Raspberry humanoide – R0B1

Francelet Samy – Gaspoz frédéric

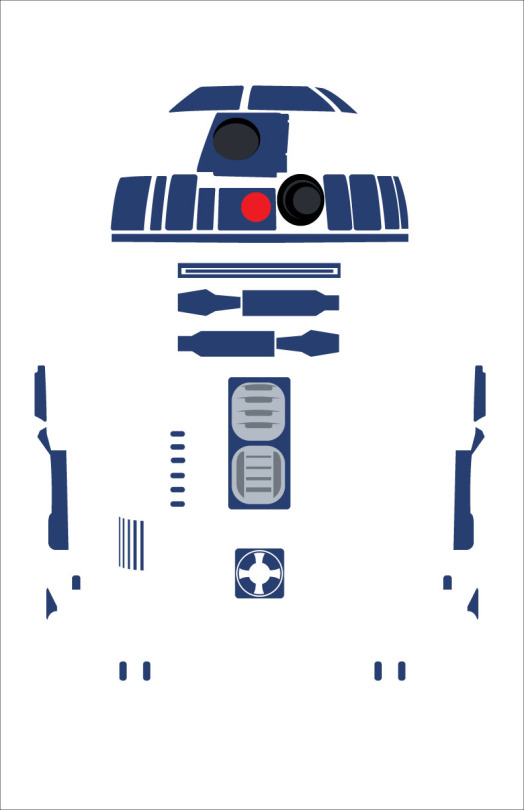


Table des matières

[1 Introduction 4](#_Toc484786138)

[2 Description 5](#_Toc484786139)

[2.1 Cahier des charges 5](#_Toc484786140)

[2.2 Schéma bloc 6](#_Toc484786141)

[2.3 Planning 9](#_Toc484786142)

[2.3.1 Planning prévu 9](#_Toc484786143)

[2.3.2 Journal de bord et synthèse 10](#_Toc484786144)

[3 Etude et développements 12](#_Toc484786145)

[3.1 Description des variantes possibles 12](#_Toc484786146)

[3.1.1 Alimentation du circuit 12](#_Toc484786147)

[3.1.2 Communication avec les moteurs 16](#_Toc484786148)

[3.1.3 Plaques du torse et du visage 18](#_Toc484786149)

[3.1.4 Schématique complète 19](#_Toc484786150)

[3.1.5 Routage 20](#_Toc484786151)

[3.1.6 Interface utilisateur 22](#_Toc484786152)

[3.1.7 Supports des moteurs 27](#_Toc484786153)

[3.1.8 Montage final du robot 28](#_Toc484786154)

[3.2 Programmation 29](#_Toc484786155)

[3.2.1 Résumés des fichiers 29](#_Toc484786156)

[3.2.2 Communication avec les moteurs 33](#_Toc484786157)

[3.2.3 Gestionnaire de fichier 34](#_Toc484786158)

[3.2.4 Fonctions simplifiées 36](#_Toc484786159)

[4 Résultats obtenus 37](#_Toc484786160)

[4.1 Problèmes rencontrés 37](#_Toc484786161)

[4.2 Mesures obtenues 39](#_Toc484786162)

[4.2.1 Alimentation 39](#_Toc484786163)

[4.2.2 Communication 39](#_Toc484786164)

[4.3 Analyse des résultats 40](#_Toc484786165)

[4.4 Tableau récapitulatif des résultats 40](#_Toc484786166)

[5 Orientations futures et Conclusion 41](#_Toc484786167)

[5.1 Orientations futures 41](#_Toc484786168)

[5.2 Conclusion 41](#_Toc484786169)

[6 Annexes 42](#_Toc484786170)

Résumé du projet

Ce projet se présente sous forme d’un robot, dont les mouvements sont programmables simplement, juste en bougeant les bras. L’interface sera simple pour éviter de perdre l’utilisateur dans des milliers de paramètres, et sera sur un écran tactile. L’écran servira aussi de visage au robot lorsqu’il répétera les mouvements enregistrés, un ou plusieurs visages seront sélectionnables.  
L’objectif est d’en faire un « condensé de technologie » pour donner envie aux enfants de se lancer dans l’électronique.

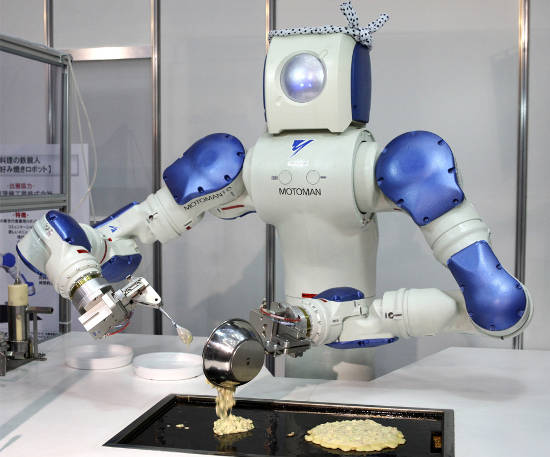
Plus tard, la reconnaissance vocale de Google sera ajoutée sur le robot pour pouvoir le contrôler avec la voix.

# Introduction

De nos jours, la robotique et l’automatisation connaissent un essor énorme. Cette expansion a commencé à faire rêver certaines personnes de robot qui ferait la cuisine ou autres tâches ménagères à leur place. Seulement, il existe très peu de robot de ce type commercialisé, et souvent leur mouvement sont soit fixes et donc simple d’utilisation mais limité, soit programmable et donc avec des possibilités infinies mais demande des connaissances avancées en programmation.

L’idée de combiner utilisation simple et programmation simple permettrait d’avoir un robot correspondant aux attentes de tous les utilisateurs.  
Programmer son robot en lui faisant faire les mouvements nécessaire à la recette de cuisine et que par la suite le robot sache répéter les mouvements de manière autonome permettrait de satisfaire ces deux conditions. Il serait imaginable de rajouter une reconnaissance vocale par la suite

Utiliser un Raspberry Pi 3 et un écran tactile comme système de contrôle permettrait d’avoir une interface simple, et de laisser la possibilité aux gens de faire des modifications simplement s’ils le souhaitent.



# Description

## Cahier des charges

* Le changement du visage vers l’interface se fera via une pression sur l’écran, et le changement de l’interface vers le visage se fera en sélectionnant un programme
* Les moteurs utilisés sont les AX-12A pour leur mode de communication utile à la récupération des valeurs de position pour la programmation.
* Plusieurs programmes pourront être enregistrés dans la mémoire du Raspberry, ils seront tous enregistrer dans des fichiers, ce qui permettra de ne pas perdre les données à l’extinction.
* Les bras seront facilement programmables, simplement en les bougeant.
* Si possible, au démarrage du Raspberry, le programme sera lancé automatiquement en plein écran. Le programme pourra être fermé avec un bouton tactile.
* Le code sera en Python.
* L’interface graphique sera faite sous Kivy.
* Le robot sera alimenté par le secteur, alimentation 12V.
* Il y aura un régulateur de tension pour créer du 5V pour le Raspberry et l’écran.
* Un ou plusieurs visages animés seront enregistrer dans le Raspberry, et on pourra choisir le visage pour chaque programme (optionnel)
* La conception mécanique du robot sera réalisée en 3D sur ordinateur.

## Schéma bloc

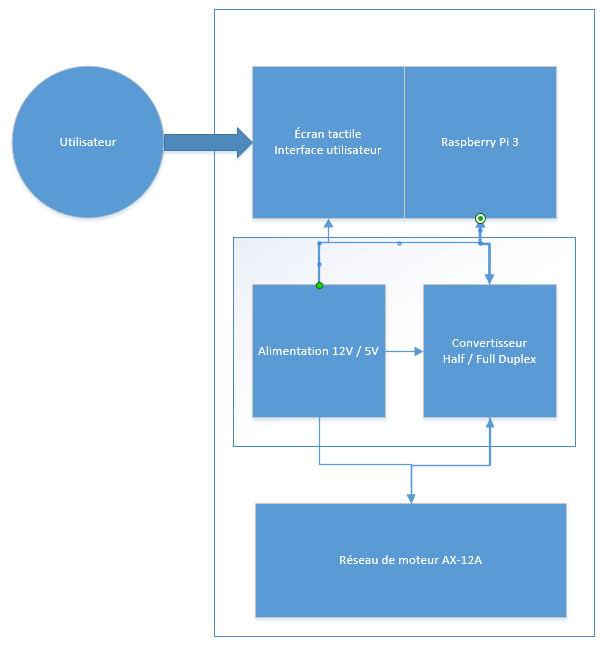


Figure 1 - Schéma bloc du robot

L’utilisateur contrôlera le robot via l’écran tactile, qui fait office d’interface utilisateur et de visage. Le Raspberry, en fonction des actions de l’utilisateur, enverra ou recevra des informations des moteurs via le port série. Le signal du port série du Raspberry est en Full Duplex, c’est-à-dire qu’il a un signal RX et un signal TX, les moteurs n’ont qu’un seul signal qui contient le RX et le TX, ils sont donc en half duplex, il faut donc un convertisseur pour cela. Le Robot entier est alimenté avec une seule prise d’alimentation.

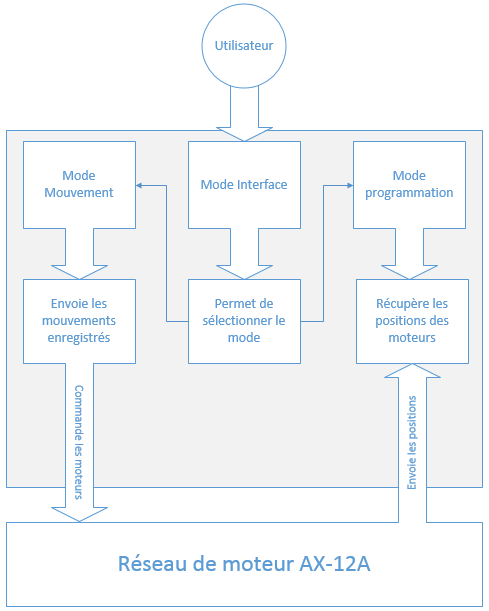


Figure 2 - Architecture software basique

L’utilisateur aura accès à 3 modes : le mode interface qui est un hub pour passer à un autre mode, le mode programmation sert à lancer la programmation des bras, et stockera tous les mouvements effectués sur les moteurs, Le mode mouvement fera reproduire les mouvements enregistrés.  
Le code contiendra trois parties principales : L’interface utilisateur, le driver des moteurs et un gestionnaire de fichier pour stocker les positions.

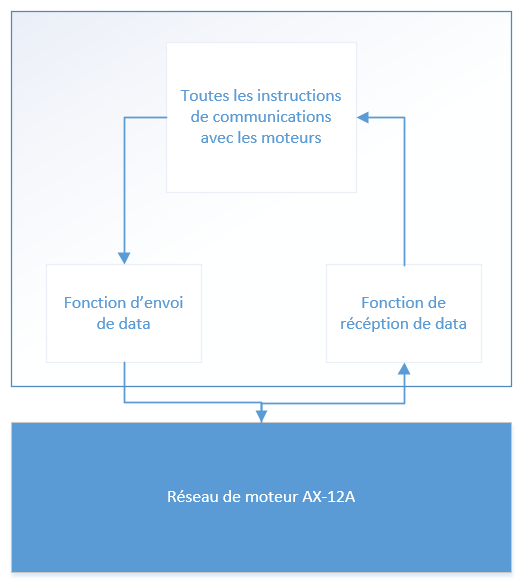


Figure 3 - Architecture basique du driver

Pour la communication, il est important d’utiliser des fonctions normalisées d’envoi et de réception pour éviter d’avoir une méthode d’envoi et de réception pour chaque instruction que l’on souhaite envoyer aux moteurs.

## Planning

### Planning prévu

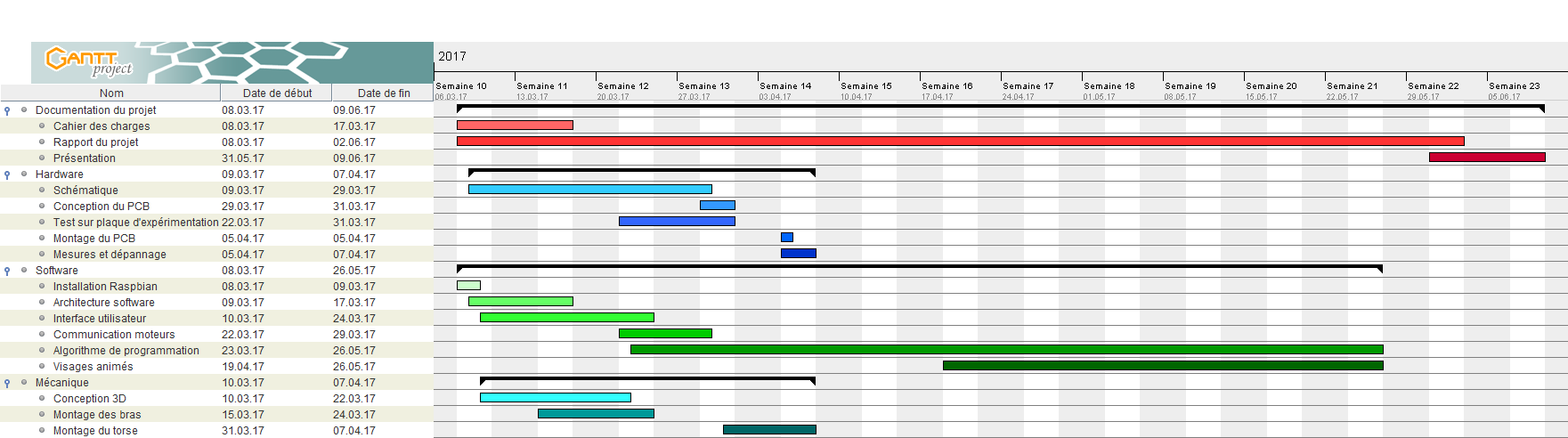


Figure 4 - Planning gantt prévu

### Journal de bord et synthèse

|  |  |  |  |
| --- | --- | --- | --- |
| Semaine | Mercredi | Jeudi | Vendredi |
| 06.03.17 | Schéma bloc + architecture software globale  Estimations des coûts  Cahier des charges | Plannification  Installation de Raspbian  Début de la schématique | Installation de Kivy et tests  Schématique |
| 13.03.17 | Découverte des fonctionnalités de Kivy  Montage d’un bras et tests des moteurs | Test du bras avec le module fourni  Réalisations de petits programmes sous Kivy | Début de l’interface utilisateur Modification de la schématique Modification de la documentation du projet |
| 20.03.17 | Interface utilisateur  Montage d’un circuit de test | Révision pratique  Paramétrage du Raspberry pour l’UART  Mesures sur le circuit de test | Essais de communications UART en Python  Test de contrôle d’un moteur |
| 27.03.17 | Commande des composants Protocole de mise en service du Raspberry Programmation du driver moteur | Programmation du driver moteur  Routage du PCB | Fin du routage  Test du driver moteur |
| 03.04.17 | Modification du routage | Tirage du PCB  Conception de la mécanique | Programmation du gestionnaire de fichier |
| 24.04.17 | Absent | Révision pratique  Montage du PCB SMD  Mise en service du PCB | Recherche de la source du problème de communication  Réécriture du driver pour corriger le problème |
| 01.05.17 | Finalisation du driver  Test du driver | Révision pratique  Protocole de test du driver | Schématique de montage mécanique  Demande d’offre Kanya  Design plaque du torse |
| 08.05.17 | Finalisation du gestionnaire de fichier  Tirage d’une plaque de torse | Protocole de test du gestionnaire de fichier | Conception 3D du support moteur  Création d’une lib pour utiliser facilement le driver via l’interface utilisateur |
| 15.05.17 | Test de mémorisation et de répétition d’un mouvement entier | Modification du driver pour corriger un problème de timeout | Finitions et test de la library |
| 22.05.17 | Test d’enregistrement d’un long mouvement, puis lecture du mouvement après redémarrage | Congé | Redesign des plaques de support du visage pour avoir une meilleure fixation |
| 29.05.17 | Test du programme entier avec l’interface graphique  Assemblage du robot avec les bras | Corrections des bugs liés à l’interface graphique | Ecriture du rapport  Examen écrit de Français |

La mécanique a été réalisée plus tard dans le projet, et finalement la programmation aura duré pendant tout le projet. Quelques problèmes ont été rencontrés, comme le problème du driver moteur qui m’aura fait réécrire le code du driver en entier. Pour le reste, la planification a plus ou moins été respectée. La réalisation des visages animés a été mise de côté, car l’apprentissage d’un logiciel d’animation aurait pris beaucoup trop de temps.

# Etude et développements

## Description des variantes possibles

### Alimentation du circuit

Pour l’alimentation du circuit, il faut pouvoir fournir assez de courant pour les moteurs en 12V, le Raspberry Pi 3 et l’écran tactile en 5V.

Pour le 12V, le transformateur 230VAC / 12VDC – 5 A fourni avec le kit robot sera utilisé.

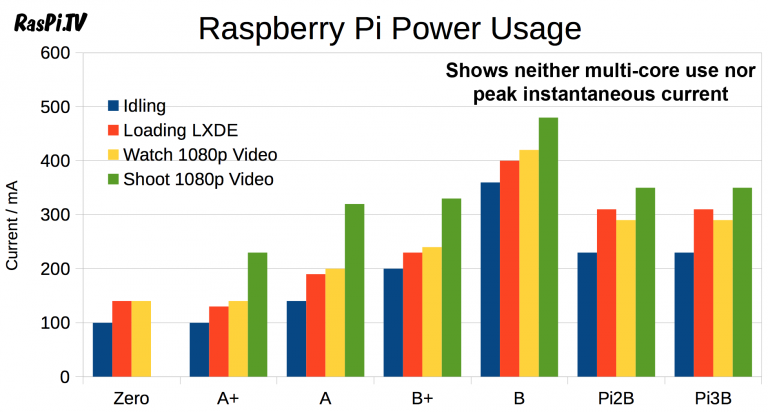


Figure 5 - Consommation des Raspberry Pi

Pour prendre de la marge, et parce que le graphe ne prend pas en compte le multi-core, on peut considérer que le Raspberry Pi 3B consomme 600mA. L’écran consomme plus de 500mA selon le site officiel, qu’on arrondit à 700mA pour avoir de la marge. Ce qui fait un total de 1,3A

De l’autre côté, en mesurant on obtient 430mA pour le Raspberry Pi 3, et l’écran donne 500mA pour afficher une image avec plusieurs couleurs. Donc les 1,3A semblent corrects, voir 1,5A pour prendre en compte les pics de courants qu’il peut y avoir.

#### LM7805

Le LM7805 est simple d’utilisation et peu cher, mais son grand défaut est son rendement faible. Il va beaucoup trop chauffer et ne pourra jamais driver 1,3A. De plus même si cette solution est choisie et qu’on trouve un régulateur pouvant driver 1,5A, il va dissiper beaucoup de puissance en chaleur, ce qui rendra nécessaire l’ajout d’un dissipateur de chaleur, donc prendra plus de place pour pas grand-chose.

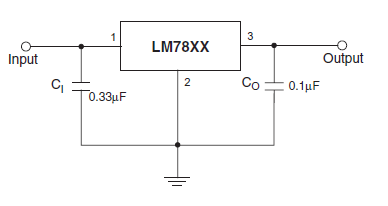


Figure 6 - Typical Application du LM7805

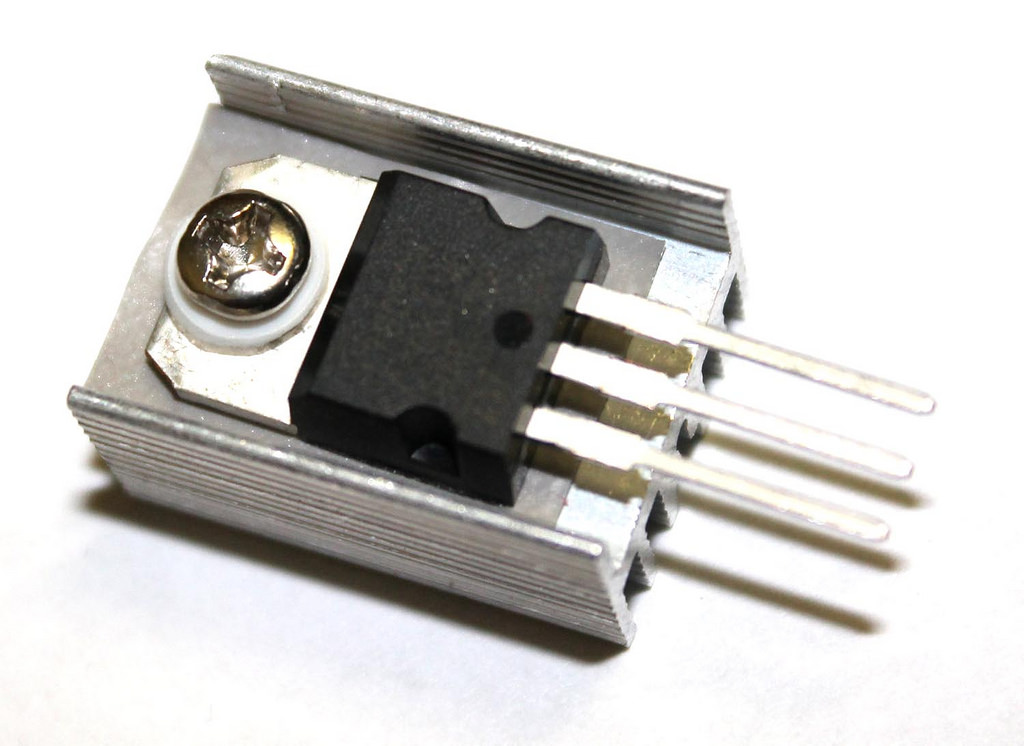


Figure 7 - LM7805 avec un dissipateur

#### Alimentation à découpage

Une alimentation à découpage serait très efficace, mais cela prendrait trop de temps à mettre en place car c’est un système compliqué.

Le principe de l’alimentation a découpage est de prendre une tension DC, et de la couper avec un transistor ou autre, pour créer un signal carré avec un duty cycle réglable. Ensuite en lissant le signal carré, on obtient une tension DC correspondant plus ou moins à la tension moyenne du signal carré.

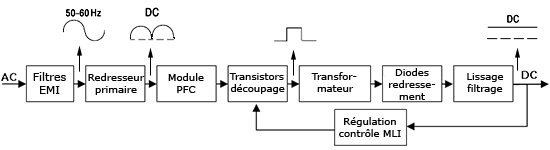


Figure 8 - Principe de l'alimentation à découpage

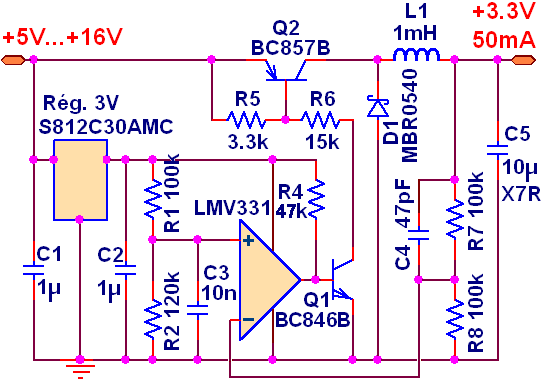


Figure 9 - Exemple d'alimentation à découpage

#### Module convertisseur DC/DC

Un module convertisseur DC/DC est en fait une alimentation à découpage dans un petit boitier. Cela permet d’avoir l’efficacité d’une alimentation à découpage, sans le temps de développement pour en créer une. Aussi, étant des alimentations à découpage, leurs rendements sont excellents (entre 80% et 95%).

C’est cette solution-là qui a été choisie. Le RECOM R-78B5.0-1.5 a été retenu, car il permet de convertir du 12V en 5V, et de fournir 1,5A en sortie. De plus son rendement de 92% fait qu’il chauffera très peu, évitant ainsi de devoir ajouter un dissipateur.

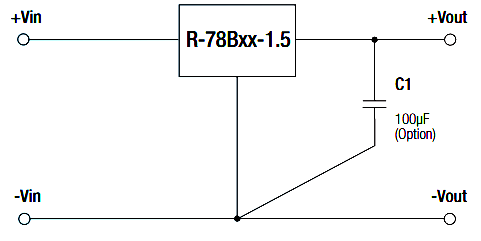


Figure 10 - Typical Application du Recom

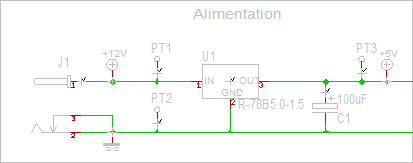


Figure 11 - Schéma utilisé du Recom

### Communication avec les moteurs

Les servomoteurs utilisés sont les AX-12A

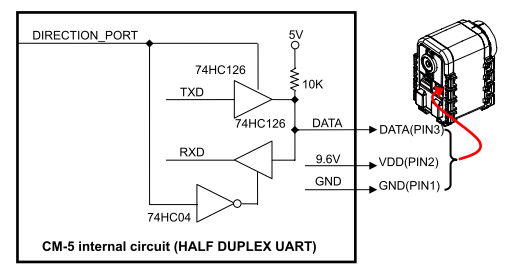


Figure 12 - Circuit proposé par le fabricant

Le circuit proposé a été conservé, mais l’IC a été remplacé par un 74HC241, car il possède deux DIRECTION\_PORT, un actif bas et l’autre actif haut, ce qui permet de contrôler la direction de la transmission avec une GPIO du Raspberry.

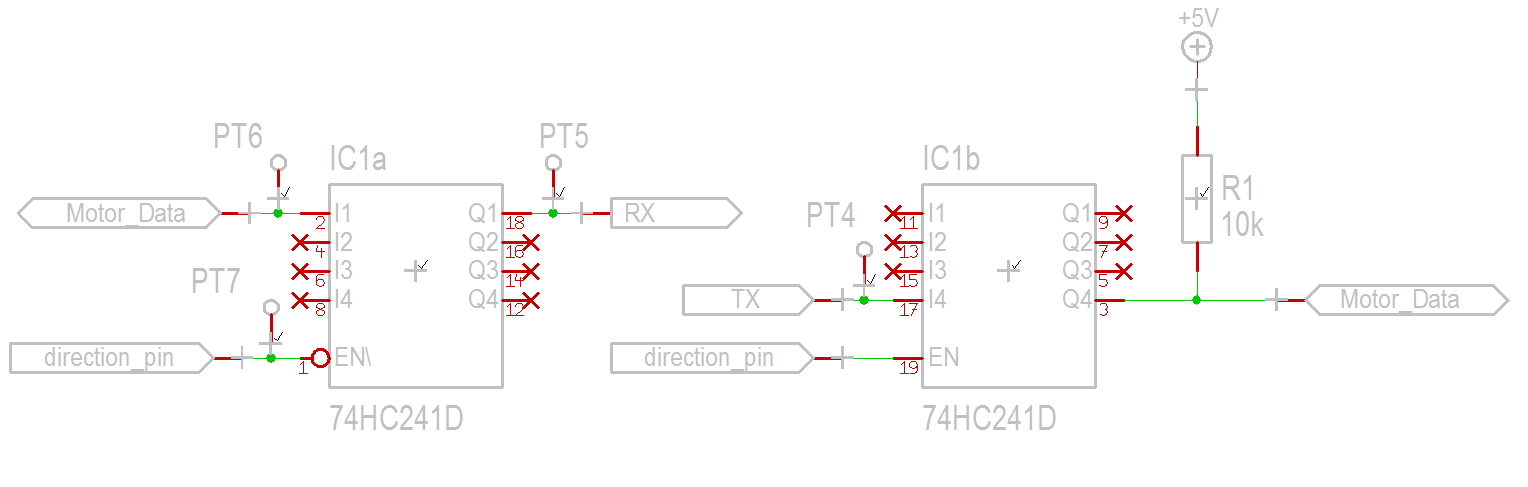


Figure 13 - Schéma du circuit utilisé

Pour l’alimentation des moteurs, le transformateur fourni avec le kit robot sera utilisé.  
Sur le circuit, il y aura 4 broches pour se connecter aux moteurs, 2 auraient peut-être suffit, mais pour éviter d’avoir trop de courant sur un seul fil, il a été préféré de rajouter plus de broches. Aussi cela peut permettre d’étendre le circuit

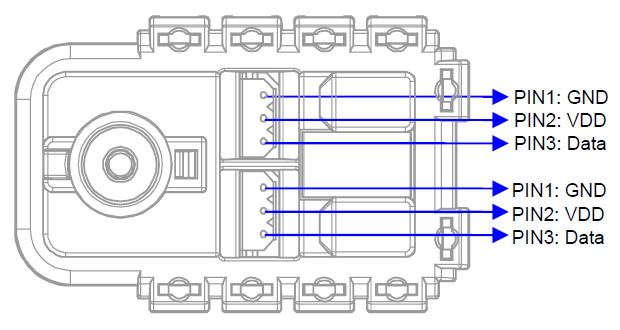


Figure 14 - Schéma de branchement du moteur

### Plaques du torse et du visage

Les plaques du torse et du visage ont été réalisées sous Illustrator, car le format de fichier vectoriel que fournit Illustrator permet d’être optimal pour une utilisation de la graveuse laser.

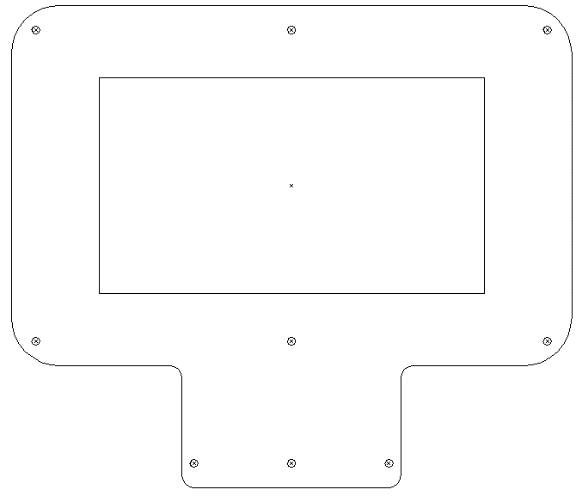


Figure 15 - Plaque frontale du visage

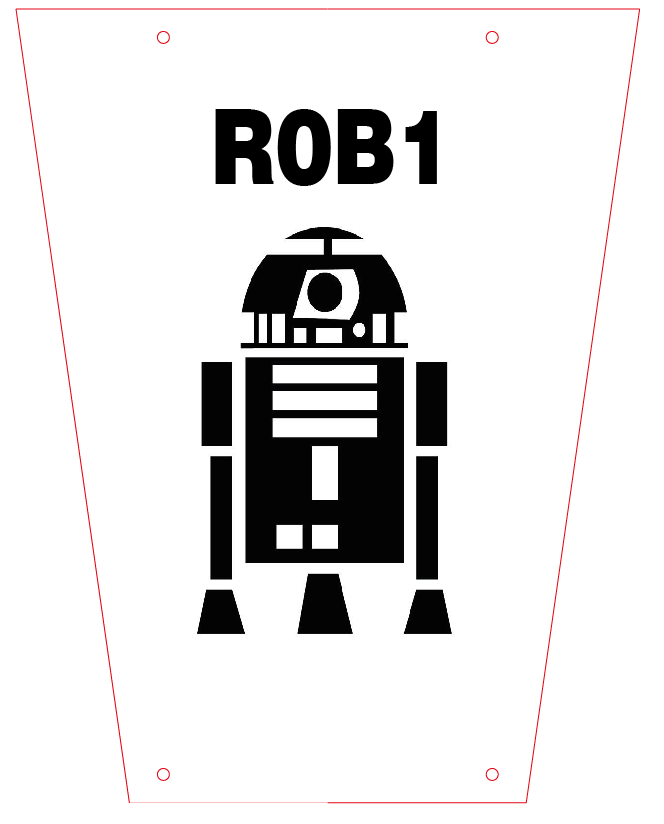


Figure 16 - Plaque du torse

### Schématique complète

La schématique est finalement très simple, car il n’y a qu’un seul composant. Des condensateurs ont été rajoutés sur l’alimentation des moteurs pour permettre de toujours alimentés en conséquence les moteurs.

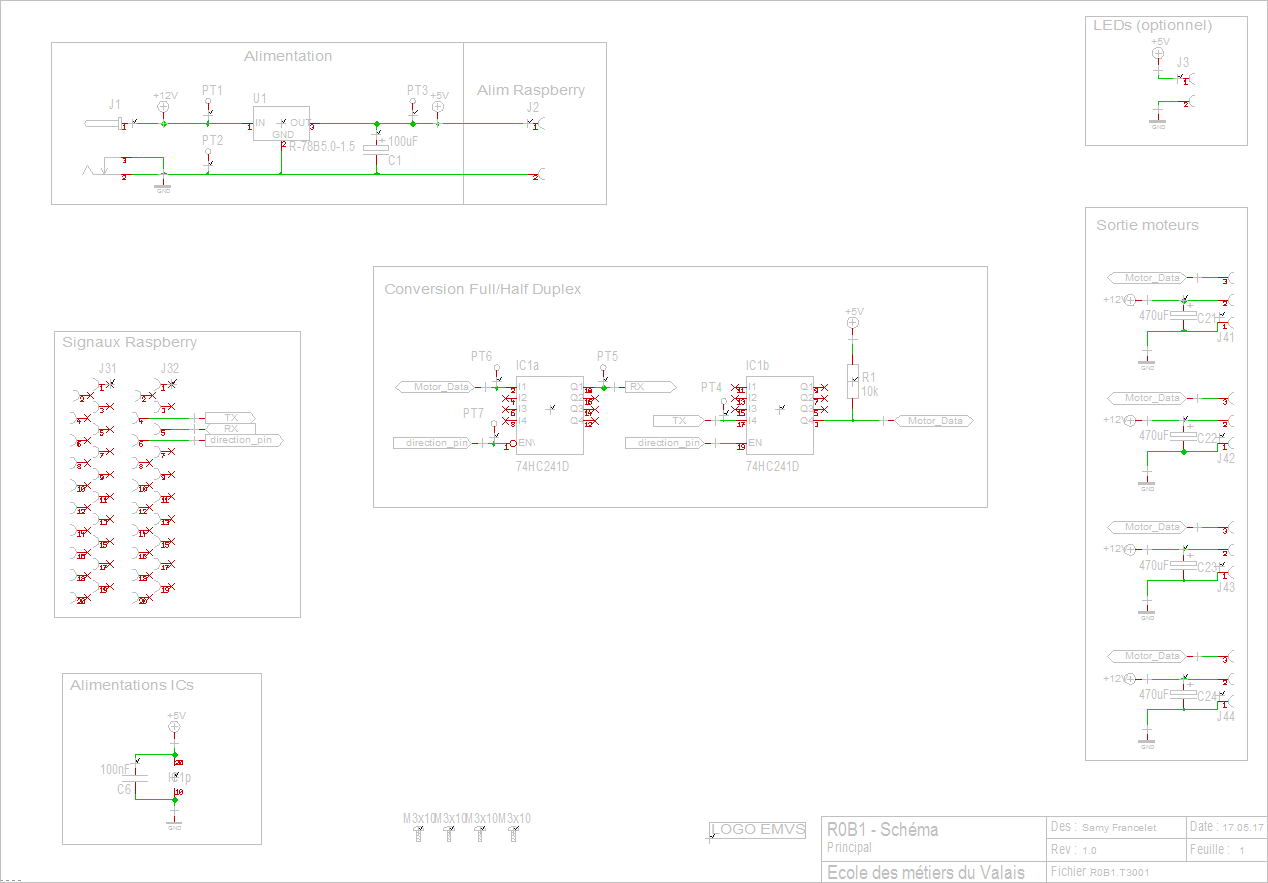


Figure 17 - Schéma complet

### Routage

Le PCB est lui aussi petit et simple, et sa taille est basée sur l’espace disponible sur le Raspberry. Les points de fixations sont eux aussi basés sur les fixations du Raspberry.

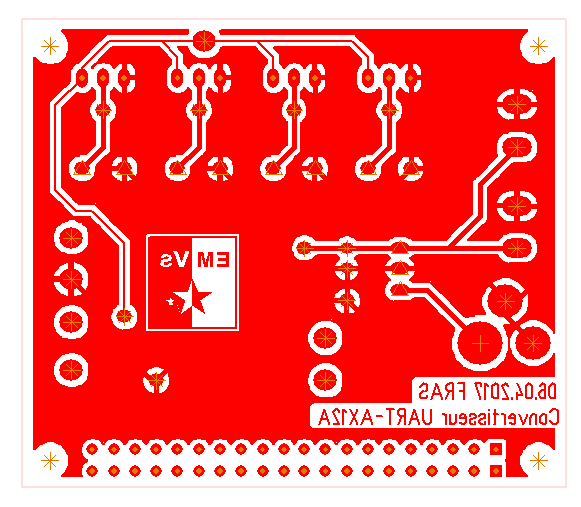


Figure 18 - Routage Bottom

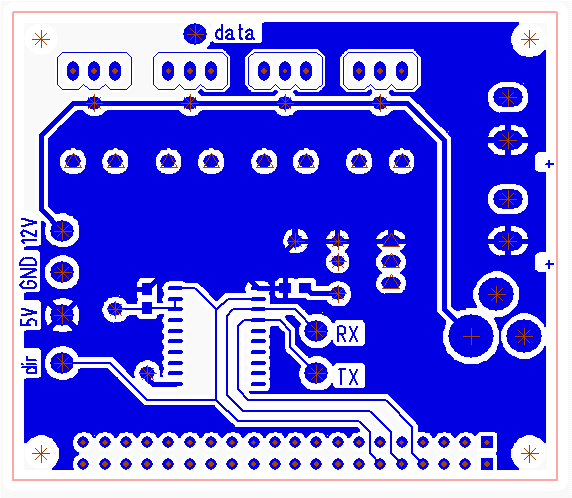


Figure 19 - Routage Top

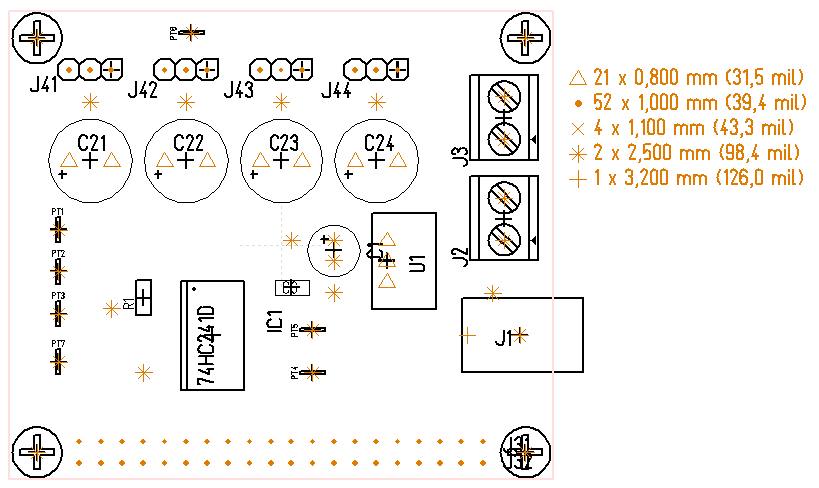


Figure 20 - Plan d'implémentation

### Interface utilisateur

Le choix du langage s’étant porté sur Python, et le système n’étant pas Windows, il est impossible d’utiliser Visual Studio de Microsoft. Il faut donc trouver une alternative permettant de faire de bonnes interfaces graphiques en Python, avec un bon support du tactile.

#### PyQt

PyQt est la version Python de Qt. Qt est un ensemble de bibliothèque permettant de faire une interface graphique. Qt a l’avantage d’être très bien développé, car utilisé par de nombreux professionnels (comme Adobe) et très utilisé au sein de la communauté Linux pour sa version Open Source. De nombreux tutoriels sont disponibles. Il possède un IDE très bien fait, égalant celui de Visual Studio.



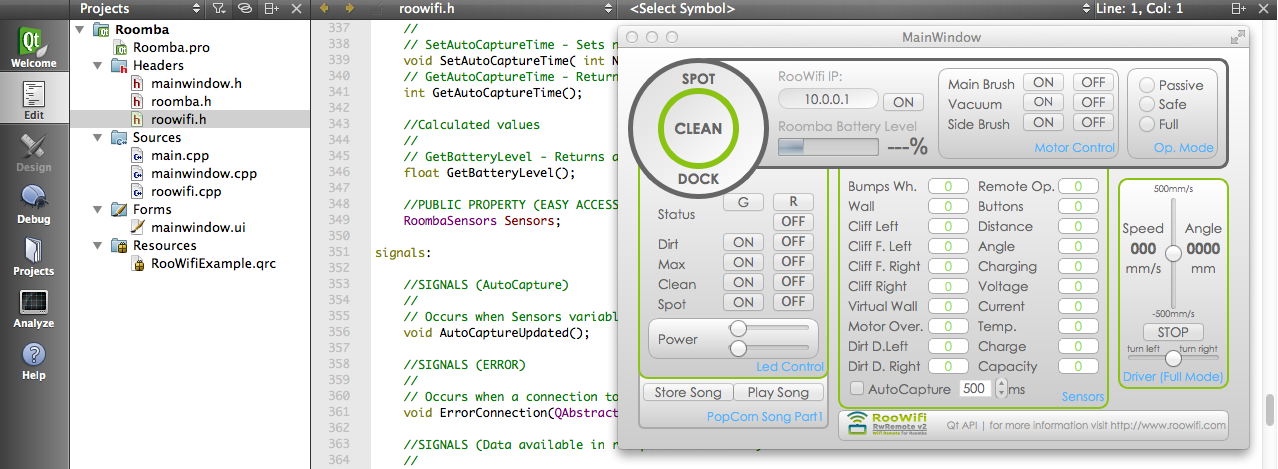


Figure 21 - Exemple de programme fait avec Qt

#### Kivy

Kivy est un ensemble de bibliothèque Python permettant de faire des interfaces simples, il est principalement utilisé pour faire des interfaces tactiles. Son objectif est d’être le plus simple d’utilisation possible. Bien qu’il soit encore récent et pas complètement terminé, Kivy reste très stable et fonctionnel. Il ne possède pas d’IDE officiel, il faudra donc tout paramétrer via le code.



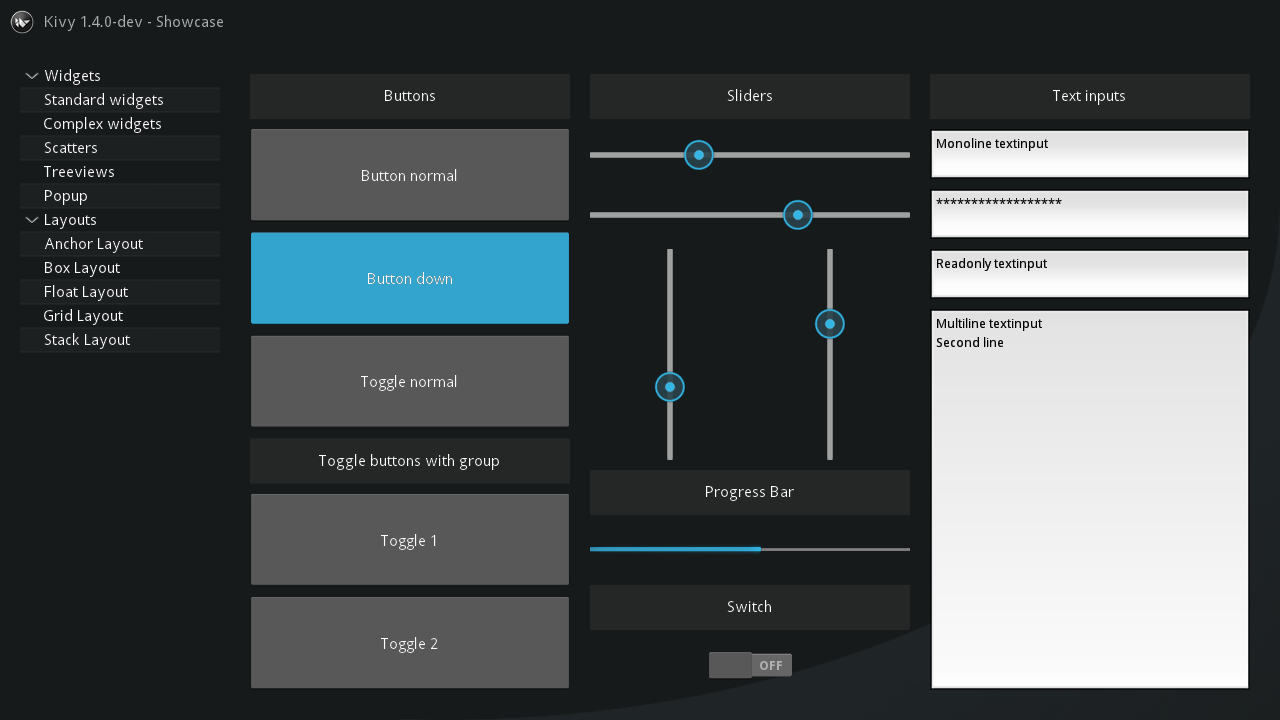


Figure 22 - Exemple de programme fait avec Kivy

Finalement, le choix s’est porté sur Kivy, pour son élégance et sa simplicité, et pour pouvoir apprendre quelque chose de nouveau qui peut être intéressant à utiliser dans le futur.

Kivy structure ces codes d’interfaces en deux parties : une partie en Python, et une partie dans le langage Kv qu’ils ont inventés. Le code python gère la partie programmation, et le code Kv gère tous les éléments graphiques.

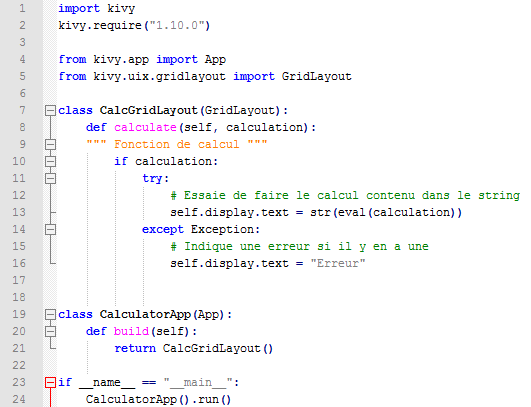


Figure 23 - Code Python d'une calculatrice sous Kivy

Le nom de l’application doit toujours se terminer par App, pour être utilisé correctement par Kivy. Le reste du code Python ne change pas énormément, mise à part que les objets n’ont pas le constructeur Python habituel. Les objets App ont la méthode build comme constructeur, et les autres widgets n’en ont pas. À la place, il faut utiliser des properties de Kivy :

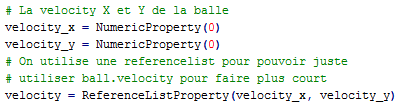


Figure 24 - Exemple de properties Kivy

Il faut donc utiliser une NumericProperty pour un int ou un float, un ListProperty pour un tableau, etc… À noter qu’il ne faut pas oublier d’importer les properties utilisées :



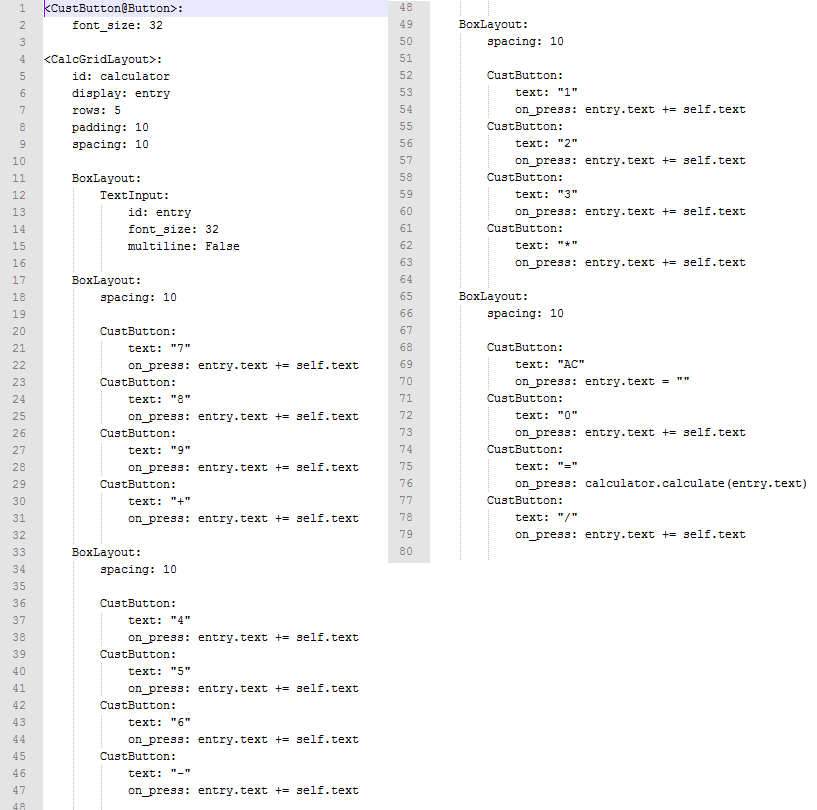


Figure 25 - Code Kv de la calculatrice

Le code Kv structure la mise en forme de la fenêtre. Dans cet exemple on a un layout en grille, qui contient tous les boutons de la calculatrice.  
Le fichier kv doit s’appeler de la même manière que l’objet application. Par exemple si l’application s’appelle **MyWindowApp**, il faudra appeler le fichier kv : **mywindow.kv**

Pour l’interface du robot, l’objectif était de faire une liste déroulante avec la possibilité de rajouter d’autres éléments à la liste à l’aide d’un bouton. Le framework n’étant pas maitrisé pour le moment, cette idée a était abandonnée.  
Finalement, l’interface est une simple grille, permettant de faire trois programmes, et de régler la durée d’un programme à l’aide d’une barre.

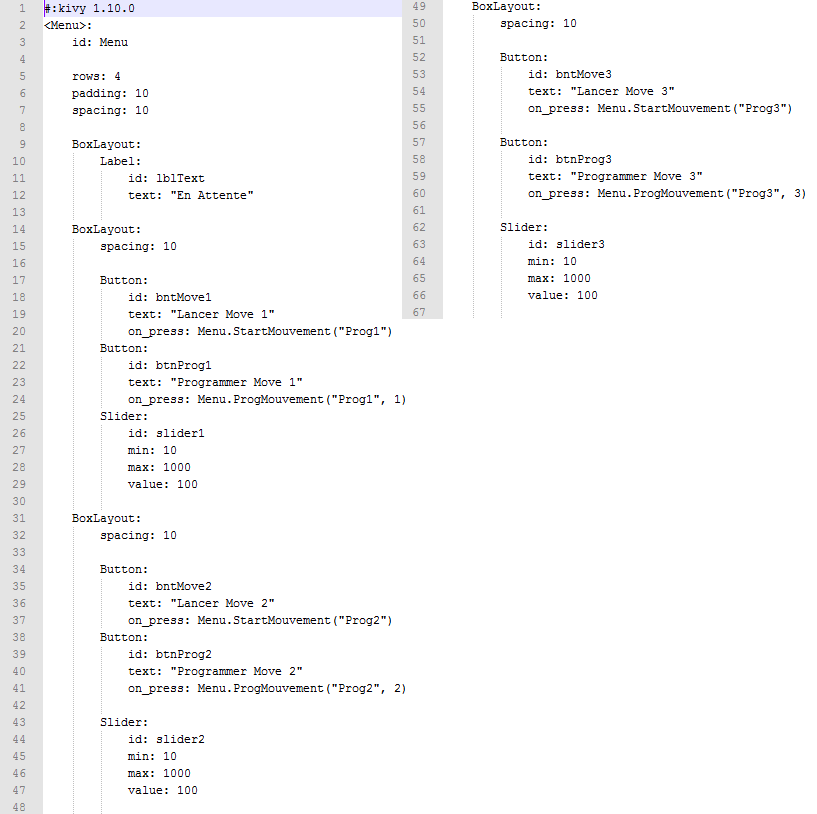


Figure 26 - Code Kv de l'interface

### Supports des moteurs

Pour accrocher les moteurs au torse, il faut un support. Pour réaliser ce support, il a été décidé d’utiliser l’imprimante 3D, afin d’avoir la forme souhaitable, et de pouvoir bien fixer ça au torse.  
Il faut donc que le point de fixation au profilé Kanya puisse s’enfile correctement dans le Kanya, pour cela il suffit de reprendre les cotations de Kanya.

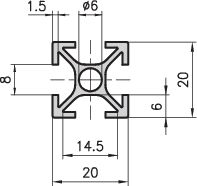


Figure 27 - Cotations d'un profilé 20x20 Kanya

La pièce 3D a été réalisée sous Inventor qui est un logiciel professionnel de conception 3D

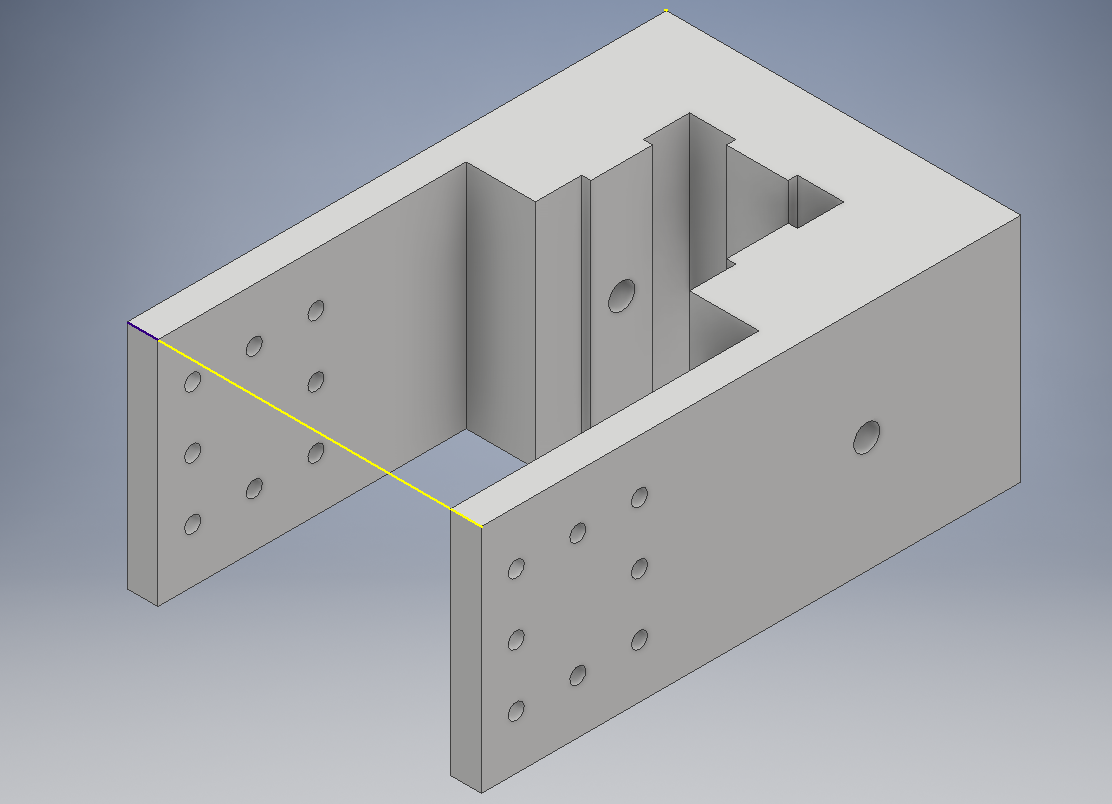


Figure 28 - Pièce 3D réalisée sous Inventor

### Montage final du robot

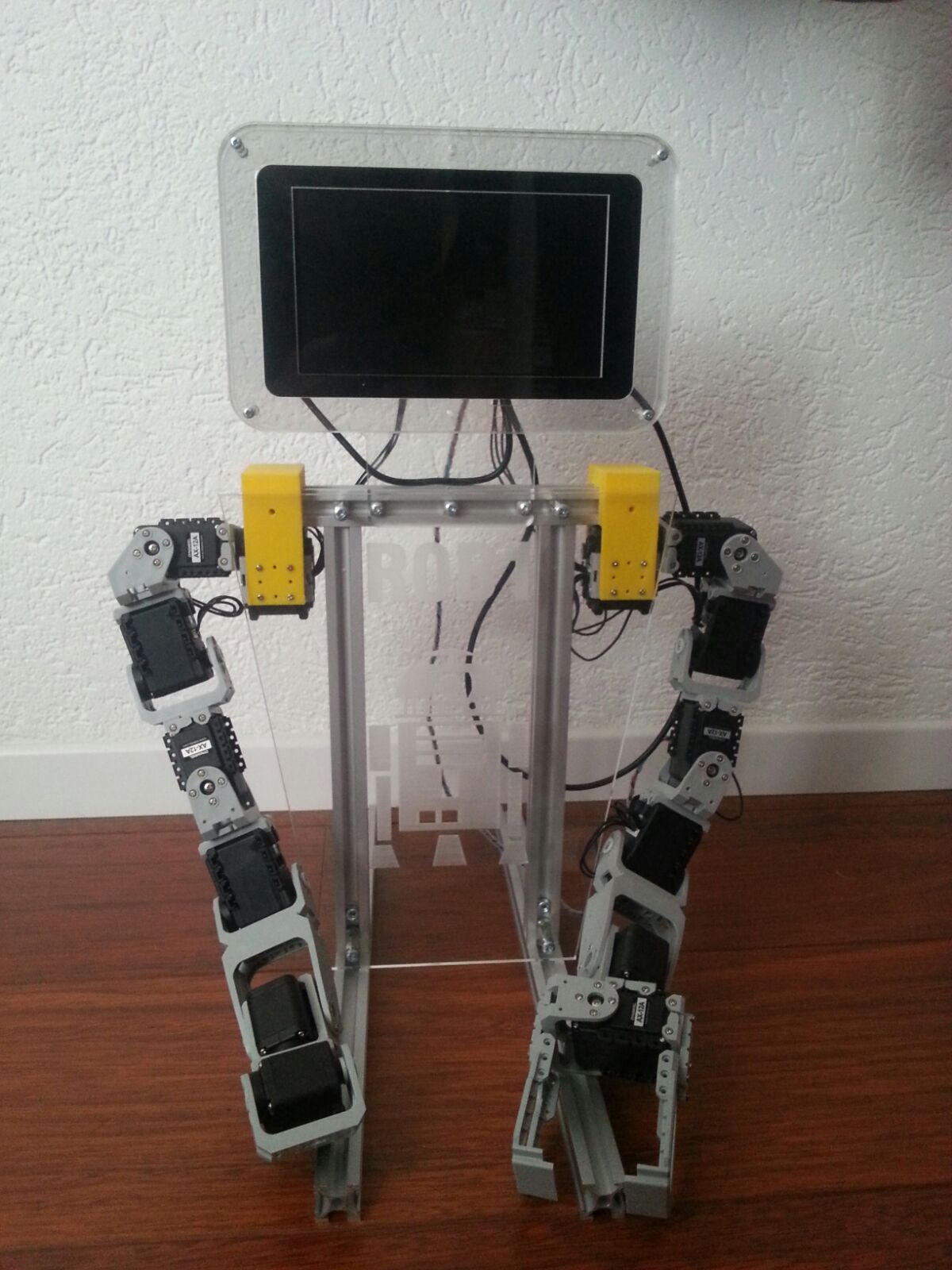


Figure 29 - Robot assemblé

## Programmation

Le programme a été séparé en trois partie principale, et en quatre fichiers pour rendre le code facilement utilisable, et pour pouvoir utiliser certains fichiers en dehors de ce projet facilement.

### Résumés des fichiers

#### axDriver.py

Ce fichier contient toutes les fonctions pour communiquer avec les moteurs AX-12A :

* sendData : Permet d'envoyer une instruction au moteur AX-12A sélectionné
* receiveData : Permet de recevoir les données de l'AX-12A sélectionné
* direction : Choisi la direction des données, soit rx, soit tx
* ping : Fonction servant à PING un moteur pour voir son état et son type d'erreur
* goToPosition : Bouge le servo a la position indiquée.
* setPosition : Ecrit la prochaine position dans le registre, mais le servo n'ira à la position que quand il recevra la commande ACTION.
* Action : Le servo lance les actions en attente
* getPosition : Fonction lisant la position du servo sélectionné
* setTorque : Permet d’activer ou désactiver le couple du moteur

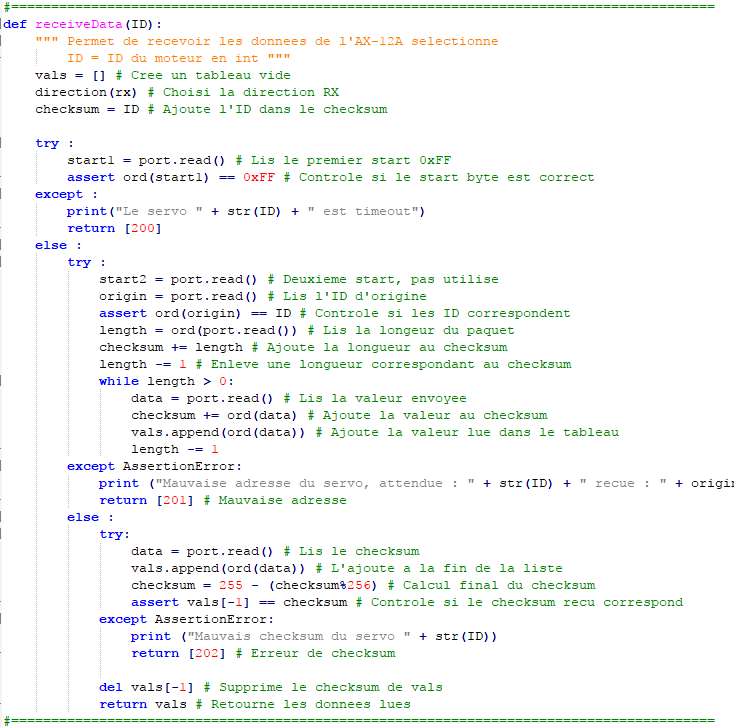


Figure 30 - Fonction receiveData

#### axFiles.py

Ce fichier contient toutes les fonctions gérants les fichiers de positions des moteurs :

* checkFile : Permet de check si le fichier existe déjà, afin d'informer l'utilisateur en cas de soucis
* renewFile : Permet de clear le contenu d'un fichier, ou de créer un nouveau fichier
* writeStep : Ecrit une étape avec toutes les positions des moteurs contenues dans le dictionnaire de positions
* writeFile : Ecrit un fichier en entier, plutôt que étape par étape
* readFile : Lis le fichier en entier, et retourne un tableau de dictionnaire contenants les positions des moteurs, chaque dictionnaire contient une étape, et le tableau contient les dictionnaires de chaque étapes.

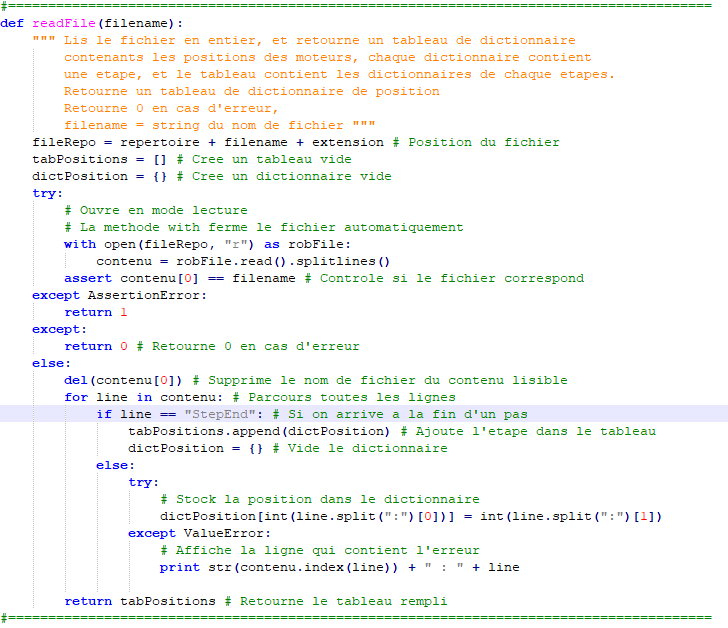


Figure 31 - Fonction readFile

#### axControl.py

Ce fichier permet d’utiliser les fonctions de axDriver plus facilement via l’interface tactile

* checkAllServo : Ping jusqu'à IDmax, et dit si un servo est sur cette ID
* progMode : Active ou désactive le couple de tous les servos
* getMouvementSteps : Récupère la position de tous les servo connectes
* saveFullMouvement : Enregistre un mouvement entier dans un fichier
* sendMouvementStep : Envoie une étape de mouvement
* doFullMouvement : Exécute un mouvement entier venant d'un fichier, étape par étape, avec un délai réglable entre chaque étape
* backToMiddlePos : Renvoie les servos à la position centrale

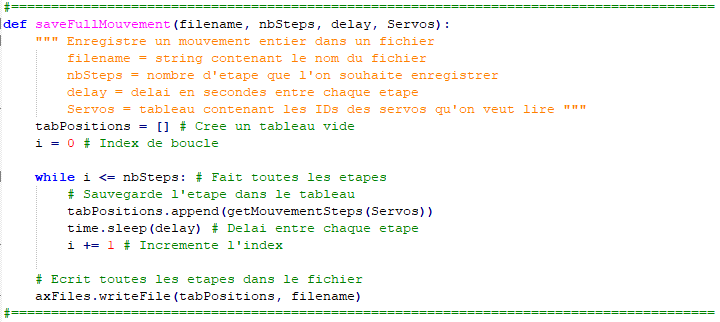


Figure 32 - Fonction saveFullMouvement

### Communication avec les moteurs

L’envoi de donnée aux servomoteurs se fait par paquet de 8 bits

C:\Users\Samy Francelet\AppData\Local\Microsoft\Windows\INetCache\Content.Word\76b567115bc6c93a941af3304bf48d90.png

Il faut d’abord envoyer les deux start bytes 0xFF, puis l’ID du servo concernés. Le prochain byte est la longueur des instructions, il s’agit de 2 + nombres de paramètres (les 2 viennent de INSTRUCTION et de CHECKSUM). Puis suivent l’instructions et les paramètres. Pour finir, le checksum doit être envoyé, c’est une variable qui permet de confirmer que tous les paramètres reçus sont corrects, si l’addition est fausse, le moteur ne tiendra pas compte de l’instruction. Le checksum se calcul en additionnant tous les bytes du paquet, sans les deux start bytes, et en envoyant que les 8 derniers bits.

Une fonction sendData communique avec les servos, elle prend en paramètre l’ID, l’instruction et un tableau contenant les paramètres. Elle s’occupe ensuite d’envoyer ces données aux moteurs avec le checksum.

La fonction receiveData prend en paramètre l’ID et permet de recevoir des données provenant des servomoteurs.

Ensuite plusieurs petite fonctions permettent de communiquer avec les servomoteurs plus facilement que de devoir retaper toutes les datas avec l’instructions comme : ping, goToPosition, setPosition, getPosition, action, setTorque

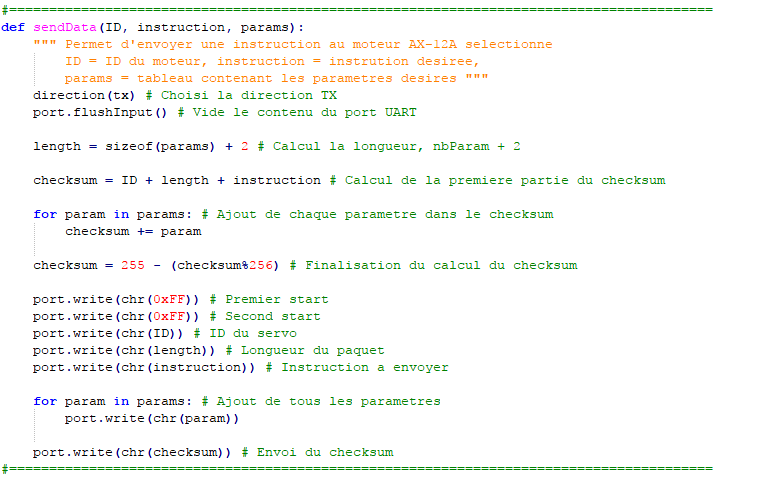
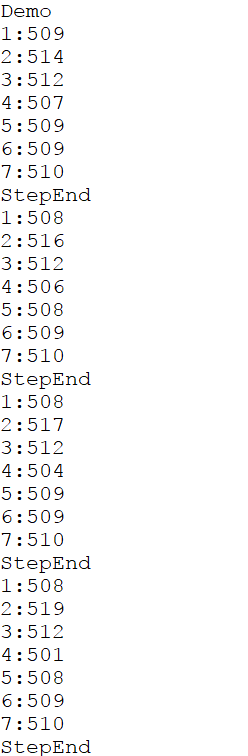


Figure 33 - Fonction sendData

### Gestionnaire de fichier

Pour enregistrer un mouvement entier, il faut créer des fichiers.  
Tout d’abord il faut définir un modèle de fichier pour ne pas se perdre.

Le fichier de mouvement doit pouvoir séparer chaque étape du mouvement, et indiquer les positions liées à chaque ID de servomoteur. Le format choisi est le suivant :



On a ainsi les étapes qui sont séparées, et la position liée à l’ID du servomoteur.

Maintenant que le format du fichier est défini, il faut de quoi lire et écrire ce type de fichier via le code. Le fichier axFiles.py contient toutes les fonctions nécessaires pour cela.  
readFile va permettre de lire un fichier et de retourné l’entièreté du contenu sous format d’un tableau de dictionnaire (qui est le format de mouvement utilisé par le code). Ce format est utilisé car : le dictionnaire est comme un tableau, à l’exception qu’au lieu d’avoir la case 0,1,2… on peut nommer les cases comme on veut. Exemple : frigo[‘’fruit1’’] = ‘’pomme’’, frigo[‘’congelateur’’] = ‘’crème glacée’’. On l’utilise pour pouvoir avoir des moteurs aux ID qui commencent un peu n’importe où, sans avoir des cases vides inutiles et désagréable à traité. Un dictionnaire contient une étape, il faut donc faire un tableau de dictionnaire pour faire un mouvement entier.

writeStep permet d’écrire un dictionnaire d’étape en entier :

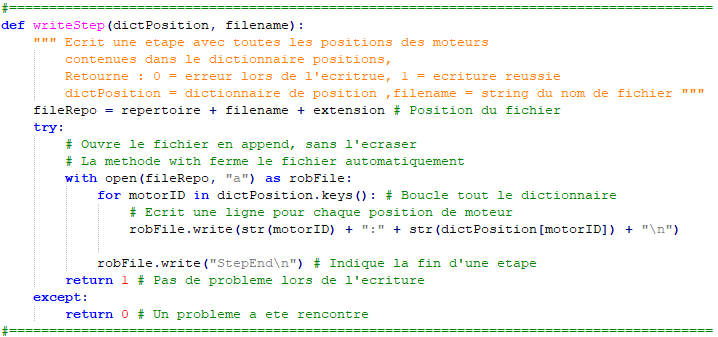


Figure 34 - Fonction writeStep

### Fonctions simplifiées

Pour pouvoir utiliser efficacement le driver avec l’interface, il faut des fonctions un peu plus simples à utiliser, comme : Récupérer un mouvement entier et l’enregistrer, désactiver le couple de tous les servos pour les bouger, lire un fichier de mouvement et l’exécuter.

Pour cela il y a le fichier axControl.py, il contient toutes les fonctions simples décrites au-dessus.

saveFullMouvement permet par exemple d’enregistrer un mouvement entier, et doFullMouvement exécute un mouvement entier contenu dans un fichier.

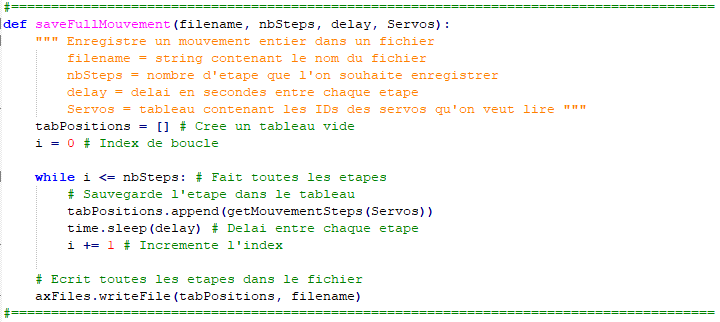


Figure 35 - Fonction saveFullMouvement

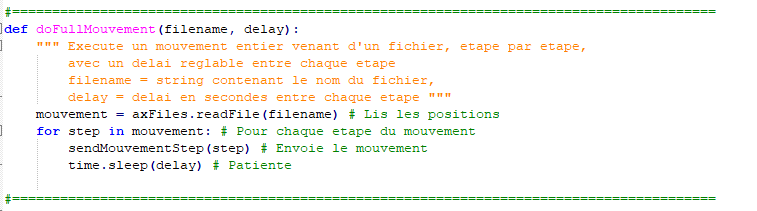
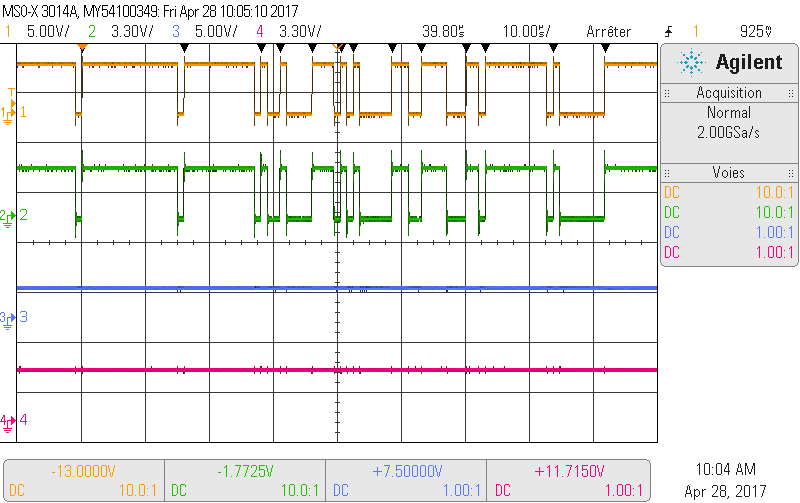


Figure 36 - Fonction doFullMouvement

# Résultats obtenus

## Problèmes rencontrés



Direction

RX

TX

Data

Figure 37 - Test d'envoi d'une trame dans le 74HC241

La pin de direction est à 1, le TX arrive bien à envoyer des données aux moteurs.

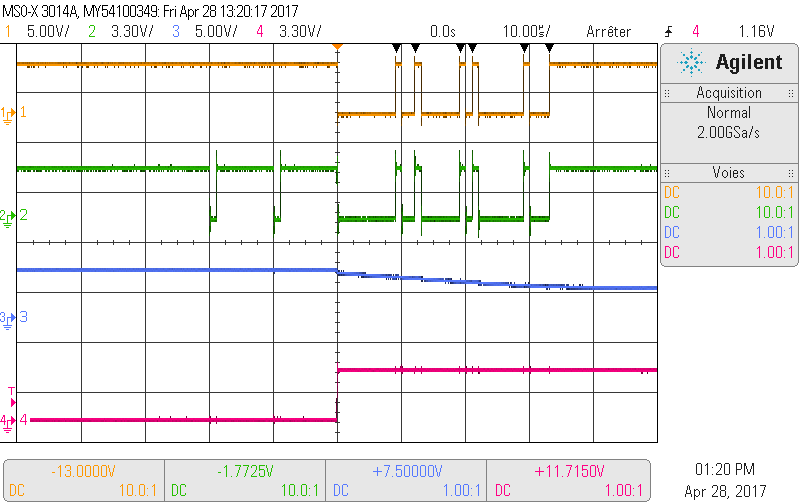


Figure 38 - Test du code driver de moteur

Pour une raison obscure, la pin de direction monte à 1 bien après qu’on lui ait demandé de monter dans le code.

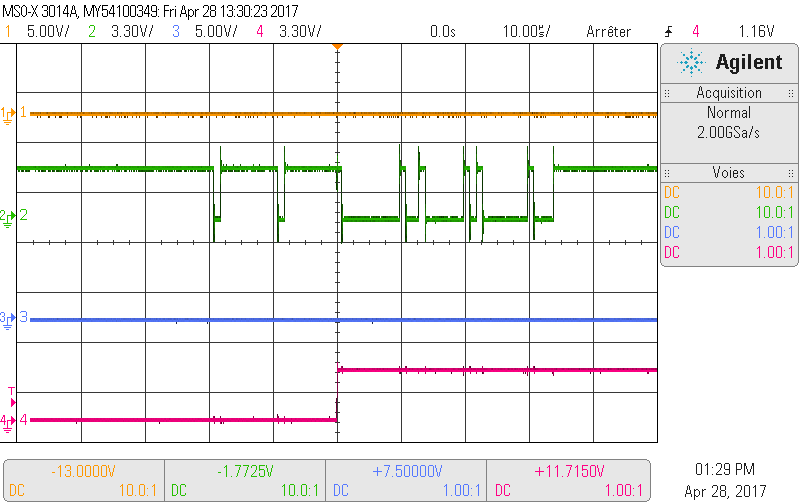


Figure 39 - Test du code driver sans la carte 74HC241

On remarque bien que le problème est software, car même sans le chip le problème subsiste.

Finalement, le problème était que depuis le driver, le paquet était envoyé d’un coup à l’UART qui se chargeait de cela tout seul pendant que le code continuait. Il a donc fallu changer et envoyer le paquet byte par byte.

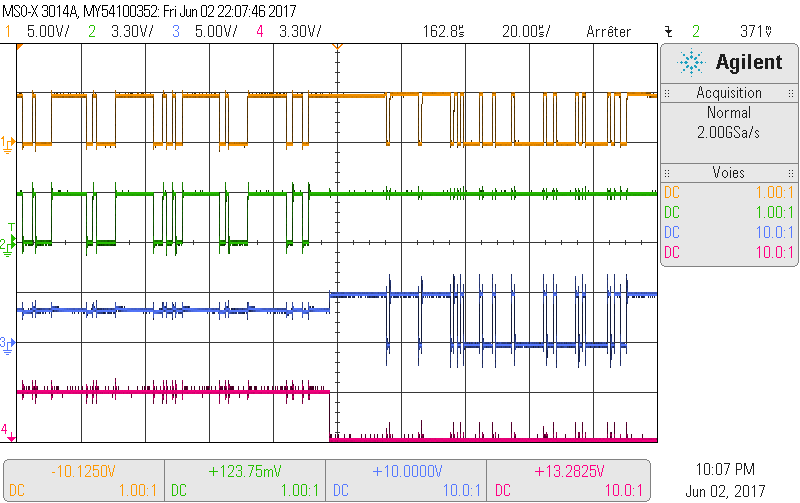


Figure 40 - Preuve du fonctionnement de la communication

## Mesures obtenues

### Alimentation

Du côté de l’alimentation, le 5V est propre à la sortie du Recom, et le courant fourni au Raspberry Pi et à l’écran est suffisant pour faire fonctionner le tout.  
Le Raspberry Pi 3 consomme 430mA et l’écran consomme 500mA

### Communication

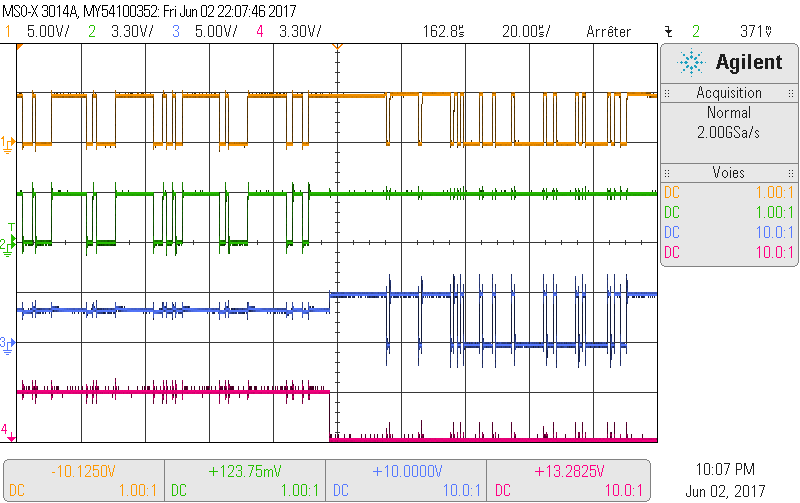


Figure 41 - Communication entre le Raspberry et les moteurs

La communication fonctionne correctement, le chip arrive bien à convertir le full en half duplex et arrive à suivre malgré un baudrate de 1Mbauds

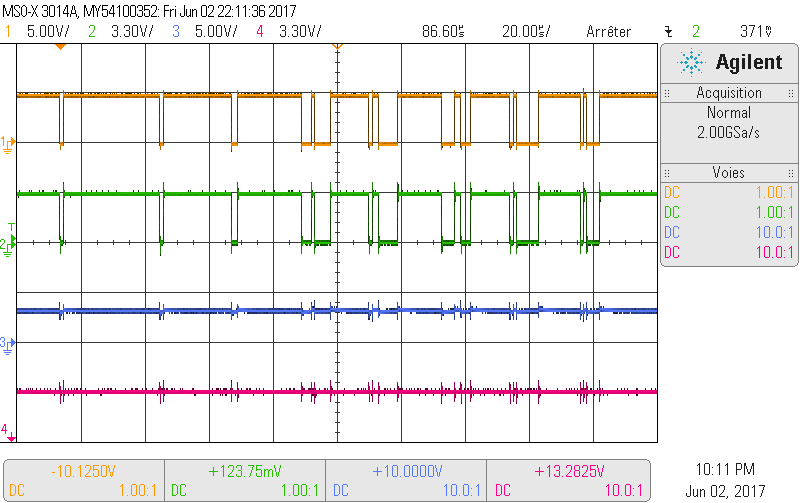


Figure 42 - Envoi d'un ping avec le Raspberry

## Analyse des résultats

Tous les composants fonctionnent parfaitement, le Raspberry arrive bien a communiqué avec les moteurs, et l’alimentation est suffisantes. Les servos répondent bien aux commandes et réagissent bien dans l’ensemble.

## Tableau récapitulatif des résultats

|  |  |  |
| --- | --- | --- |
| Objectif | Oui/Non | Remarque |
| Fonctionnement de l’alimentation | Oui |  |
| Fonctionnement de l’envoie de données | Oui |  |
| Fonctionnement de la réception de données | Oui |  |
| Fonctionnement de l’enregistrement d’un mouvement | Oui |  |
| Fonctionnement de la lecture d’un mouvement | Oui |  |
| Fonctionnement de l’interface graphique | Oui | Peut-être améliorée |

# Orientations futures et Conclusion

## Orientations futures

Le robot pourrait être amélioré en ajoutant un moyen de bouger le corps du robot, comme par exemple un système qui suivrait l’utilisateur. Il serait aussi possible d’intégrer la reconnaissance vocale de Google pour pouvoir contrôler et communiquer avec le robot par la voix. Ajouter des visages animés comme il était prévu au départ.  
On pourrait aussi reprendre le code et réaliser d’autre projet avec ces moteurs et le Raspberry.

## Conclusion

Le projet a été très intéressant dès le début, l’idée de pouvoir programmé des mouvements facilement ainsi était plaisante a réalisée. Les objectifs ont été atteints, à part celui des visages animés. Les délais étaient un peu courts mais le projet a pu être terminé dans les temps. La découverte de ces moteurs m’a permis d’entrevoir plein de possibilités à réaliser avec ceux-ci. J’ai acquis une certaine autonomie lors de ce projet, ce qui va beaucoup m’aider à l’avenir.

# Annexes

* Dossier de fabrication
* Liste des composants
* Datasheets intéressantes
* Rapports de mesures et protocole de mise en service
* Protocoles de tests
* Fichier des cadres du visage
* Fichier du torse
* Fichier côté de la pièce 3D
* Setup du Raspberry Pi 3
* Code

Francelet Samy :