La Pinokio Lampe

Dany Samy, Hugo Tram,
Matthieu Nogatchewsky, Simon Bauer

Table des matières

Introduction

C'est une lampe à laquelle on impute des expressions que l'humain peut interprêter : l'expression d'un oui de la tête ou d'un non, de la peur, de la joie... En effet son aspect quelque peu humanoïde, dû à ces quatres leviers en bois ou encore à son petit oeil en verre, renforce l'impression que son public humain a d'avoir en face de lui un petit personnage qui a des impressions, des ressentis, des émotions : cette lampe pourrait ainsi être utilisée à des fins médicales dans le traitement psychologique d'un patient. Pour faire notre projet, il nous a été fourni la lampe et un raspberry Pi 3 qui seront connectés par câble. Notre objectif est de faire en sorte que la lampe suive un visage face à elle et que les algorithmes de calculs soient effectués sur un PC à distance. Nous devrions ainsi être en mesure de placer dans une salle fermée, un sujet en face de la lampe, seuls.

Dans un premier temps, il faut rendre possible la gestion des mouvements de la lampe via un PC à distance grâce au raspberry Pi 3 préconfiguré en point d'accès wifi. Nous avons besoin d'envoyer au PC les images de la webcam et de connecter au PC les moteurs de la lampe. Ensuite nous devons implémenter deux algorithmes : l'un rendant les images de la webcam plus légères (compression en jpeg) pour qu'elles puissent être envoyées ; l'autre permettant de détecter un visage. Enfin nous allons nous servir de l'ensemble des données reccueillies et de l'algorithme de détection du visage pour envoyer des informations de commande à la lampe et ainsi organiser le suivi du visage à distance.

1 Transmission des informations

Le raspberry a été préconfiguré en point d'accès WIFI ce qui nous permet d'envoyer et de recevoir des informations à distance à ce dernier à partir d'un PC à distance. La première partie consiste alors : à créer un port usb virtuel sur le réseau local à l'image du port usb physique du raspberry connecté aux moteurs ; à configurer le raspberry en serveur UDP pour envoyer les images fournies par la webcam au PC.

1.1 Contrôle des moteurs à distance avec usbip

Nous avons à notre disposition un fichier python, INTERFACE.py, qui permet de régler l'orientation des 5 moteurs de la lampe suivant certains comportements ; cette interface permet en plus de visualiser les paramètres de chacun des moteurs comme la température ou l'angle de rotation, ce qui sera utile par la suite. Cependant, tout comme le code python qui assurera le suivi du visage, l'utilisation de ce fichier nécessite que les moteurs de la lampe soient branchés en USB sur l'ordinateur, ce qui n'est pas notre cas. Il faut alors partager ces moteurs, branchés au raspberry, à notre ordinateur distant. Cela est possible par exemple avec USB/IP, qui permet ce partage de manière simple à l'aide d'envoi de messages USB I/O sur le réseau.

Pour réaliser le partage du port USB, nous connectons l'ordinateur en WIFI au raspberry. Nous supposerons par la suite que l'adresse IP locale de ce dernier est **192.168.0.2**.

Il faudra faire attention lors de l'installation car il ne faut pas prendre le paquet **usbip**, obsolète et inutilisable depuis la version de Linux-3.17, mais le paquet **linux-tools-generic**. L'installation doit se faire aussi bien côté serveur (raspberry) que client (ordinateur).

1.1.1 Configuration du serveur

Il s'agit, de ce côté, d'identifier et de partager le port USB sur lequel sont branchés les moteurs. On commence par lancer le module et le serveur :

```
modprobe usbip_host
sudo /usr/lib/linux-tools/[Version]/usbipd
```

A chaque port USB est associé un identifiant, de la forme 1-1.1 . Il faut donc d'abord trouver l'identifiant du port recherché :

```
/usr/lib/linux-tools/[Version]/usbip list -l
>>
- busid 1-1.1 (0424:ec00)
  Standard Microsystems Corp. : SMSC9512/9514 Fast
  Ethernet Adapter (0424:ec00)

- busid 1-1.3 (16d0:06a7)
  MCS : unknown product (16d0:06a7)

- busid 1-1.5 (046d:0825)
  Logitech, Inc. : Webcam C270 (046d:0825)
```

Dans notre cas, nous souhaitons synchroniser le port sur lequel est branché "MCS" qui correspond aux moteurs. L'identifiant cherché est donc 1-1.3 . Il ne reste plus qu'à le mettre en partage :

```
sudo /usr/lib/linux-tools/[Version]/usbip bind -b 1-1.3
```

Les moteurs sont prêts à être partagés.

1.1.2 Configuration du client

De ce côté, il faut retrouver et attacher le port USB sur lequel sont branchés les moteurs.
On commence par exécuter le module "Virtual Host Controller Interface" :

```
modprobe vhci-hcd
```

L'identifiant du port à attacher est le même que celui trouvé précédemment. Il est néanmoins possible de l'obtenir depuis le client :

```
/usr/lib/linux-tools/[Version]/usbip list -r 192.168.0.2
>>
Exportable USB devices
======================
- 192.168.0.2
    1-1.3: MCS : unknown product (16d0:06a7)
         : /sys/devices/platform/soc/3f980000...
         : Communications / unknown subclass / unknown...
```

On retrouve bien l'identifiant 1-1.3 . Il ne reste plus qu'à l'attacher :

```
sudo /usr/lib/linux-tools/[Version]/usbip attach -r
192.168.0.2 -b 1-1.3
```

Les moteurs sont maintenant virtuellement branchés à notre ordinateur, permettant ainsi l'utilisation des fichiers python.

1.1.3 Automatisation du procédé

Les manipulations précédentes, bien que simples, deviennent très vite lassantes. Nous avons donc décidé de créer deux scripts, un pour le client (**client**) et l'autre pour le serveur (**scriptServ**), permettant l'automatisation du processus. Il a donc fallu que le script soit capable de détecter lui-même sur quel port étaient branchés les moteurs pour en déduire l'identifiant. Il était possible d'utiliser **awk** pour trouver la ligne de l'identifiant, puis d'extraire de cette ligne l'identifiant cherché. Cependant, n'étant pas familiers avec, nous avons préféré utiliser d'autres outils comme **grep**, **sed** ainsi que d'autres outils de base. Ainsi, chacun des deux scripts se décomposent en plusieurs étapes de la manière suivante :

- Récupération de l'emplacement exact de usbip
- Acquisition de la liste de périphériques USB
- Filtrage de la liste pour obtenir uniquement l'identifiant
- Synchronisation du port

Il est également possible de rajouter à la fin du script **client** une ligne exécutant le fichier de suivi du visage. Les codes sont disponibles en pièce jointe.

En ajoutant l'utilisateur pinokio aux *sudoers* pour le script **scriptServ**, nous avons pu faire en sorte de ne pas avoir à entrer le mot de passe sudo à chaque lancement du script. En revanche, l'exécution automatique au lancement du script, qui permettrait une accélération de la procédure de mise en route, a échoué. Nous avons tenté par plusieurs moyens de faire en sorte d'exécuter **scriptServ** au démarrage du raspberry : utilisation de update-rc.d, édition de .bashrc, /etc/rc.local et de cron, sans succès.

1.2 Envoi des images en serveur UDP

Il était théoriquement possible d'également partager la webcam par USB/IP. Cependant, transmettre un flux vidéo brut par réseau aurait été difficile. Dans cette partie, on s'intéresse donc à la création d'un serveur UDP sur le

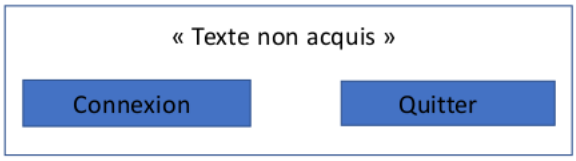
Raspberry qui devra envoyer les images de la webcam au PC distant. Cependant, les images constituant des fichiers lourds, on commence par envoyer des nombres entiers depuis le serveur qui sont générés par un compteur. C'est intéressant pour vérifier que :

- le PC reçoit les informations
- il n'y a pas de retard pour un envoi de données aussi légères
- en cas de déconnexion et de reconnexion il y a toujours un suivi des données

On se propose ici de coder deux fichiers python : **server.py** et **client.py**. Ainsi **server.py** se lance sur le Raspberry et **client.py** sur le PC.
Pour vérifier plus aisément le bon fonctionnement du serveur, on a créé une interface graphique à l'aide de la bibliothèque **tkinter**. Le tout doit être codé en Python 2 maximum car la bibliothèque liée à OpenCV (bibliothèque pour les fonctions de traitement de l'image dans la partie suivante) n'est disponible que sous cette version de Python.

1.2.1 Interface

L'interface graphique que nous utilisons pour tester notre serveur a cet aspect :



Ainsi après avoir lancé le script **server.py**, en appuyant sur "Connexion" nous devrions voir apparaître un défilement d'entiers à la place du label.

1.2.2 Côté serveur

Pour implémenter le serveur UDP, on utilise la bibliothèque "socket" de Python. Celle-ci nous laisse par ailleurs la possibilité de créer un serveur TCP ou UDP. Pour cela, il suffit de choisir respectivement le paramètre SOCK_STREAM ou SOCK_DGRAM lors de l'appel de :

```
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

On stocke ensuite les informations nécessaires pour créer le serveur : l'hôte et le port. L'hôte estc en fait l'adresse IP du serveur sur le réseau :

```
host = 192.168.0.2
```

On choisit le port 1080. Ce dernier est alloué pour les serveurs TCP/UDP :

```
port = 1080
```

Il ne reste plus qu'à faire le lien entre notre socket créé et ces informations :

```
sock.bind((host,port))
```

On démarre la boucle d'envoi des informations :

```
while(running)
```

— On attend la requête du client

```
data,address = sock.recvfrom(4)
```

— S'il n'y a pas d'erreur on renvoie l'entier

```
sock.sendto(x.toBytes(),address)
```

1.2.3 Côté client

On crée cette fois-ci un socket qui n'est lié à aucun port et adresse : il ne fait qu'envoyer des requêtes et recevoir les réponses :

```
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sent = sock.sendto("get",(host,port))
data,server = sock.recvfrom(65507)
```

Ici data correspond en fait à notre nombre entier reçu. Il ne reste plus qu'à actualiser le label de notre fenêtre à chaque itération :

```
fenetre.label.setText()
```

On observe ainsi que notre serveur UDP fonctionne correctement pour un envoi de données. Pour adapter ce code avec les images de la webcam, on remplace "data" par les images à envoyer du côté serveur et à recevoir du côté client. Cependant on n'utilise plus l'interface graphique "Tkinter" qui interfère avec OpenCV lors de l'affichage des images reçues : on a donc abandonné

les boutons et laissé place à la fonction "imshow" de OpenCV. Toutefois la boucle qui gère la réception des données est inchangée.
Nous avons donc réussi à implémenter une structure de serveur UDP fonctionnelle. Néanmoins, il nous reste à traiter les images en amont (compression) et en aval (reconnaissance du visage).

2 Traitement de l'image

2.1 Compression des images en jpeg

En considérant que le Raspberry n'est pas assez puissant pour la détection des visages, il est nécessaire d'envoyer à partir de celui-ci les images à l'ordinateur de contrôle.

Dans un premier temps, nous allons récupérer les images en utilisant la bibliothèque OpenCV sous Python 2. plus précisément, nous avons utilisé la fonction VideoCapture(0) où 0 correspond à la webcam par défaut.

Pour envoyer les images, nous allons utiliser le serveur créé précédemment. Mais pour éviter une surcharge d'informations à envoyer de la part du Raspberry, nous devons compresser chacune des images. Pour cela, on va utiliser la fonction imencode de OpenCV en prenant le format "JPEG" avec un taux de compression à 10%. Il ne reste plus qu'à les envoyer. On récupère ensuite ces images à l'aide du client.

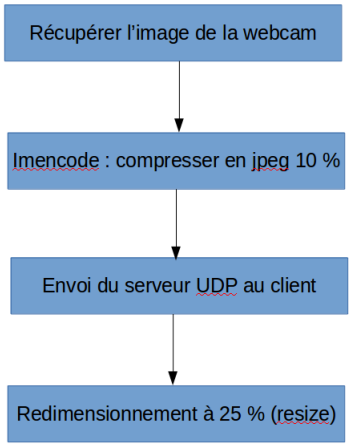
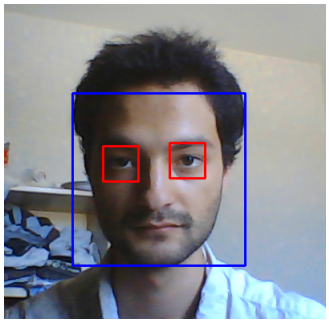


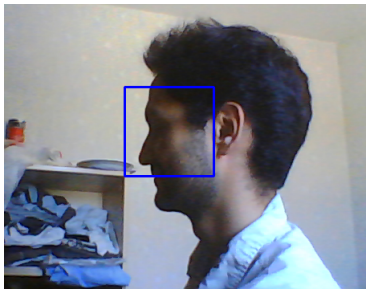
FIGURE 1 – Schéma résumé du transfert de l'image

2.2 Détection du visage

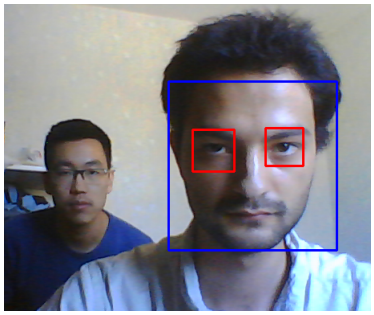
Pour récupérer les images de la webcam, il existe la fonction de OpenCV "VideoCapture" qui attend les images de la webcam. Pour la détection des visages, nous avons utilisé la méthode de Viola et Jones à l'aide de la bibliothèque OpenCV qu'ils ont déjà implémentée. Mais pour accélérer la rapidité de l'algorithme de détection, nous avons réduit la taille de l'image à traiter : ainsi, nous avons réduit jusqu'à 25% de la taille de l'image initiale en s'appuyant sur la fonction "resize" (nous sommes passé d'une image de 640x480 pixels à 160x120 pixels).

Après cela, à partir des fichiers "haarcascade" fournis par OpenCV, nous pouvons ainsi utiliser la détection du visage : nous nous sommes simplement concentrés sur le visage de face et de profil.





Cependant, pour éviter des problèmes futurs avec l'asservissement, nous avons considéré que la webcam doit seulement reconnaître un seul visage. Si plusieurs visages sont reconnus, le critère sur lequel la webcam se base est la surface de box de reconnaissance. En d'autres termes, plus la box de reconnaissance est importante, plus le choix sera porté sur ce visage.



Nous récupérons par ailleurs quelques informations utiles à l'asservissement : la position de la box et ses dimensions. Cela correspond à (x, y) et (hx, hy). À partir de ces informations on obtient le centre de la box :

$$x_{center} = x + \frac{l_x}{2}$$

$$y_{center} = y + \frac{l_y}{2}$$

3 Asservissement

Le but de l'asservissement est que la lampe puisse suivre le visage d'une personne. Pour cela, on réutilise les informations obtenues précédemment le centre de la box la résolution des images après "resize" qui est en posant r_x et r_y la résolution de la webcam :

$$r'_x = 0.25 \times r_x$$

$$r'_y = 0.25 \times r_y$$

En posant les variables a et b qui sont les angles des moteurs qui sont les angles des moteurs 4 et 3 qui va faire bouger respectivement la caméra horizontalement et verticalement, nous avons, avec $a = 0$ et $b = 0$ initialement :

$$a = a + 5 \times \frac{r'_x/2 - x_{center}}{r'_x/2}$$

$$b = b + 5 \times \frac{r'_y/2 - y_{center}}{r'_y/2}$$

En d'autres termes, la déviation angulaire horizontale (respectivement verticale) est proportionnelle à la différence de pixels entre les deux abscisses (respectivement ordonnées) des centres de la box et de l'image. Puis on exécute une fonction "goto_position" en envoyant les nouveaux paramètres a et b avec un temps de repos de 1/10 de secondes. Nous remarquons que le suivi est bien réalisé mais que nous avons des saccades et de fortes variations lorsque les deux centres sont très éloignés. Pour régler cela, nous avons modifié le temps d'exécution de la fonction "goto_position" à l'aide d'une exponentielle qui va atténuer les fortes variations. D'où ce temps d'exécution "timerexe" est égal à :

$$timerexe = e^{1.5*(|d_x/(r'_x/2)|+|d_y/(r'_y/2))|)} * 0.1$$

Où :

$$d_x = r'_x/2 - x_{center}$$

$$d_y = r'_y/2 - y_{center}$$

En utilisant "timerexe" la lampe exécute les mouvements entre 0,1s dans les cas les plus favorables et de 2s dans les cas les plus défavorables, c'est-à-dire aux "coins de l'image".

Par ailleurs, lorsque l'algorithme de détection ne détecte pas de visage on applique un algorithme de fonction de recherche à la lampe c'est-à-dire que la lampe va bouger horizontalement sa tête avec une ouverture de 180 et s'arrête lorsqu'il a détecté un visage. (On modifie la valeur de a tous les dixièmes de secondes on augmente de 4).

Enfin, quand le centre de la box est proche du centre de la webcam (une différence de 10 pixels) on souhaite que la lampe puisse atteindre un état d'équilibre le plus rapidement possible. On va donc imposer un seuil limite : si la différence de pixels entre les deux centres est inférieure à 10 pixels la lampe est fixe et ne bouge pas. Inversement, si elle n'atteint pas ce seuil elle s'adapte.

4 Modifications sur la lampe

4.1 Problèmes de déconnexions

Dans le cadre de notre étude de la lampe, nous avons été confrontés à divers problèmes d'ordre plus techniques, auxquels il a fallu trouver des so-

lutions pratiques et faciles à mettre en oeuvre. En effet, ces problèmes nous empêchaient de pouvoir avancer sur la partie du projet de laquelle nous devions nous charger à la base. Ce problème était le suivant : souvent, pendant une utilisation de la lampe, nous perdions le contact entre les moteurs situés sur la lampe et l'interface du Raspberry. Il nous a donc fallu émettre des hypothèses sur les raisons qui nuisaient au fonctionnement correct de cette lampe.

4.1.1 Consommation du moteur

La première idée qui fut évoquée était une surconsommation de courant de la part des moteurs. La lampe étant branchée sur une alimentation liée au secteur, dans le cas où notre consommation dépasse celle maximale permise par l'alimentation, l'alimentation doit probablement se couper afin d'éviter d'endommager les câbles ou le boîtier de l'alimentation.

Après une recherche sur les capacités de notre alimentation et de nos moteurs, nous avons vu que nous étions un peu juste au niveau de l'alimentation. En effet, nos moteurs consomment, pour un couple maximum, une valeur maximale de 2,2A, pour le modèle AX-18A, et 0.9A pour le modèle AX-12A. Notre alimentation, quant à elle, fournit un courant maximal de 5A. Avec les 5 moteurs alimentés et tournant au maximum, nous serions à une consommation de 8,8A, bien au dessus de la limite que nous pouvons nous imposer. De plus, il faut considérer que l'ensemble des systèmes auxiliaires bien que mineurs, comme l'ampoule située à l'entrée du circuit d'alimentation des moteurs.

Il a donc été nécessaire de regarder la consommation des moteurs pendant une période d'utilisation pratique, afin de savoir si nous dépassions ou non la limite de courant de notre alimentation. Nous avons donc fait ouvrir une partie de notre câble d'alimentation, pour pouvoir y placer un ampère-mètre.

Après mesures de notre consommation en utilisant chacun des moteurs, nous nous sommes aperçus que nous sommes en réalité bien en dessous de la limite permise par notre alimentation. Nous avons donc pu en conclure que l'alimentation est suffisante pour notre robot et que le problème venait d'autre part.

AX-12A Stats	
Operating Voltage	12V
Stall Torque*	15.3 kg·cm 212 oz·in
No-load Speed	59 RPM 0.169sec/60°
Weight	55g
Size	32 x 50 x 40 mm
Resolution	0.29°
Reduction Ratio	1/254
Operating Angle	300° or Continuous Turn
Operating Voltage	9-12V (Recommended Voltage 11.1V)
Max Current	900 mA
Standby Current	50 mA
Internal Operating Temp	-5°C ~ 70°C
Protocol	TTL Half Duplex Async Serial
Module Limit	254 valid addresses
Com Speed	7343bps ~ 1Mbps
Position Feedback	Yes
Temp Feedback	Yes
Load Voltage Feedback	Yes
Input Voltage Feedback	Yes
Compliance/PID	Yes
Material	Plastic Gears and Body
Motor	Cored Motor
Manual	AX-12A manual

AX-18A Stats	
Operating Voltage	12V
Stall Torque*	18.3 kg·cm 254.9 oz·in 1.8 Newton Meter
No-load Speed	97rpm 0.103 sec/60°
Weight	54.5g
Size	32 x 50 x 40mm
Resolution	0.29°
Reduction Ratio	1/254
Operating Angle	300° or Continuous Turn
Max Current	2200 mA
Standby Current	50 mA
Operating Temp	-5°C ~ 85°C
Protocol	TTL Half Duplex Async Serial
Module Limit	254 valid addresses
Com Speed	7343bps ~ 1Mbps
Position Feedback	Yes
Temp Feedback	Yes
Load Voltage Feedback	Yes
Input Voltage Feedback	Yes
Compliance/PID	Yes
Material	Plastic Gears and Body
Motor	Cored Motor
Manual Download	AX-18A manual

FIGURE 2 – Intensité maximale des moteurs

4.2 Retouches de connectique

Après quelques temps d'expérimentation, nous nous sommes aperçu d'un élément qui pouvait nous guider dans notre recherche d'une source de problèmes pour la connectique de nos moteurs. En effet, en manipulant les câbles, que ça soit manuellement ou lié à l'entraînement de la lampe, il arrivait parfois que le système s'arrête brusquement et se mette à redémarrer.

Bien que ce problème de câble fournissait une alternative bien pratique afin de pouvoir redémarrer la lampe, il fallait y remédier afin de pouvoir utiliser l'ensemble des moteurs sans avoir à craindre que le tout s'éteigne brusquement. Deux problèmes pouvaient se présenter à nous au sujet des câbles. On pouvait soit avoir des problèmes de connexions entre les connecteurs, soit avoir des problèmes de ruptures au niveau du câble. En s'intéressant davantage aux câbles, nous nous sommes aperçu qu'il y avait bien un peu de jeu entre le connecteur, les câbles qui en partaient et le moteur.

Une première tentative fut tout d'abord d'attacher les câbles entre eux pour réduire le jeu qui pouvaient exister lors de leur manipulation à l'aide de serres-câbles en plastique. On a en effet constaté une légère amélioration,

mais le problème de déconnections subsiste toujours.

Une autre tentative fut de démonter les embouts plastiques des câbles afin de s'assurer qu'ils étaient correctement montés et de les remonter avec une pince dans le cas contraire. On s'est aperçu que quelques câbles avaient ce problèmes d'embouts mal connectés, et après correction, il s'avère que l'ensemble du montage fonctionne mieux.

4.3 Problème d'échauffement

Un autre phénomène que nous avons constaté avec l'un des moteurs est le fait qu'il s'échauffe de manière extrêmement rapide lors d'une utilisation même courte. Le moteur est capable de mesurer sa propre température interne et de la transmettre à l'interface de la lampe. Et on voit qu'on arrive facilement à dépasser les 75°C après 10 minutes d'utilisation sur le moteur qui sert de base à la lampe. Cependant, cette hausse de température pose un problème majeur qui peut expliquer certaines pertes du contact avec les moteurs.

En effet, selon la documentation des moteurs, le AX-12A est conçu pour fonctionner entre -5°C et 70°C et le AX-18A peut fonctionner entre -5°C et 75°C. On fait donc effectivement face au problème suivant : on sort très rapidement de la plage de température du moteur. Il est possible que lorsque le moteur détecte qu'il est hors de la plage de température, il se coupe pour éviter tout risque de détérioration. Et comme tous les moteurs sont liés entre eux, si l'un se déconnecte de l'ensemble du système, on peut imaginer qu'il va entraîner les autres avec lui.

On a donc 2 solutions possibles. Soit on cherche à refroidir le moteur pour ne pas atteindre la température limite pendant l'utilisation, soit on cherche à modifier quelque chose dans la lampe pour éviter que le moteur chauffe autant. Cependant, l'installation d'un système de refroidissement aurait été

AX-12A Stats	
Operating Voltage	12V
Stall Torque*	15.3 kg·cm 212 oz·in
No-load Speed	59 RPM 0.169sec/60°
Weight	55g
Size	32 x 50 x 40 mm
Resolution	0.29°
Reduction Ratio	1/254
Operating Angle	300° or Continuous Turn
Operating Voltage	9-12V (Recommended Voltage 11.1V)
Max Current	900 mA
Standby Current	50 mA
Internal Operating Temp	-5°C ~ 70°C
Protocol	TTL Half Duplex Async Serial
Module Limit	254 valid addresses
Com Speed	7343bps ~ 1Mbps
Position Feedback	Yes
Temp Feedback	Yes
Load Voltage Feedback	Yes
Input Voltage Feedback	Yes
Compliance/PID	Yes
Material	Plastic Gears and Body
Motor	Cored Motor
Manual	AX-12A manual

AX-18A Stats	
Operating Voltage	12V
Stall Torque*	18.3 kg·cm 254.9 oz·in 1.8 Newton Meter
No-load Speed	97rpm 0.103 sec/60°
Weight	54.5g
Size	32 x 50 x 40mm
Resolution	0.29°
Reduction Ratio	1/254
Operating Angle	300° or Continuous Turn
Max Current	2200 mA
Standby Current	50 mA
Operating Temp	-5°C ~ 85°C
Protocol	TTL Half Duplex Async Serial
Module Limit	254 valid addresses
Com Speed	7343bps ~ 1Mbps
Position Feedback	Yes
Temp Feedback	Yes
Load Voltage Feedback	Yes
Input Voltage Feedback	Yes
Compliance/PID	Yes
Material	Plastic Gears and Body
Motor	Cored Motor
Manual Download	AX-18A manual

FIGURE 3 – Plage de températures permises

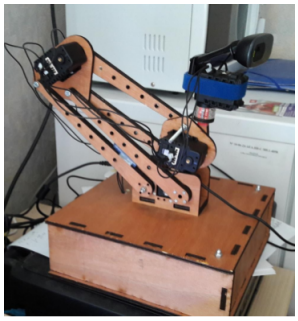
un peu compliqué à inclure dans le temps qu'il restait pour travailler sur la lampe, et nous avons donc chercher des techniques pour diminuer les efforts du moteurs et donc la chaleur qu'il générait.

4.4 Retouches mécaniques

Après avoir cherché ce qui pouvait importuner le moteur de la lampe pendant son fonctionnement, nous avons trouvé 3 axes de travail qui ont été

développés.

4.4.1 Réajustement de la structure



Nous avons tout d'abord tenté d'alléger la structure en retirant certaines pièces de bois qui n'étaient pas indispensable au fonctionnement de la lampe. Par exemple, l'ensemble de la partie de la tête de la lampe avec la webcam a été modifié pour remplacer toute la tête par un scratch capable de tenir la webcam.

De même, nous avons tenté de retirer certaines pièces annexes qui servaient à tenir les bras de la lampe afin de diminuer le poids qui était imposé par la lampe sur le moteur.

Cependant, nous avons constaté que ces modifications avaient un effet très peu visible sur le poids de la lampe en général, qui est lié au moteur principalement, et donc un effet sur l'échauffement des moteurs assez peu considérable.

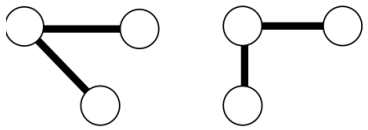
4.4.2 Diminution des frottements

Une autre idée fut d'installer de quoi diminuer les frottements entre toutes les parties de bois de la lampe. Par exemple, des rondelles de papiers ou de plastiques permettraient d'éviter le contact bois/bois et donc de permettre au moteur d'avoir à fournir moins d'effort pour pouvoir faire bouger le reste de la lampe.

Nous avons donc utilisé des morceaux de plastiques découpés de manière à ce que l'axe de chaque moteur en ait un, et puis nous en avons mis sur toutes les pièces qui possédaient une zone de contact bois/bois qui pouvait poser problème.

Cependant, comme pour le réajustement de la structure, les effets de ces rondelles de plastiques sur l'échauffement total du moteur ont étés négligeables, et ils nous a donc fallu trouver une autre solution.

4.4.3 Modification de la position de la lampe



Enfin, une dernière solution fut de modifier la position de base de la lampe, afin d'éviter que le poids soit inutilement concentré sur un seul point de la lampe, et de permettre donc de mieux répartir les efforts.

En modifiant donc la position de base de la lampe, nous pourrions alors déplacer les efforts exercés sur le moteur de base pour les répartir partout, ce qui limiterait donc l'échauffement de ce moteur lors de son fonctionnement.

Néanmoins, comme cette solution n'a été pensée qu'en fin de projet, nous n'avons pas eu l'occasion de tester beaucoup de positions pour la lampe, et nous n'avons donc pas pu voir si certaines provoquaient un échauffement moindre du moteur.