d’en déduire la position suivante du doigt. On pourrait comparer cette position estimée a la position mesurée pour plus de précision. Une approche plus haut niveau proche de celle ci pourrait être de reconnaître des mouvements ou des pattern classiques.

3.2. Elaboration de la commande

3.2.1. Etat actuel

Chaque doigt de la main est sous actionné, en effet, un seul servomoteur contrôle la flexion et l'extension. On n'a donc besoin que d'une seule entrée par doigt. On choisit de se servir de la distance entre la dernière phalange et le centre de la main que nous avons obtenue au paragraphe précédent.

Ainsi, il nous faut transformer une distance en un angle de commande pour le servomoteur. Dans un premier temps nous avons choisi de stocker les distances minimale et maximale détectées pour chaque doigt, et ensuite d'utiliser une échelle linéaire pour lier cette plage de valeurs à la plage de valeurs prises par l'angle de commande du servomoteur. On utilise une simple transformation affine.



Ce choix a l'avantage d'être robuste, en effet, il y aura toujours une valeur minimale et maximale de la distance et on pourra toujours en déduire un angle de commande. Cette méthode est également rapide et peu coûteuse en ressources puisqu'il s'agit d'une très simple fonction mathématique. Néanmoins, cette méthode est très sensible aux perturbations. En effet, si le système détecte une valeur anormalement grande ou anormalement petite, il va la stocker dans le maximum ou le minimum de la plage de valeurs d'entrée, et donc dilater l'intervalle. On perd alors grandement en précision.

Par la suite nous avons remplacé cette mesure simple par une moyenne sur 5 mesures. Cela permet d'une part de lisser les mesures et d'obtenir une valeur plus proche de la réalité, et d'autre part cela permet de compenser les mesure aberrantes. En effet, si une valeur anormalement haute est détectée à cause d'un bug du capteur, son effet sera atténué par les autres valeurs prises en compte dans la moyenne, cette mesure aberrante ne viendra pas top dilater l'intervalle.
Le nombre de 5 mesures a été choisi car il permet d'avoir une assez bonne précision sans trop amputer les performances du capteur et la rapidité de la mesure.

3.2.2. Améliorations possibles

Une piste d'amélioration serait de remplacer la simple transformation affine entre les deux intervalles par un correcteur PID. Les gains du correcteurs seraient alors à déterminer en fonction de contraintes de stabilité et de rapidité.

3.3. Transmission des informations à l'Arduino avec pyFirmata

Le développement de la solution a nécessité la recherche d'une interface de communication entre les scripts pythons permettant de récupérer les informations du Leap Motion et l'Arduino commandant les servomoteurs.

Plusieurs interfaces sont proposées sur le site Arduino (voir bibliographie). Parmi ces interfaces, nous avons choisi d'utiliser la bibliothèque pyFirmata qui utilise le protocole Firmata pour communiquer avec l'Arduino. La choix de cette bibliothèque vient de sa simplicité d'utilisation et de la disponibilité de différents exemples.

L'utilisation se décompose en deux parties. Premièrement, il faut charger dans l'Arduino le script d'exemple StandardFirmata.ino (voir projet GitHub). Ensuite, nous pouvons créer le script python et initialiser la connexion avec l'Arduino :

```
import pyfirmata

# Adjust that the port match your system, see samples below:
# On Linux: /dev/tty.usbserial-A6008rIF, /dev/ttyACM0,
# On Windows: \\.\COM1, \\.\COM2
PORT = '/dev/ttyACM0'

# Creates a new board
board = pyfirmata.Arduino(PORT)
```

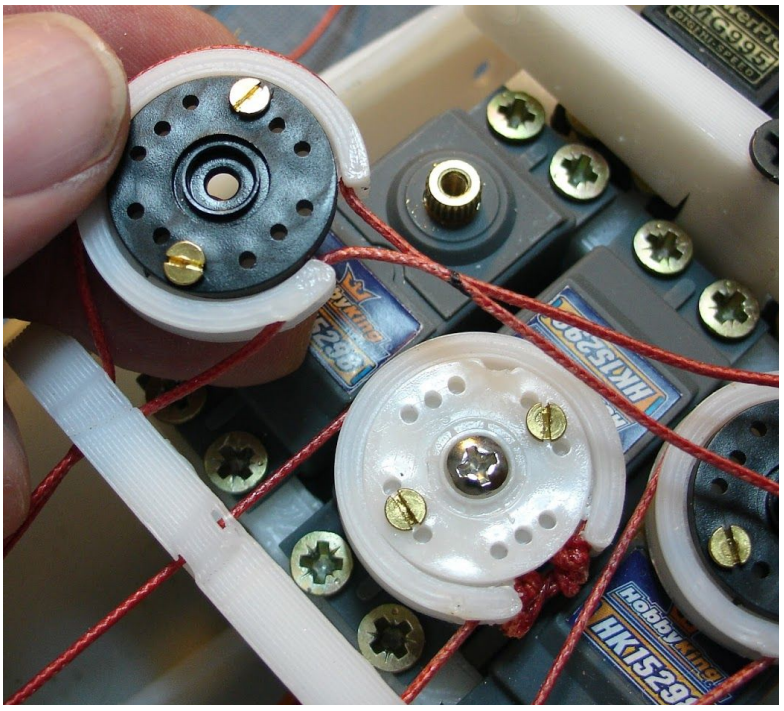
Une fois l'Arduino reliée, les commandes pour obtenir ou envoyer des informations sont simples et plusieurs exemples sont disponibles (voir bibliographie).

3. Problématiques annexes

3.1. Tension dans les tendons

Un des problèmes principaux dans la plupart des mains bioniques actionnées par câbles est la perte de tension dans les câbles. Les câbles peuvent notamment se détendre avec le temps ou à cause d'une utilisation répétée.

De plus, la notice de montage de la main Inmoov propose de tendre le câble à l'aide d'un noeud.

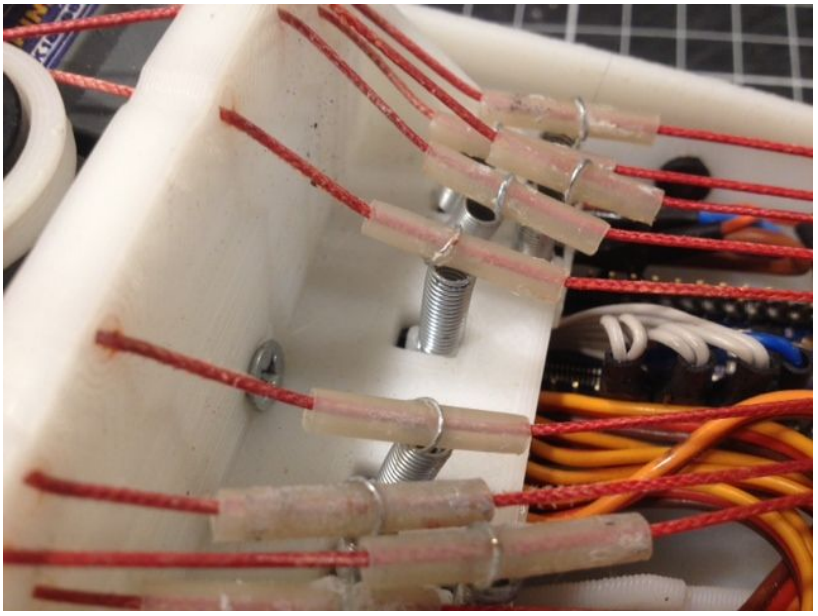


Mise en place des liaisons câbles-servomoteurs (inmoov.fr)

Avec ce système il est en fait difficile de maîtriser la tension dans le fil. En effet, il est en pratique très difficile de nouer le fil suffisamment proche de l'encoche pour avoir une tension satisfaisante.

3.1.1 Solution proposée par Inmoov

Pour répondre à ce problème, Inmoov propose d'imprimer une pièce supplémentaire à laquelle on peut attacher des ressorts. Le ressort relie ainsi le tendon à la structure du bras et permet de maintenir la tension en faisant varier l'élongation du ressort.



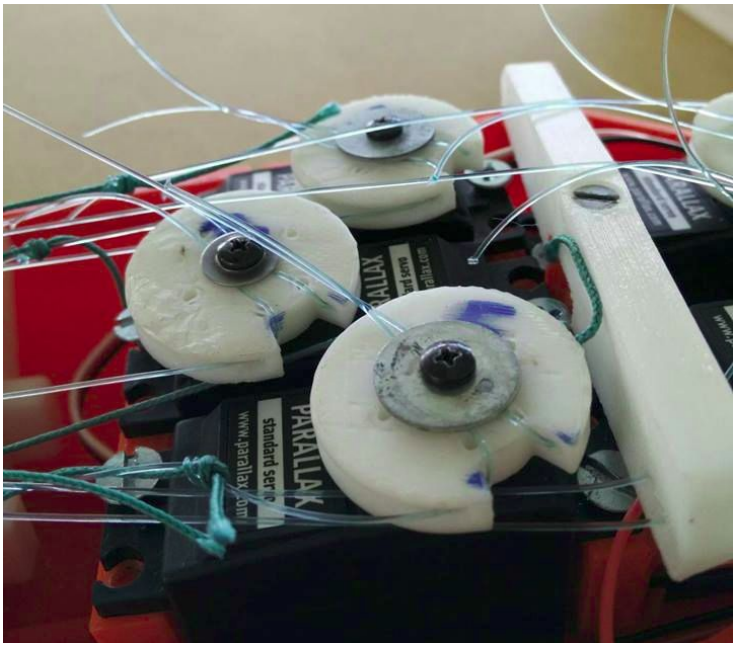
Principe de maintien des câbles en tension (inmoov.fr)

Si cette solution est bonne en théorie, elle est en fait difficile à mettre en place en pratique. En effet, l'espace disponible pour manipuler les tendons et les ressorts est trop petit, et il faudrait en principe avoir un ressort tendu au maximum au moment où on noue le câble au servomoteur ce qui est difficilement réalisable.

Nous nous sommes donc orientés vers une autre solution.

3.1.2 Notre solution

La solution que nous avons retenue est d'utiliser une vis et une rondelle afin de maintenir le tendon en position. Le système est très simple à mettre en place avec un tournevis. Le réglage est facile et rapide à faire.



Système de maintien en tension des tendons

Ainsi dans cette solution, plutôt que de chercher à corriger la tension en toutes circonstances, on accepte que le fil se détende et on recommence complètement la tension du câble lorsque c'est nécessaire.

3.2. Shield Arduino et Alimentation

D'après la fiche technique de l'Arduino Uno, chaque sortie digitale peut fournir un courant de 20mA en utilisant la tension interne de 5V d'après la spécification technique donnée par le site arduino.cc.

Nous utilisons au total 6 servomoteurs, qui peuvent consommer jusqu'à 190mA chacun. Soit un courant total de 1,14A au maximum, ce qui dépasse de loin les capacités de l'Arduino.

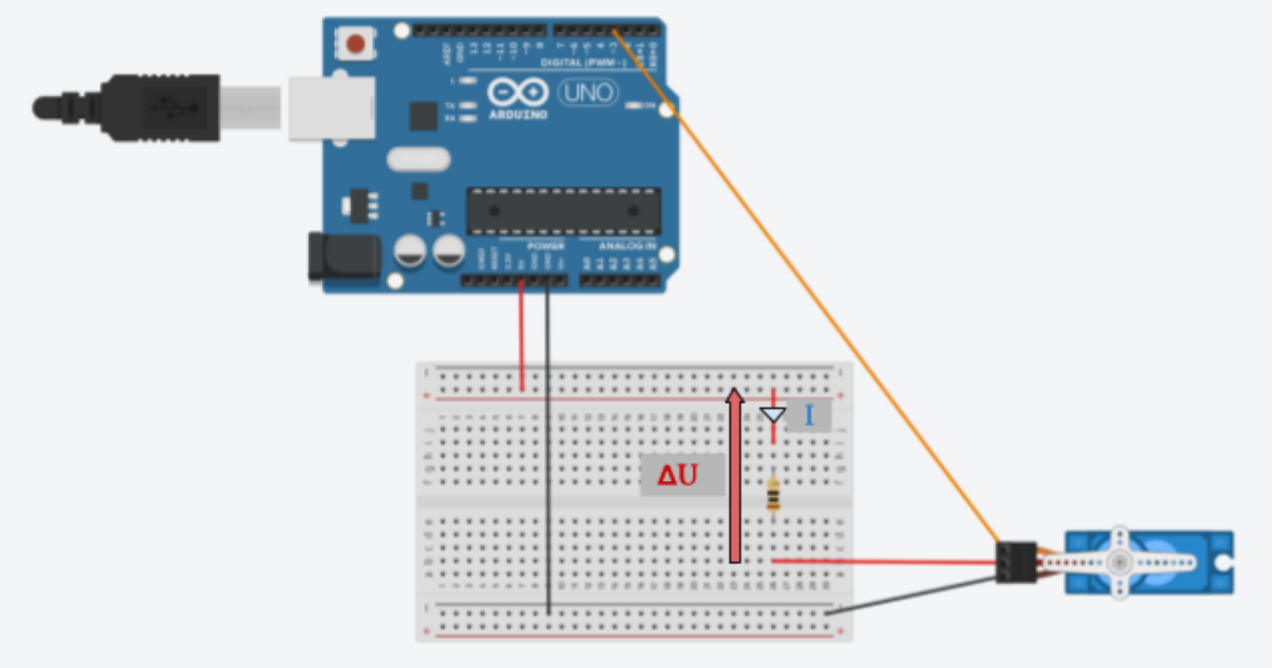
Notre première idée a été de brancher l'Arduino directement sur secteur via un transformateur pour vérifier que la limite ne venait pas du port USB du PC. Cela n'a pas été suffisant et nous avons donc choisi d'utiliser un générateur 6V pour alimenter directement les servomoteurs sans passer par la carte arduino et ainsi désolidariser le circuit de puissance et le circuit d'information. Nous avons récupéré notre tension de 6V sur une alimentation ATX pour PC.

Une amélioration intéressante à apporter à ce système serait de concevoir un shield arduino inspiré des shields moteur existants pour permettre de connecter et déconnecter les différents servomoteurs beaucoup plus simplement et éviter les très nombreux fils qui s'emmêlent. Ce shield pourrait intégrer le circuit de mesure développé dans la partie suivante.

Shield moteur officiel Arduino (store.arduino.cc)

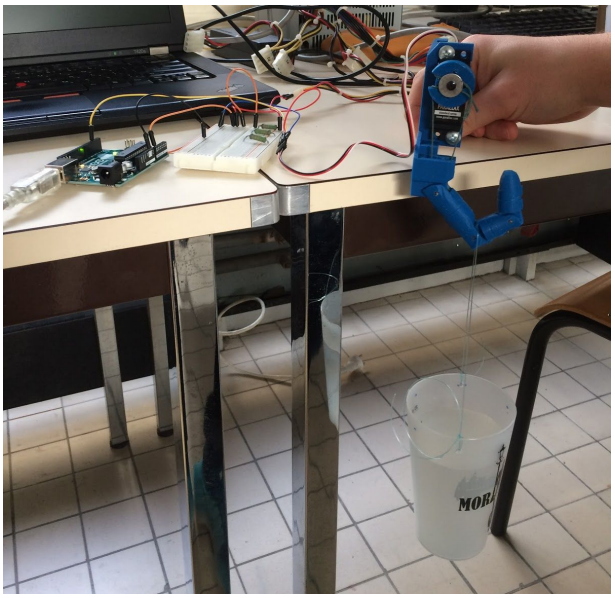
3.3. Détection des efforts dans la main

Afin de mesurer les efforts fournis par chaque doigt et ainsi pouvoir déterminer la force de préhension de la main, nous avons pensé à mesurer le courant utilisé par chaque servomoteur. Pour obtenir ce courant, nous plaçons une résistance faible (~2,3Ω) en série entre le générateur de tension 5V et le servomoteur. En mesurant la chute de tension entre le générateur on obtient le courant utilisé par le servomoteur par $I = \frac{\Delta U}{R}$.



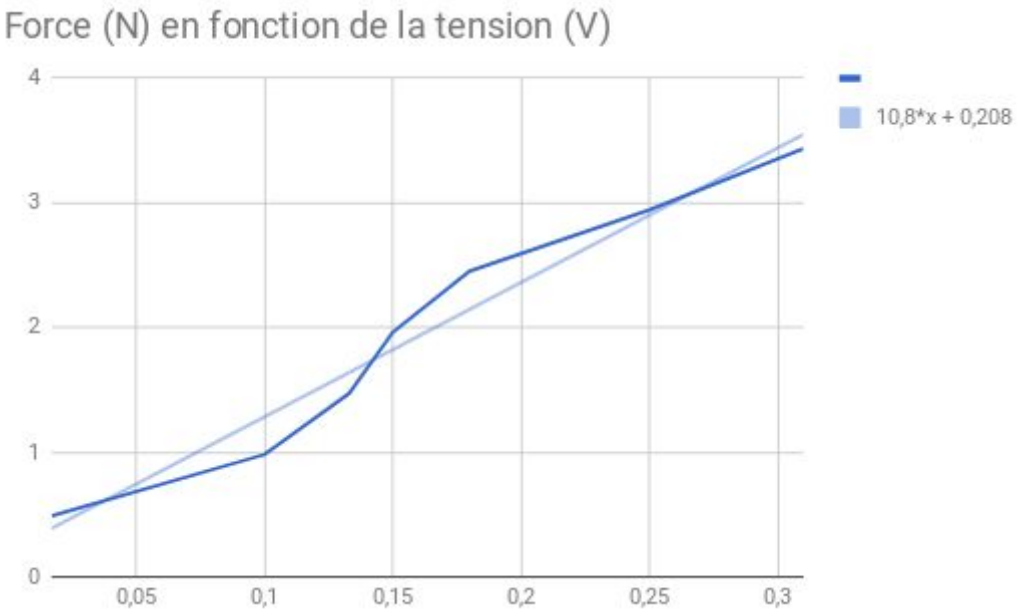
Montage de détection d'efforts par une résistance

Nous avons ensuite dû concevoir un dispositif permettant d'étalonner notre système de mesure. N'ayant pas à notre disposition de masse calibrée, nous avons choisi d'utiliser des gobelets remplis d'un volume prédéfini d'eau.



Montage de mesure des efforts engendrés par l'eau dans le gobelet

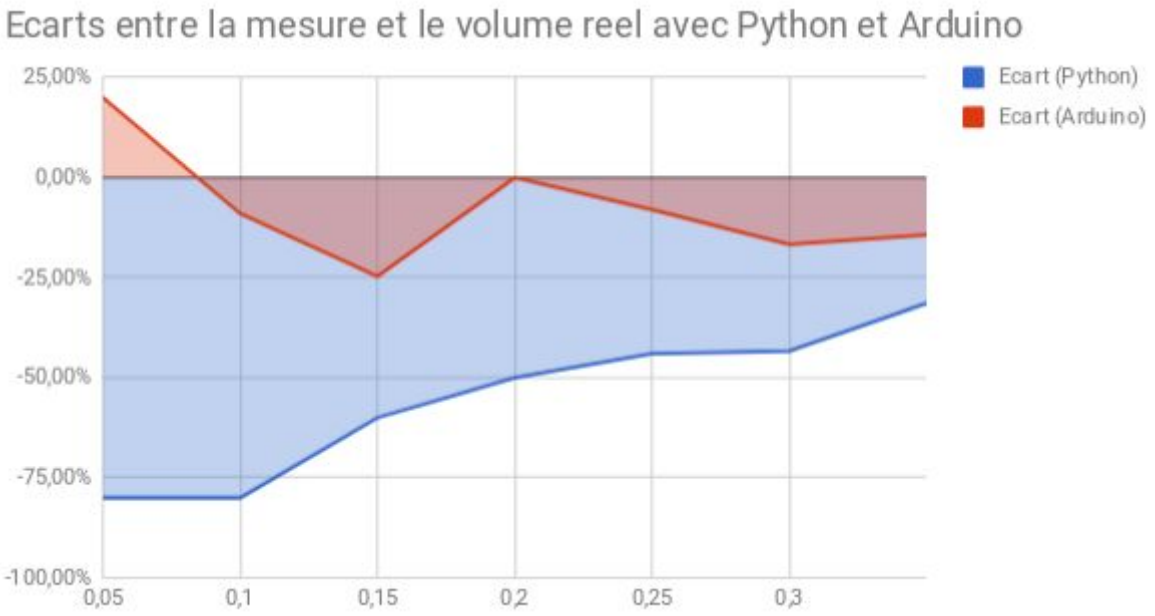
Nous avons ensuite pris autant de mesures que nous le permettaient les graduations des gobelets et tracé le graphique de la force appliquée au doigt (reliée à la masse d'eau par un facteur g) en fonction de la tension mesurée (liée au courant moteur par un facteur R).



On se rend compte qu'une relation affine entre la force et la tension n'est pas absurde, on trace donc la courbe de tendance (dont le R^2 est de 0,96) et on obtient donc l'équation $F \approx 10,8 \times V + 0,208$.

Il y a plusieurs limites évidentes à notre méthode et notre expérience. Le nombre de points utilisé pour tracer les courbes est faible et la mesure de volume en utilisant les graduations du verre est peu précise. Ces limites sont dues au matériel disponible et n'invalident pas la méthode, si celle ci s'avère concluante il est possible de reproduire l'expérience en prenant beaucoup plus de mesures avec des masses précises. Par ailleurs les mesures ont été fait dans un montage ou un seul servomoteur était branché, il est possible qu'elles varient dans un montage avec six servomoteurs branches sur la même alimentation. Nous avons également négligé la chute de tension due a la résistance placée en série du servomoteur, ainsi lorsque le servomoteur fournit plus d'efforts, il utilise une plus grande intensité, ce qui augmente la tension aux bornes de la résistance et diminue celle aux bornes du servomoteur, le servomoteur a alors besoin d'une plus grande intensité pour maintenir la même puissance. On néglige également de nombreux facteurs comme la température de la pièce ou la résistance des fils par exemple.

Afin de tenter de valider le modèle on reproduit l'expérience à l'envers et on cherche à deviner la masse de liquide contenue dans le verre en ne mesurant que la tension aux bornes de la résistance.
Dans un premier temps nous avons effectué le traitement sous python en utilisant le pyFirmata, et sur l'intervalle de masses utilise (0,05 kg a 0,35 kg) on obtient un écart maximal entre les valeurs mesurées et les valeurs réelles de l'ordre de 80%.
En effectuant ce même traitement directement sous Arduino, cet écart maximal est réduit à 25%.



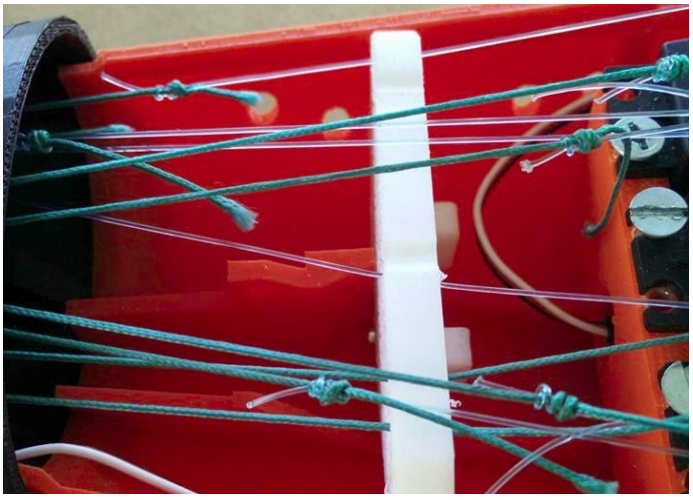
L'équation appliquée à la mesure de tension pour obtenir l'estimation du volume étant la même sous Arduino ou sous Python, l'écart ne peut venir que de la mesure elle même. Notre hypothèse est que la mesure perd en précision (un arrondi est probablement fait) en passant d'Arduino à Python.

On peut donc ici conclure que la mesure par résistance n'est pas la meilleure méthode de mesure de force dans notre cas. En effet, si on utilise une résistance trop élevée (>10 Ohm, ce qui est déjà extrêmement faible), cela entraîne une chute de la tension aux bornes du servomoteur qui l'empêche de fonctionner. Dans le cas ou l'on prend une résistance suffisamment faible pour que cette chute de tension soit acceptable, la précision est mauvaise, et il y a quand même une certaine chute de tension qui empêche le servomoteur d'atteindre son couple maximum.

Il semblerait qu'un montage utilisant la déformation d'une mousse conductive soit plus performant.

3.4 Problèmes de montage et de friction

Pour motoriser le poignet nous avons dû démonter puis remonter l'avant bras. Cependant il n'était pas possible de démonter la main. Nous avons donc dû sectionner et raccorder les tendons. Cependant, certains des noeuds de raccord ont tendance à se coincer dans les guides prévus dans le poignet et les servomoteurs ne sont pas assez puissants pour les décoincer ce qui rend inutilisable les doigts actionnés par ces tendons. En plus de cela, il y a de fortes contraintes de friction dans les articulations du pouce notamment, et de même le servomoteur n'est pas assez puissant pour l'actionner.



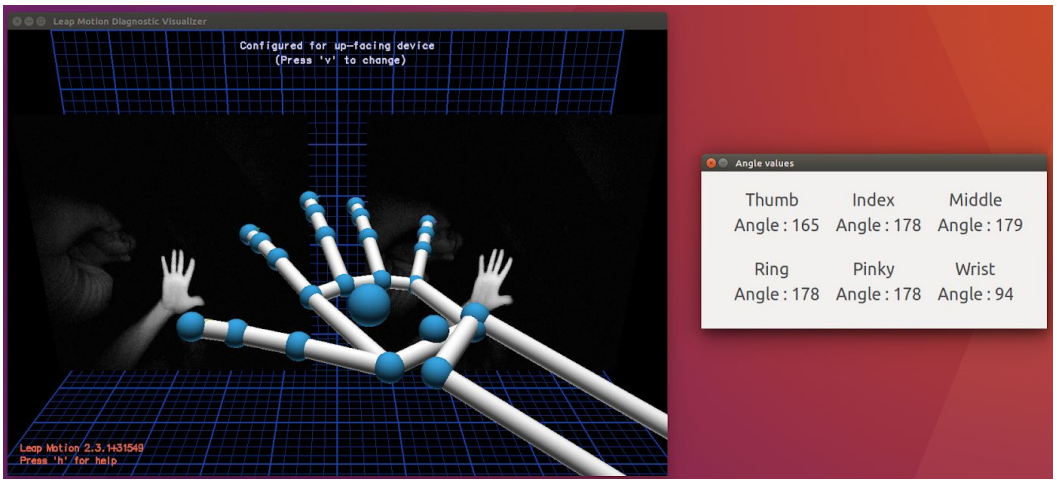
Noeuds de raccordement des tendons

La seule solution à ce problème était de réimprimer la totalité de la main, de l'avant bras, et des pièces structurelles pour les servomoteurs afin de pouvoir refaire complètement les tendons. En effet, les pièces qui constituent la main et l'avant bras sur lequel nous travaillons actuellement sont issues de différentes versions du modèle inMoov, et engendrent donc déjà des problèmes de compatibilité. Afin de contourner les problèmes mécaniques, nous avons créé un affichage des commandes moteurs.

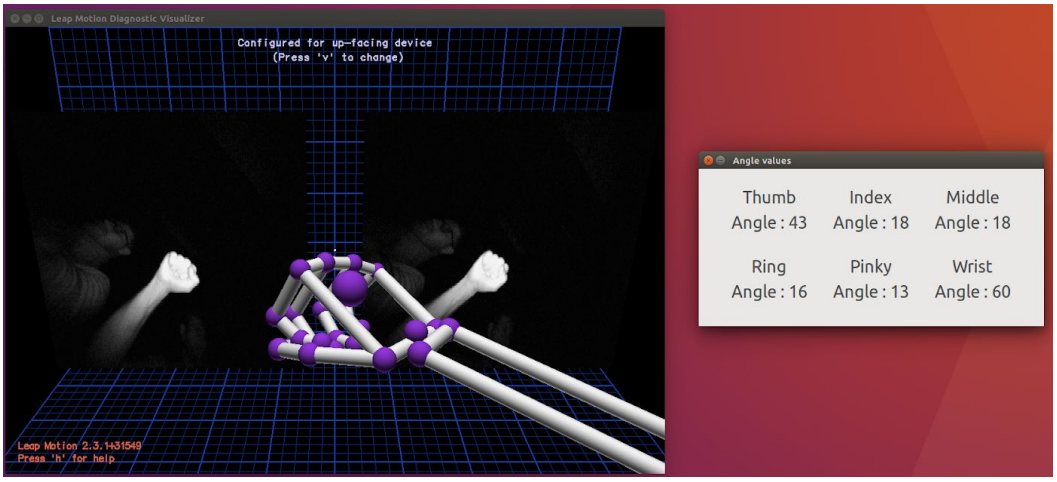
3.5 Visualisation des commandes sur une interface graphique

Avec les différents problèmes mécaniques que nous avons pu avoir, nous avons choisi de réaliser une interface graphique afin de visualiser simplement les commandes transmises et de s'assurer du bon fonctionnement de notre modèle.

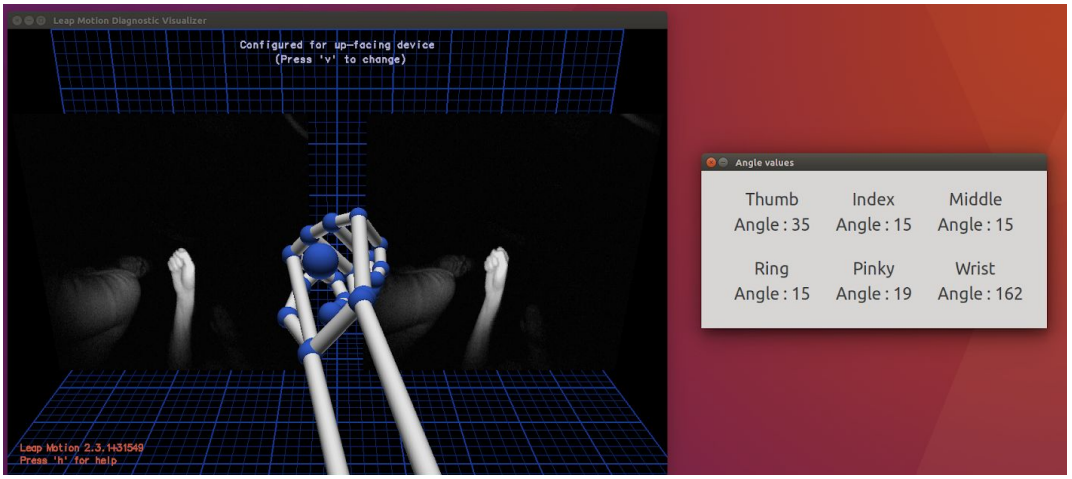
Dans ce cadre, nous avons choisi d'utiliser la librairie wxPython afin de garder le même langage de programmation et la librairie multiprocessing afin de partager les données en deux processus. Voici le rendu final pour plusieurs positions de la main :



Visualisation des commandes en main ouverte



Visualisation des commandes en main fermée



Visualisation des commandes en main fermée et poignet orienté

Nous pouvons ainsi apprécier la qualité de la commande en fonction des différentes positions. Nous remarquons tout d'abord la bonne performance du système avec des valeurs proches de zéro pour les doigts en main fermée et des valeurs proches de 180 en main ouverte. Le manque de précision du capteur ainsi que certaines valeurs aberrantes peuvent être à l'origine des résidus de commande, angle de commande à 15° au lieu de 0°, notamment visible en main fermé.

Conclusion

Ce projet de Recherche et développement a donc abouti sur plusieurs problématiques différentes. En partant de la problématique de base qui était la commande par mimétisme d'une main robotisée, nous avons traité des sujets allant de la mécanique pour l'adaptation de la main à la visualisation en temps réelle des commande, en passant par la recherche des efforts appliqués par la main et des problématiques énergétiques.

Après nos travaux, nous avons pu résoudre notre problématique de base et proposer plusieurs pistes de recherche afin de faire évoluer notre solution. Pour de plus amples recherches, il faut tout de même souligner la nécessité d'imprimer une nouvelle main. Ceci évitera les problèmes mécaniques de blocage et de détente pour s'assurer de la bonne réponse de la main aux sollicitations tout en intégrant les mesures d'efforts par mousse déformable.

Ce projet nous a permi de développer nos compétences dans les domaines de la mécanique, de l'électronique et de l'informatique et de découvrir les contraintes liées aux conceptions mécaniques.

Bibliographie

- Projet GitHub : https://github.com/ybri/inmoov/tree/develop
- Site internet InMoov : http://inmoov.fr/
- Site internet Arduino : https://www.arduino.cc/
- Interface Pyhton-Arduino : https://playground.arduino.cc/Interfacing/Python
- Site internet Leap Motion Developer : https://developer.leapmotion.com/
    - Pages de documentation Python pour la classe Hand : https://developer.leapmotion.com/documentation/python/devguide/Leap_Hand.html et https://developer.leapmotion.com/documentation/python/api/Leap.Hand.html
    - Page de documentation Python pour la classe Finger : https://developer.leapmotion.com/documentation/python/devguide/Leap_Pointables.html et https://developer.leapmotion.com/documentation/python/api/Leap.Finger.html
    - Bibliothèques pour le développement : https://developer.leapmotion.com/documentation/python/devguide/Project_Setup.html
- Documentation et sources pyFirmata : https://github.com/tino/pyFirmata
- Exemples pyFirmata : https://bitbucket.org/fab/pyfirmata/src/96116e877527?at=default
- Documentation servomoteur parallax Standard : https://www.parallax.com/sites/default/files/downloads/900-00005-Standard-Servo-Product-Documentation-v2.2.pdf
- Mesure de force par deformation de mousse conductive : https://www.youtube.com/watch?v=SFLLjkhvsYs
- ROS : http://wiki.ros.org/leap_motion http://wiki.ros.org/rosserial_arduino/Tutorials/Servo%20Controller