Création d'un interface de contrôle pour le robot InMoov

Thomas Dargent

Rapport de Travaux d'Études et Recherche

Encadré par :
Youssouf Saidali

Université de Rouen Normandie
Soutenu le 29 mars 2019

Remerciements

Merci à *A. Luboya M'bingila* pour son aide dans la fabrication du robot. À *M. A. Boubtane* et *S. Kanku Mulumba* pour leurs participation au développement de la partie visuelle de l'interface. À eux trois pour leurs idées et conseils tout le long du projet.

Merci à *S. Anani* pour m'avoir confié le projet et partagé sa vision de celui ci. Merci aux *Francas de Seine Maritime* pour m'avoir donné les moyens nécessaires à l'accomplissement du projet.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

1.1.1 Le développement de la robotique humanoïde

La robotique est en règle général un sujet fascinant, mais s'il est un de ses sous-domaines qui fascine encore plus c'est la robotique humanoïde. La vision de ces machines nous ressemblant fait bien souvent naître des sentiments complexes chez l'observateur. Cette fascination a engendré très tôt de nombreux projets humanoïdes.

L'être humain traite beaucoup de choses de manières anthropomorphique : de l'animal à de simple formes géométrique qui bougent, nous attribuons naturellement des pensées et des sentiments à des choses qui n'en ont pas forcément (ou en tout cas pas de manière humaine). Il est donc naturel que la robotique anthropomorphique prennent une place importante dans le domaines. C'est d'autant plus important quand l'on étudie le domaine des interaction homme machines. Il semble que les robots humanoïdes soient mieux traités que les autres et on pourrait les percevoir comme intelligents. De plus, des mouvements humains sont plus facilement prévisibles, ainsi cet aspect pourrait faciliter l'usage des robots. Par exemple l'ajout d'un cou aide l'humain à percevoir ce que le robot regarde (Blanchard A. & Mebarki D. 2018).
Les robots humanoïde peuvent également donner des informations sur la manière dont l'homme perçois ce qui l'entoure. Penser les robots de manière anthropomorphique suscite des questions philosophiques et psychologiques importantes (Zlotowski J. et al. 2015).

En somme créer un être artificiel nous ressemblant est à la fois utile dans certaines situations pratiques où le robot doit interagir avec nous (assistance à la personne, accueil,

dialogue..), et pour mieux comprendre l'être humain lui même.

1.1.2 Le mouvement des bidouilleurs et l'accès au numérique

L'informatique est rendue de plus en plus accessible et des robots programmables se retrouvent désormais sous les sapins de Noël. De manière générale on note que l'électronique et la programmation se sont démocratisées et sont désormais à la portée de tout amateurs. Cela a contribué à la naissance du mouvement des Makers.
Au cœur de ce mouvement se trouve l'impression 3D et à fortiori l'impression par dépôt de matière fondue (Fused Deposition Modeling). Ce processus de démocratisation a été porté par des particuliers comme des entreprises. En 2005, *Adrian Bowyer* lança le projet **RepRap** visant à créer des machines d'imprimante 3D open source, notamment en imprimant leurs pièces en 3D. De plus, des sociétés comme **Makerbot** ont très vite fournis des imprimantes montables soi-même à prix relativement faibles. Ces prix n'ont fait que baissé depuis, par exemple il est désormais possible d'avoir une imprimante bas de gamme pour moins de 200€.
De la même manière l'électronique devient de moins en moins chère et des marques tentent de la rendre accessible à tous. On notera par exemple **Adafruit** ou **Sparkfun**, qui vendent des produits facilement accessibles. Ces dernières années ont aussi été le berceau de nombreux progrès en pédagogie. **Scratch** (MIT Media Lab. 2006) — un langage de programmation graphique — est enseigné en cours de mathématiques de la primaire jusqu'au collège. Après l'**Arduino** (Banzi M. 2003), **Microbit** (Micro :bit Educational Foundation 2015) a rendu encore plus accessible la programmation embarquée en offrant une carte programmable graphiquement, dotée d'une matrice de DEL et de quelques capteurs. En somme des utilisations de plus en plus pointues et complexes de la technologie sont accessibles à de plus en plus de personnes.

1.1.2.1 InMoov

C'est dans ce contexte que *Gaël Langevin* créé **InMoov** (Langevin G. 2012). Les pièces du robot sont disponibles gratuitement sur internet ce qui fait de lui «Le premier robot humanoïde open source» selon son auteur. Il a été conçu pour être imprimé sur des imprimantes d'amateurs d'un format minimal de 12x12x12cm. Accompagné d'un site fournissant des explications sur son montage, InMoov a vite été repris par de nombreuses personnes.
Le robot est contrôlé par le logiciel **MyRobotLab**. Développé en Java ce dernier permet de contrôler chaque moteur indépendamment à distance. Il comporte la possibilité d'ajou-

ter des fonctionnalités de synthèse vocale, reconnaissance vocale, vision par ordinateur et programmation de protocole de mouvement.
Après de nombreux essais un problème nous est apparu : l'usage du logiciel est complexe, sa documentation éparse et sa modularité le rend compliqué à installer/paramétrer. Il nous semblait que le robot pouvait être rendu plus accessible de ce côté-ci.

1.1.2.2 Les cartes *Raspberry Pi*

En 2006, la carte **Raspberry Pi** voit le jour. Il s'agit de nano-ordinateurs (taille d'une carte de crédit) open source vendus à bas prix. Destiné à encourager l'apprentissage de la programmation, ces cartes sont aujourd'hui utilisées dans plusieurs champs et permettent d'utiliser des ordinateurs sous Linux pour 30€ maximum. Elles sont ainsi idéales pour servir de petits serveurs.
L'un des atouts majeurs de la carte est qu'elle donne accès à des ports GPIO (General Purpose Input/Output). Ceux-ci permettent de communiquer électriquement avec l'ordinateur sans passer par des intermédiaires.

1.2 L'objectif pédagogique

Ce Travail d'Études et de Recherche vient s'insérer dans ce contexte. Fais en partenariat avec l'association des **Francas de Seine Maritime**, l'objectif du projet est d'enrichir leurs dispositifs pédagogiques d'une plateforme de développement de robot humanoïde et de construire le robot *InMoov* comme exemple. On chercheras à offrir une manière simple et peu coûteuse de contrôler *InMoov* pour les non-initiés, les jeunes programmeurs et mêmes les plus confirmés. Pour cela on vise à leur mettre à disposition :

- Un moyen de faire bouger les servo-moteurs du robot ;
- Un système de développement de routine permettant de **contrôler le robot** en *Python* et en programmation graphique (type *Scratch*) ;
- Une **interface** simple permettant d'accéder a ces routines et de les lancer ;
- Un moyen d'interagir avec les **capteurs embarqués** sur le robot (caméras, microphone, *Kinect*..).

Cela nécessite évidemment un système faisant la liaison entre cette interface et le robot physique. Il faut donc réfléchir à la manière de faire un système embarqué relié aux servo-moteurs et aux capteurs.

La problématique principale est donc de savoir si l'on peut rendre accessible à tous la robotique humanoïde. Peut-on leur permettre d'expérimenter dans les différents champs où elle est utilisée ?

1.3 Plan

On structurera ce rapport en quatre parties :

Premièrement, dans le **chapitre 2**, on présentera l'état de l'art en robot pédagogique et en robotique humanoïde. Cela nous permettra de situer le projet dans son contexte, et de mettre en avant ses qualités et ses lacunes.

Deuxièmement, dans le **chapitre 3**, on détaillera les étapes de réalisation et les problèmes majeurs rencontrés tout le long du développement du projet. On présentera les choix faits en les justifiants.
Dans le **chapitre 4**, on fera un état des lieux du projet, son fonctionnement et on présentera l'interface.

Enfin on conclura au **chapitre 5** sur ce qui a été réussi lors du projet, ce qui l'a moins été, et on abordera les pistes de développement futur.

Chapitre 2

État de l'art et proposition

Dans ce chapitre on présentera ce qui se fait de mieux au niveau de la robotique humanoïde, et en contrôle/programmation de robot. Cela nous permettra à la fois de voir les possibilités des deux domaines et d'y comparer notre projet.

2.1 Robotique humanoïde

Il y a une multitude de robots existants dans ce domaine et il est dur de les classer (sur quels critères ?). On fera donc un bref comparatif de robots que l'on a vu cités beaucoup de fois ou qui nous paraissent particulièrement intéressants.
Comme on l'a vu brièvement dans l'introduction ces robots ont des buts divers. Parmi ceux-ci on peut retrouver celui de confirmer l'hypothèse que le corps influence la manière d'acquérir du savoir. C'est l'hypothèse émise par ECCE1 (Gravato Marques H. et al. 2010) et iCub. L'idée est qu'un robot ne pourra jamais apprendre totalement à interagir avec son environnement avec un corps minimaliste comme on le fait classiquement. Ces chercheurs tendent donc vers des robots biomimétiques, en supposant que si notre intelligence s'est développé c'est au moins en partie grâce a notre corps. A noter que le projet ECCE1 est relativement vieux, mais que les suites annoncées sont introuvables. Toujours de manière biomimétique, mais avec une approche presque symétrique, on trouve Kengoro et Kenshiro (Asano Yuki et al. 2017). Ceux-ci serviraient à mieux comprendre la manière dont notre corps fonctionne en recréant un corps artificiel.

Parmi des robots moins orienté recherche on a Asimo qui sans avoir de but véritablement défini, servira probablement dans un contexte d'accueil commercial ou social. Nao a ce but-là également, mais est plus généralement une plateforme robotique accessible aux

universités, lycées et diverses structures. Poppy (Lapeyre M. 2014) a également cherché à créer une plateforme robotique libre et accessible pour des études, de la recherche mais également pour l'art. InMoov lui ne semble pas avoir de but défini.

Nom	Dof	jambes fonctionnelles	coûts estimé	But	Open source
ECCE1	?	Non	?	Connaissance du monde par le corps	Non
iCub	53	oui	250 000	Connaissance du monde par le corps	oui
Kengor	114	oui	?	Meilleur compréhension des mouvements humains	Non
Poppy	25	oui	7 500	Plateforme de robotique abordable	oui
Asimo	57	oui	?	Robot commercial, accueil	Non
Nao	14-25	oui	9000	Plateforme educative, accueil, robot commercial	Non
inMoov	31	non	1500	?	Oui

Ce tableau montre bien l'intérêt principal, et la cible de InMoov : les amateurs. Le robots est relativement peu coûteux et open source.

2.2 Programmation de robot

De la même manière on parlera brièvement de toute les façons possible de contrôler un robot. On se concentrera plus accessibles.

BlocklyArduino est un logiciel web basé sur Blockly permettant de créer un code Arduino à partir de blocs graphiques. Edublocks est similaire, mais vise à recréer un environnement python de manière graphique. Scratch3 dispose de fonctionnalité de contrôle de moteurs etc.. via à la MicroBit entre autre. Ce dernier n'est pas une encore une solution idéale pour cela. Enfin MyRobotLab est un logiciel fait pour contrôler tout type de robot sur un ordinateur, dispose de pleins de capteurs et de fonctionnalités.

Nom	Création de procédure	Contrôle direct de moteurs	Difficulté d'utilisation	Open Source	Capteurs exploitables
Blockly Arduino	Oui	Non	Plutôt Simple	Oui	Capteurs simples
Edublocks	Oui	Non	Simple	Oui	Variés
Scratch3	Oui	Non	Simple	Oui	Capteurs simples
MyRobot Lab	Oui mais non natif	Oui	Complexe	Oui	Variés
TER	Oui	Non	Simple	A venir	Variés

On voit que l'intérêt principal du TER est qu'il propose une manière simple de contrôler des robots équipés de capteurs complexes. Contrairement a EduBlocks il n'a pas pour objectif de rendre graphique la programmation Python, mais bien de rendre cela le plus simple possible.

Chapitre 3

Réalisation

3.1 Choix technologiques et commencement du projet

3.1.1 Électronique

Lors de la conception du projet, l'idée était de se baser sur deux technologies, la **Pyboard** — une carte de prototypage similaire à l'Arduino mais programmables en Python — et la **Raspberry Pi** — un nano-ordinateur.
Très vite on a décidé de se baser sur les technologies du web pour faire l'interface. Ainsi le robot est vraiment embarqué, puisqu'au lieu de mettre un écran dans son dos ou de le relier en série à un ordinateur, il suffit de se connecter au robot sur un ordinateur externe. Dans ce cas la Raspberry sert à la fois de serveur — elle héberge l'interface en locale — et de système de contrôle — elle envoie les ordres appropriés aux composants qui lui sont reliés. On souhaitait utiliser la Pyboard comme esclave qui exécuterait les ordres liés aux actionneurs (Figure 3.1). Cela présentait un avantage majeur : les procédures auraient pu être exécutées directement dessus, limitant les problèmes de sécurité qui surviennent quand on laisse les visiteurs écrire des programmes. De plus, les procédures n'encombreraient pas la Rasberry Pi le temps de leurs exécution.

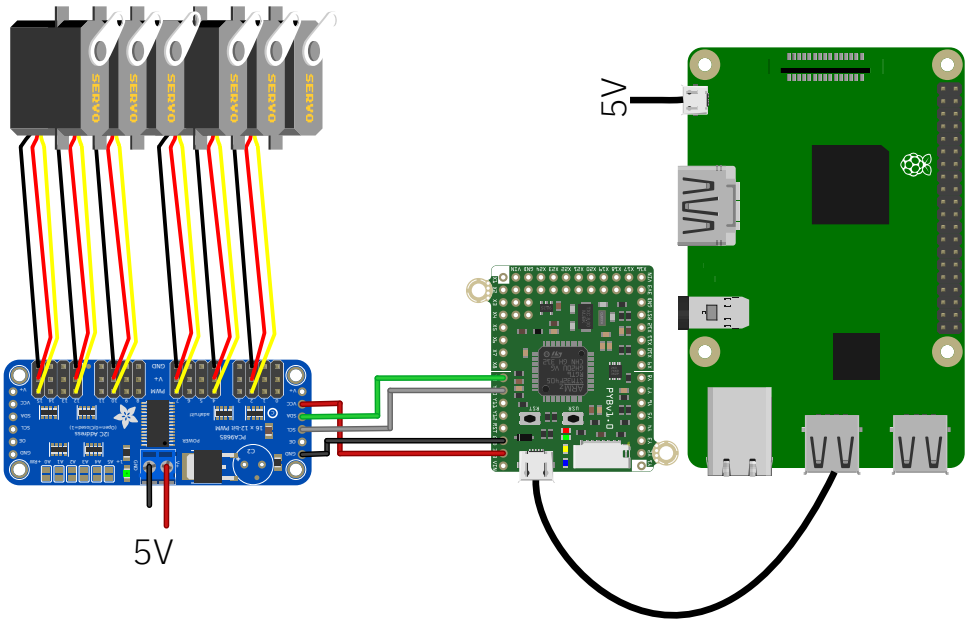

FIG. 3.1: Circuit permettant de contrôler les servomoteurs par le biais d'une Pyboard.

Toutefois, cette idée manquait de clairvoyance. En effet le robot étant équipé de capteurs et ceux-ci étant trop complexes pour un microcontrôleur (Webcam, Kinect..), on ne peut se contenter d'envoyer le programme sur la carte. De plus, certaines opérations destinées à être utilisées par le robot nécessitent une certaine puissance de calcul (reconnaissance faciale, traitement d'image, synthèse vocale..). Tout cela doit donc se faire sur la Raspberry Pi et nous n'avons personnellement trouvé que trois solutions :

- Système de communication en série avec le microcontrôleurs permettant de transmettre et recevoir les informations pertinentes
- Analyser le programme avant son exécution et envoyer au microcontrôleur uniquement la partie le concernant.
- Exécuter le programme directement sur la Raspberry pi.

Étant donné la complexité des deux premières, on s'est tourné vers la dernière (Figure 3.2). On a donc décidé d'exécuter la totalité des programmes sur la Raspberry Pi.

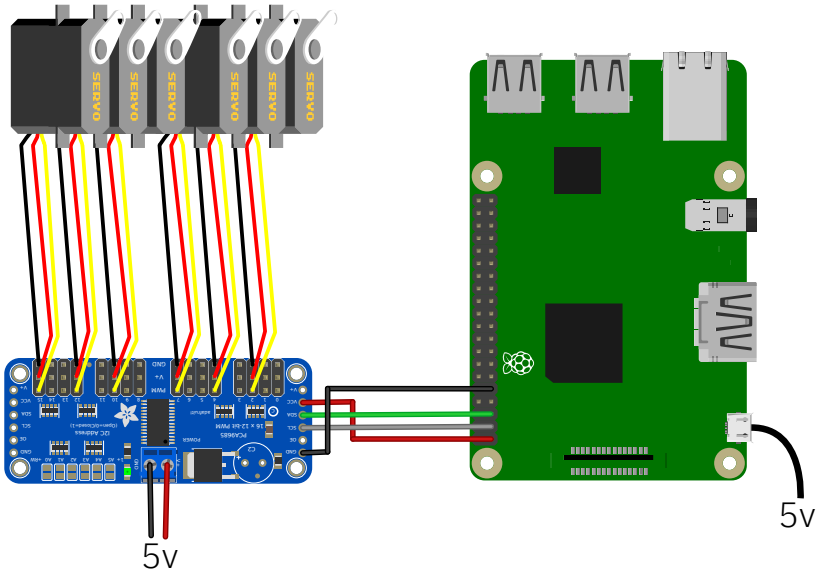

FIG. 3.2: Circuit final du système (ne sont pas inclus : les capteurs, les enceintes et autre appareils pouvant être branchés sur la Raspberry Pi).

3.1.2 LOGICIEL

Nous nous sommes tournés vers plusieurs outils pour faire tourner l'interface, d'abord attiré par **Flask** pour sa légèreté, on s'est ensuite tourné vers **Django** pour sa puissance et sa simplicité d'utilisation. Django est un outil permettant de servir des pages internet dynamiques à l'utilisateur. Il nous permet à la fois d'afficher des informations contenues dans la base de données et de traiter les requêtes de l'utilisateur en Python. Cela évite de passer par des intermédiaires par exemple pour lancer les scripts.

Très vite on a pu mettre en place le système de procédures : L'utilisateur créer un programme Python permettant de contrôler le robot, qui était à cette étape du développement représenté par des DEL. Il lui donne un nom, éventuellement une brève description, et la totalité des script est affiché sur la page d'accueil. On a ensuite permis à l'utilisateur d'exécuter les procédures. Celles-ci sont stockées sous forme de fichier que l'on envoyait au début par le REPL de la Pyboard, mais sont maintenant pré-compilés et exécutés sur place. S'étant concentré sur un prototype de l'interface de contrôle avant de faire l'électronique, nos premiers scripts se contentait d'allumer des DELs. Lorsque le choix d'utiliser la carte PCA9685 pour contrôler les servo-moteurs fut pris, il fallut découvrir comment s'en servir. *Adafruit* propose une librairie bas niveau permettant de les contrôler en Python, ainsi on avait ce genre de programme :

```
import Adafruit_PCA9685

# Initialise la PCA9685 avec l'adresse par défaut (0x40).
pwm = Adafruit_PCA9685.PCA9685()

pwm.set_pwm(0, 0, 150) # Longueur de pulsation minimale sur 4096
```

Le but du projet est de rendre accessible à tout type d'utilisateur la programmation. Dès lors il était évident que d'obliger les utilisateurs à rentrer une largeur de pulsation pour controler le moteur n'était pas idéal. On a donc fait le choix de faire des «sur-librairies» simplifiant la programmation. Par exemple :

```
import movement

move_servo(3, 0x40, 100) # Bouge le moteurs n°3 de la carte 0x64
    a 100% de son amplitude.
```

Toutefois pour le rendre accessible au plus jeunes il fallait rajouter un système de programmation graphique (à la Scratch). **Blockly** a donc été utilisé. Il s'agit d'une librairie **Javascript** de *Google* permettant de créer des environnements de programmation graphique exportable en Javascript ou en Python par exemple. Ainsi on créer un environnement Blockly sur les pages d'édition de scripts, les utilisateurs peuvent placer les blocs donnés pour faire le programme souhaité, puis l'on exporte ce code en Python.
Ensuite la procédure est la même que pour les script Python standards. Toutefois il faut aussi garder une trace de fichier XML représentant l'état de l'environnement Blockly à l'enregistrement (la conversion Python -> Blockly étant très complexe et bien au delà du cadre du projet).
La librairie contient déjà beaucoup de bloc standards (Calcul, traitement de chaîne de caractères, boucles, conditions, logique..) mais permet l'ajout de blocs. On a donc pu créer des blocs pour le projet définis en deux fichiers :

- l'un définis leur représentation et leur logique au sein de l'éditeur : type de bloc, couleur, nom, champs, type de sortie et d'entrées..
- l'autre est un générateur : il sert a créer le code Python correspondant. Pour garder un code de sortie lisible (qui pourrait être lu dans des versions futurs), le code crée est fait en utilisant les librairies du projet (présentées au chapitre 4).

À ce moment le cas de base était remplis : Le robot bougeais et était programmables à distance de manières plutôt simple.

3.2 Perfectionnement

3.2.1 L'anatomie d'un robot

Pour permettre un code plus clair et éviter aux utilisateurs d'avoir à retenir la position de leurs moteurs sur les cartes, nous avons ajouté des objets `Part`. Les `Part` sont des objets fournis dans la librairie `movement` encapsulant les informations nécessaires à l'utilisation des servos de chaque carte. Les parties du corps sont représentées par un objet éponyme, et les parties symétriques sont représentées par un dictionnaire éponyme contenant les objets voulus. On les utiliseras ainsi :

```
1 import movement
2
3 move_part(Thumb['left'], 50) # Plis le pouce gauche à moitié
4 move_part(Jaw, 100) # Ouvre la bouche au maximum
```

L'ajout de ces objets nécessita la construction d'un fichier de configuration permettant de définir les attributs sus-cités et l'étendu de modulation de possible. De plus on peut y définir plusieurs axes par partie. Cette possibilités est source d'une grande indécision. Prenons l'exemple de plusieurs parties d'InMoov :
Les yeux se déplacent sur deux axes : horizontal et vertical, si InMoov nous regarde. Maintenant s'il tourne la tête à droite les yeux se déplacent de haut en bas toujours mais aussi de l'avant vers l'arrière (sa gauche et sa droite deviennent notre avant/arrière). L'épaule d'InMoov à 3 «DoF» (axes de rotation) : de face on à haut/bas, avant/arrière, interne/externe. Comment nommer tout ses axes et surtout *comment les utiliser dans un code simple* ?

Pour l'utilisation on a choisi d'abord de favoriser certains axes dans la configuration. En effet `move_part(Eyes, 50)` se doit de rester utilisables de la même manière que si l'on faisait `move_part(Jaw, 50)`. Premièrement par soucis de simplicité et de cohérence, et ensuite car cela assure que même si le robot n'a qu'un seul axe de déplacement des yeux, le code fonctionne quand même. Pour cela on traite les axes comme les éléments d'une liste dans la configuration :

```
1 #...
2 'eyes': [Servo(0x40, 0, 1, 300, 2100),
3              Servo(0x40, 1, 2, 300, 2100)],
4 #...
```

ainsi on peut choisir de mouvoir des parties complexes comme on le ferait pour des

parties plus simples. Quant à l'utilisation du deuxième axe on peut y accéder en utilisant `move_part(Eyes, 50, n)` où n est le numéro de l'axe dans la liste de la configuration. C'est loin d'être idéal malheureusement mais c'est la solution la plus simple pour le moment.

3.2.2 Site dynamique

On a ensuite ajouté un ensemble de fonctionnalités rendant le site plus agréable à utiliser. Par exemple la possibilité de stopper un script qui a été lancé. Pour cela on a gardé en mémoire le thread dans lequel le script a été lancé, et fourni une API donnant le nom du script exécuté. Chaque page effectue une requête à cette adresse à intervalle régulier pour s'informer du lancement de script et ouvre un encart permettant de le stopper quand elle le détecte. De plus cela nous a amené à stopper le script en cours au lancement de chaque script. Cela évite les conflits éventuels.
On a ajouté la possibilité de modifier le fichier de configuration directement dans le site, et non juste en éditant le fichier avec un éditeur de texte. Un bouton pour supprimer les scripts a été ajouté sur la page d'accueil.

3.2.3 Autre fonctionnalités

Le contrôle de servo-moteur est primordial, mais InMoov dispose également d'une enceinte et de capteurs variés. On détailleras maintenant comment on a pu en tirer partie.

3.2.3.1 Synthèse vocale

La synthese vocale se fait avec MaryTTS. MaryTTS est une plateforme open-source, multilingue de synthèse vocale écrite en Java (Multimodal Speech Processing Group s. d.). Elle a été choisie car elle est open source, et aussi car ses voix françaises sont de très bonne qualités.
On fait tourner la plateforme en parallèle de la notre et communique par leur API en *Python*. Cela nous permet de récupérer un fichier wave correspondant a un texte fournis, mais également d'obtenir la liste des phonèmes de ce texte, et le temps où il sont prononcés dans ce fichier. On peut ainsi jouer le son et tenter de faire bouger la mâchoire en fonction. Si on a tenté de faire une synchronisation des lèvres avec le son, rien n'y fait ce n'est pas convaincant. Toutefois voici ce qu'on a choisi de faire : Si le phonème est une *consonne bilabiale*, alors on ferme la mâchoire. Si le phonème est une consonne on l'ouvre de manière aléatoire. On n'exécute un mouvement que pour les phonèmes longs (pour ne pas surcharger

le servo-moteur).
Cela ne rends pas bien pour une raison : On a fait fis des règles linguistiques élémentaires. En effet Le robot n'ayant pas de lèvre et la linguistiques décrivant les sons selon la position des lèvres, il est complexe de savoir que faire. Une possibilité serait d'avoir plusieurs exemples visuel de personne prononçant tout les sons possibles et d'en tirer le degré d'ouverture de la mâchoire. À partir de la on pourrait associé à chaque phonème un degrés d'ouverture et le robot n'aurait qu'à se référer à ce dictionnaire. C'est une vision simpliste mais qui pourrait donner des résultats acceptables.

3.2.3.2 Reconnaissance vocale

Capteurs primordial pour une interaction complète avec le robot, le micro d'InMoov n'a pourtant pas d'emplacement défini. L'anthropomorphisme nous dicterait de la placer sur les cotés de sa tête, toutefois l'emplacement est destiné à être utilisé par les enceintes, et le bruit, parfois important, provenant des mouvements de mâchoire rendrait son utilisation compliqué. Gael Langevin semble favorisé l'utilisation d'un casque micro qu'il porte lors de convention. Cette solution nuis, selon nous, à l'aspect embarqué du robot. L'endroit où il serait le moins affecté par le bruits des moteurs est le torse, mais cela demanderait un sérieuse re-modélisation des pièces.

Pour reconnaître les voix on à utilisé la librairie Python Speech Recognition (Zhang 2017), celle-ci nous permet simplement d'utiliser plusieurs plateforme de reconnaissance vocale. Celle que nous souhaitions favoriser est CMU Sphinx, toutefois sa complexité de configuration pour la reconnaissance de voix françaises nous a fait nous tourner vers Google. Google fournis une API de reconnaissance vocale gratuite pour beaucoup de langages. Le problème avec celle-ci est que la version gratuite est limité (50 requêtes par jour) et lentes.

3.2.3.3 Exploitation de la vidéo

Les caméras d'InMoov sont placés dans chacun de ses yeux. L'objectif final de ces capteurs est bien entendu de permettre la reconnaissance de formes (visages, objets), et un déplacement autonome. Toutefois ce sont des taches complexes, et nous avons donc souhaités simplifié le problèmes en réalisant d'abord une fonction de reconnaissance de couleur. L'idée était que le robot pourrait réagir à des signaux colorés, par exemple un carton rouge. Avec le recul c'était une très mauvaise idée.
On souhaitait pouvoir comparer une couleur avec avec celles de l'image. A l'aide de deux paramètres, la tolérance (seuil à laquelle une couleur est considérée équivalente) et la pré-

cision (nombre de subdivision de l'image en rectangle de couleur moyenne), on espérait obtenir des résultat exploitable. En réalité les webcams modifiant sévèrement les couleurs et la distance euclidienne entre deux couleurs ($\sqrt{\Delta R^2 + \Delta G^2 + \Delta B^2}$) n'est pas une bonne mesure de la distance perçue par l'Homme. Il semble que pondérer la somme donne une meilleur approximation de celle ci (Compuphase s. d.) :

$$\sqrt{2 \times \Delta R^2 + 4 \times \Delta G^2 + 3 \times \Delta B^2}$$

Toutefois nos tests n'ont pas indiqué une amélioration en utilisant une ou l'autre formule.

Sur conseil de *C. Petitjean* on examina également la possibilité d'utiliser l'espace HSV et un histogramme. Comparer la teinte seule ne donna pas de bon résultats, et malgré de nombreux test de pondération des trois coefficient cela ne sembla pas nous avancer. Le test de présence d'une couleur majoritaire dans l'histogramme rendit l'algorithme bien plus rapide puisque nous n'avions pas à effectuer une moyenne sur un nombre de subdivision de l'image. Toutefois un algorithme inefficace mais rapide reste inefficace.

Devant la complexité du problème on a donc décidé d'ignorer cette fonctionnalité.

Chapitre 4

Résultats

Ici on présentera les résultats obtenus au terme du TER. On décrira d'abord l'avancée sur le robot puis sur le logiciel.

4.1 Le robot

On s'est concentré sur la réalisation de la tête, les mains étant déjà réalisées par une équipe précédente. Au terme du projet on a une tête parfaitement fonctionnelle (Voir figure 4.1 et 4.2) sur presque tout ses axes. Le déplacement vertical n'est pas encore en place puisqu'il nécessite que la tête soit fixée au torse (son moteur y est placé), mais celui ci est beaucoup plus complexe à assembler.

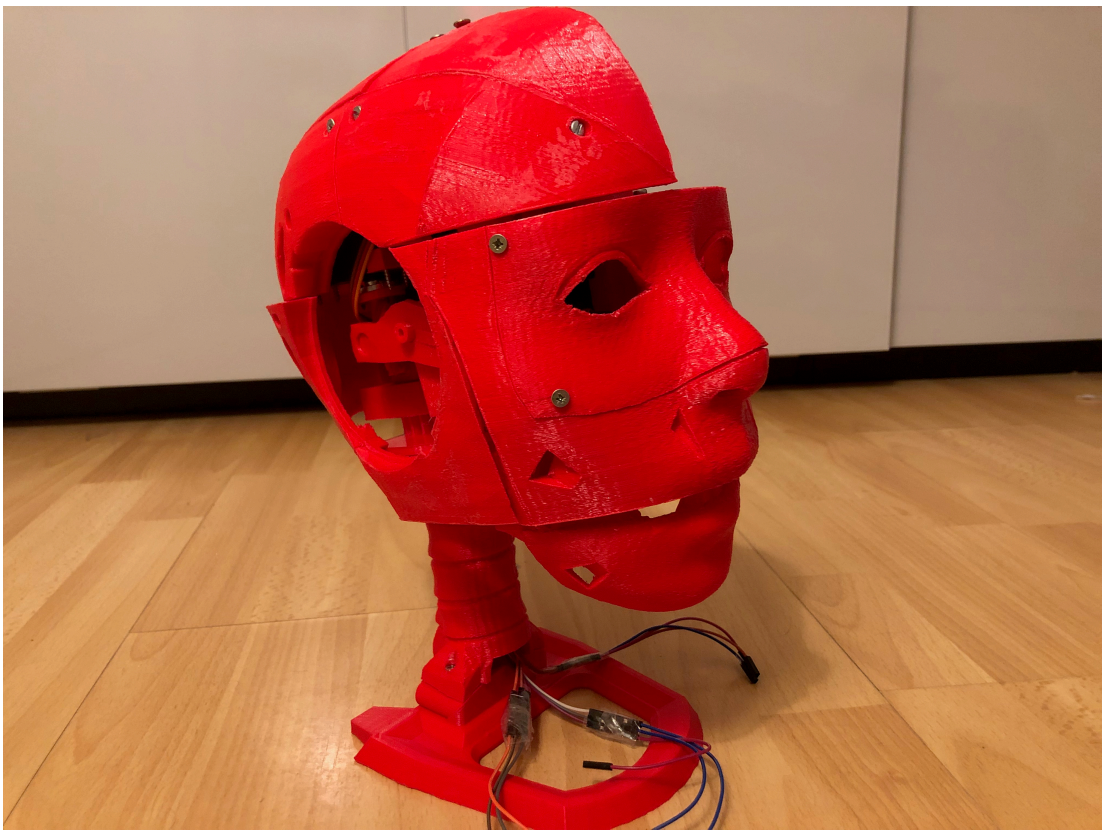

FIG. 4.1: Tête sur support vu de 3/4 face

Les caméras n'ont pas été mises sur les yeux par manque de matériel, mais cela est rapide en théorie.

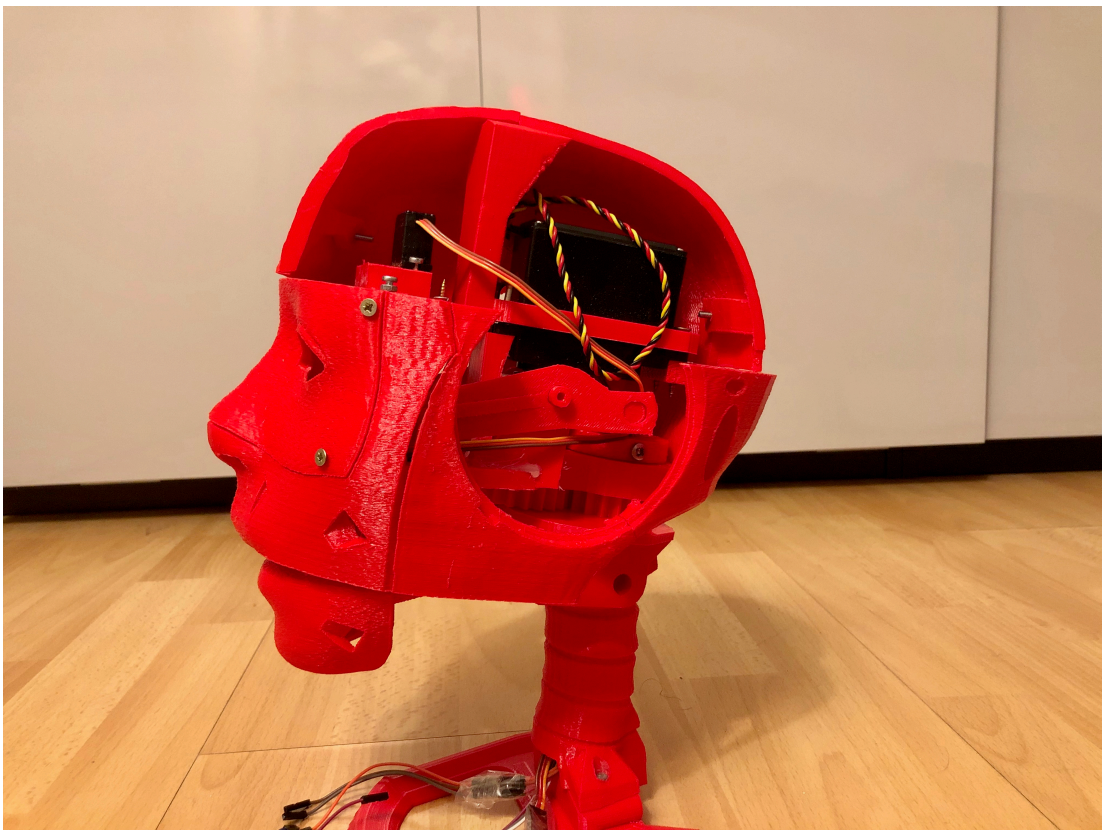

FIG. 4.2: Vu interne de la tête

4.2 Le logiciel

Voici donc l'interface finalement obtenue. Sur la page d'accueil (Figure 4.3) l'utilisateur peut voir les scripts déjà écrits. Ils sont marqué d'une couleurs différentes selon leurs modes de développements : Blockly ou Python. Ces scripts sont exécutables, éditables et supprimables.

Pour la création et l'édition de script l'utilisateur a ces deux choix. S'il fait un script Python, il a accès à un éditeur avec coloration syntaxique (Figure 4.4). Ce dernier est fait avec Ace (Ajax.org s. d.). S'il fait un script en Blockly il se retrouve sur un éditeur graphique (Figure 4.5).

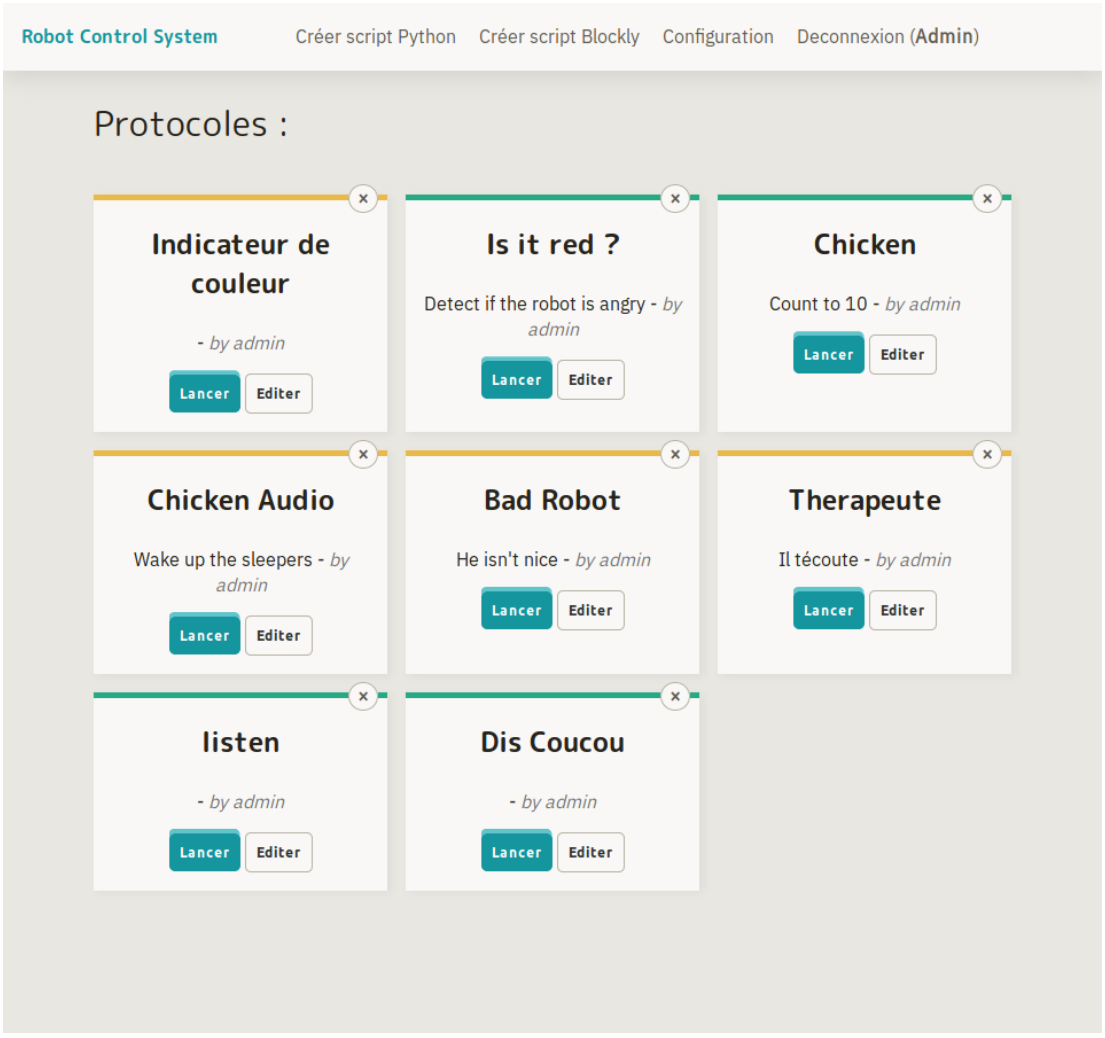

FIG. 4.3: Page d'accueil du site affichant une liste de script créé par l'utilisateur

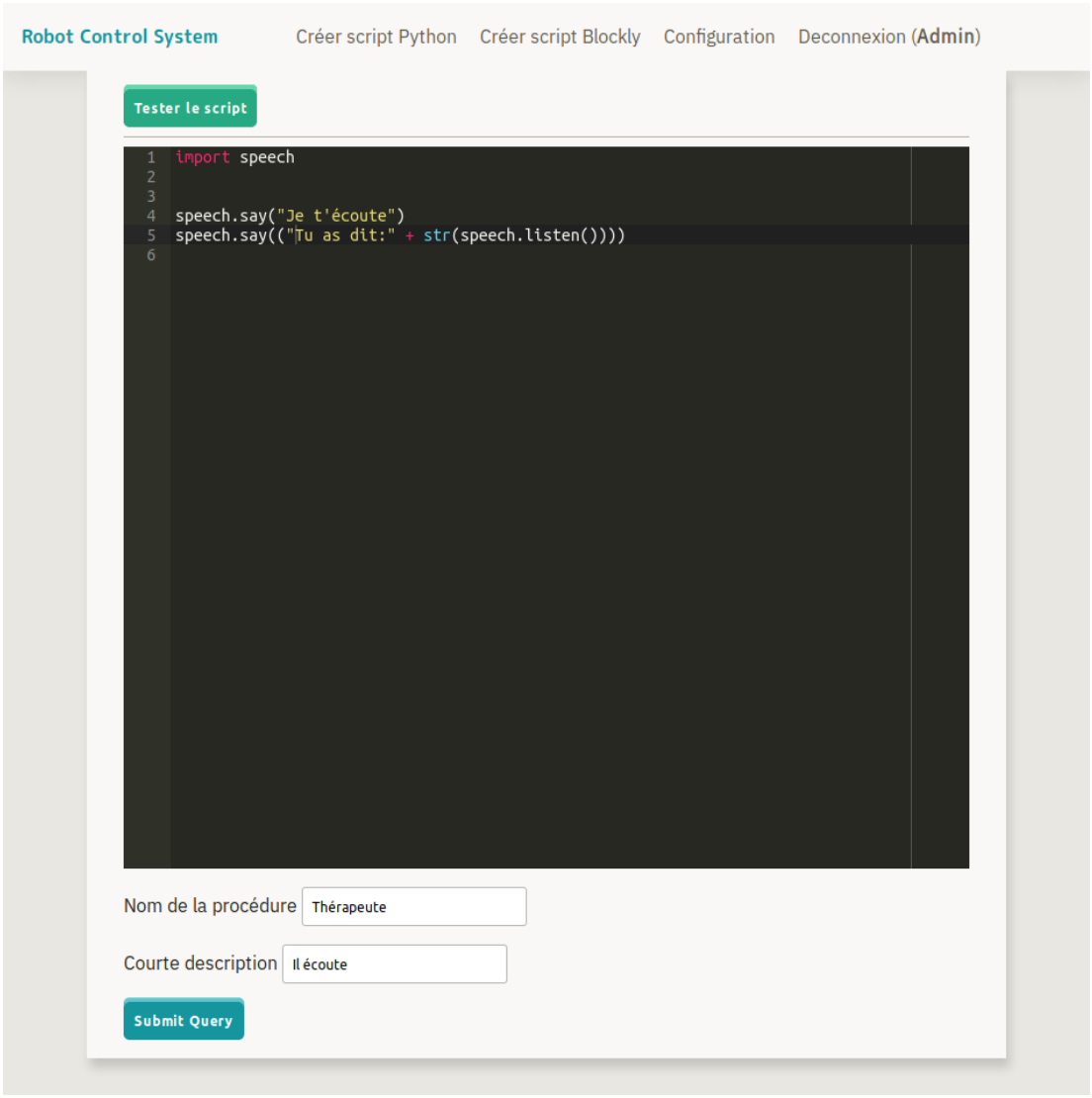

FIG. 4.4: Page de création de script python

FIG. 4.5: Page de création de script blockly

Au niveau des fonctionnalités, voici une liste des majeurs blocs permettant de programmer le robot :

Éléments des librairies Python	Blocs Blockly
time.sleep(**n**)	Attendre **n** seconde(s)
speech.listen()	Ce que le robot entend
speech.say(**s**, move_mouth=True)	Dire **s**
speech.say(**s**)	Ventriloquer **s**
movement.move_servo(**servo_nb**, **pca_id**, **percent**)	Mettre moteur n°**servo_nb** de carte **pca_id** à **percent**%

Éléments des librairies Python	Blocs Blockly
movement.stop_servo(**servo_nb**, **pca_id**)	Mettre moteur n°**servo_nb** de carte **PCA_id** au repos
movement.move_part(**part**, **percent**)	Bouger **part** à **percent**%
movement.move_part(**part**, **percent**, axis_rank=**axis_rank**)	Bouger **part** à **percent**% sur l'axe n°**axis_rank**
vision.detect_color_hex(**color**)	Couleur **color** détectée
Eyes, Jaws, Head, Shoulder[**side**], ...	Partie du corps variées

Du reste, les fonctionnalités de base du langage Python sont dans leurs majorités accessibles dans Blockly.

Chapitre 5

Conclusion

5.1 Réponse à la problématique

On s'est posé la question suivante au début du rapport : «Peut-on rendre accessible à tous la robotique anthropomorphique. Peut-on leur permettre d'expérimenter dans les différents champs où elle est utilisée ?». On cherchera à voir si notre projet y répond.

Le projet à un coup total d'environ 50€ (Voir table 5.1).

TAB. 5.1: Liste des coûts engendrés par le projet.

	Coût
Raspberry 3B+	32€
PCA9685 * 2	$\approx$ 3€
Câblage et divers	Une vingtaine d'euros
Total	autour de 50€

InMoov en lui même coûte dans les 1000€. Le coup du projet n'est donc pas négligeable.
Au niveau de l'installation, le projet est accessible à toutes personne ayant des notions de l'utilisation du shell Linux. Si un script d'installation est fourni, il est encore trop brut pour garantir une installation sans problème. L'installation du système Linux sur la Raspberry Pi est également une étape qui peut être complexe et qu'il faut prendre en compte. Les branchements sont en revanche minimales, et abordables pour des débutants (pour peu qu'on sache lire un schéma de câblage).

Le lancement du projet est plutôt simple. Une fois lancé il est à la portée de tous. Cela s'est vérifié lors de quelques tests auprès d'enfants de 8 à 11 ans : Les enfants comprennent sans problèmes comment programmer le robot, si on les guides. Des tests auprès d'adultes amateurs sont tout aussi concluants, et l'intérêt presque plus grand.

En somme le projet est encore non trivial à installer, mais son utilisation rempli la problématique posé. Les utilisateurs peuvent concevoir des projet d'interaction homme machine, se concentrer sur les réaction humaines grâce à des conditions techniques aisément reproductible. Bien entendu les fonctionnalités d'interaction manquent encore sérieusement.

Le robot InMoov reste encore un robot complexe à assembler, et coûteux. Il n'est donc pas idéal pour ce genre d'objectif.

5.2 Pistes futures

Que reste il à faire, et comment le projet pourrait il se développer dans le futur ?
On a vu que le projet pourrait être simplifié quand à son installation. des distribution pré-configurés de Raspbian pourrait être une solution. L'association pourrait également fournir des cartes micro-sd déjà configurées. En termes de construction du robot, sa réalisation nous a montré qu'InMoov n'était pas ce que l'on cherchais. Nos recherches nous on amenées à trouver quelques parties assemblable simplement et à coup réduit (On notera le robot de *Ryan Gross* par exemple). Peut être qu'un robot fait pour le projet seraient une bonne idée, se nourrissant de l'expérience acquise a en monter d'autres. Dans ce cas le projet prendrais un aspect triple : L'interface, le matériel de contrôle, et le robot. La totalité pourrait être estimé à quelques centaines d'euros.

Au niveau des fonctionnalités, c'est l'océan des possibles :
Le travail en traitement d'image n'a quasiment pas été fait, on pourrait donc implémenter divers algorithmes de reconnaissances. On pourrait imaginer une fonction `search_for(recognizer, image)` qui retournerait l'emplacement d'objets reconnus par un algorithme de reconnaissances de formes. Ensuite il faudrait développer ces algorithmes pour reconnaître des visages, le nombre de doigts, un chemin, voir... des couleurs ? L'utilisation d'apprentissage profond semble possible (via Tensorflow Lite par exemple) et s'est déjà vue sur des projets Raspberry pi, sinon la solution usuelle est l'utilisation de cascade de Haar prévue par opencv2.
Pour la synthèse vocale, le «lip sync» (mouvement de la bouche) est à revoir. On pourras exploiter les idées proposées dans le chapitre 3. La reconnaissance de la langue du texte écris et le changement de voix en fonction pourrait être une fonctionnalités utile. Dans tout

les cas la localisation, la traduction, pour d'autres langues que le français est à prévoir.
Si l'utilisateur souhaite créer un dialogue avec le robot, étant donné la fidélité relative de la reconnaissance vocale, une fonction de similarité de phrase serait pertinentes. Mettons que l'utilisateur souhaite que si l'on dit «Bonjour robot», le robot réponde «Bonjour, je suis un robot.». La reconnaissance vocale pourrais retourner au lieu de la phrase prononcée «Bonjour Roro» où autre quasi homophone, ou l'utilisateur final pourrait dire des phrases similaires («Salut Robot», «Bonjour Machine»...). Il faudrait une fonction qui traite tout ces cas-là et retournerait que ces phrases sont équivalentes ou presque. Évidemment c'est un sujet complexe, et le cas décris ici est idéal. Mais on pourrais déjà associer un dictionnaire de synonymes fréquents, et comparer les mots de la phrases à ceux ci. Pour éviter les erreurs liés à des conjugaison vaguement différentes, ou à des différences de genre, il faudrait également obtenir la racine du mot (le paquet python NLTK le permet). On pourrais aussi fournir un degrés de ressemblance phonétique, MaryTTS nous fourni les informations nécessaire de ce cotés-ci.
La possibilité d'importer un script doit être ajoutée. Il faudrait aussi récupérer la sortie d'erreur de python et la sortir à l'utilisateur du site. Le problème étant que plusieurs personnes personnes peuvent écrire du code en même temps, il faut donc afficher ces erreurs seulement à celui qui à exécuté le script. Il faudrait donc retenir l'utilisateur responsable de chaque exécution, et enregistrer dans la base de donnée la sortie standard et d'erreur. Ensuite un peu d'Ajax sur les pages d'édition se chargerait de récupérer les messages concernant chaque utilisateurs en permanences. C'est primordial pour une gestion d'erreur correcte. Dans la même veine, l'ajout d'une queue pour le lancement des scripts serait importante.
Reste enfin le mouvement par axes des servo-moteurs décrit au chapitre 3. Il faut absolument trouver une manière simple de gérer ce problème.

–>

References

Ajax.org, Ace Code Editor. Accessible via : https://ace.c9.io/.

Asano Yuki, Okada Kei & Inaba Masayuki, 2017. Design principles of a human mimetic humanoid : Humanoid platform to study human intelligence and internal body system. *Science Robotics*, 2(13). Accessible via : http://robotics.sciencemag.org/content/2/13/eaaq0899.

Banzi M., 2003. Arduino. Accessible via : https://www.arduino.cc/.

Blanchard A. & Mebarki D., 2018. The Neck of Pinobo, a Low-cost Bio-inspired Robot. Accessible via : https://hal.archives-ouvertes.fr/hal-01898012.

Compuphase, Colour metric. Accessible via : https://www.compuphase.com/cmetric.htm.

Gravato Marques H., Jäntsch M., Wittmeier S., et al., 2010. ECCE1 : The first of a series of anthropomimetic musculoskeletal upper torsos. *2010 10th IEEE-RAS International Conference on Humanoid Robots*, p.391-396.

Langevin G., 2012. *Inmoov*, Accessible via : http://inmoov.fr/.

Lapeyre M., 2014. *Poppy : plate-forme robotique open source, imprimée en 3D et totalement modulaire pour l'experimentation scientifique, artistique et pédagogique*. Theses. Université de Bordeaux. Accessible via : https://hal.inria.fr/tel-01104641.

Micro :bit Educational Foundation, 2015. Micro :bit. Accessible via : https://microbit.org/fr/guide/features/.

MIT Media Lab., 2006. Scratch 3. Accessible via : https://scratch.mit.edu/.

Multimodal Speech Processing Group, *MaryTTS*, Accessible via : http://mary.dfki.de/.

Zhang, A., 2017. *Speech Recognition*, Accessible via : https://github.com/Uberi/speech_recognition#readme.

Zlotowski J., Proudfoot D., Yogeeswaran K., et al., 2015. Anthropomorphism : Opportunities and Challenges in Human–Robot Interaction.