République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane Mira Bejaïa

Faculté de Technologie

Département de Génie Electrique

Mémoire de fin d'études

En vue de l'obtention du diplôme de Master en Automatique

Thème

Exploration des facultés de reconnaissances du robot NAO

Préparé par :

M^r^. DERRADJI Rabah

M^r^. AINOUCHE Micipsa Mehdi

Encadré et dirigé par :

M^r^. MENDIL Boubekeur

Devant le jury composé de :

M^me^. GAGAOUA Meriem

M^r^. SADJI Mustapha

Soutenu le : 07/09/2020

Année Universitaire : 2019/2020

Remerciements

Nous adressons nos plus sincères remerciements à notre encadreur Mr B. MENDIL pour sa disponibilité, ses orientations, ses remarques et ses conseils ainsi toute la confiance qu'il a mise en nous.

Nos profonds remerciements aux membres du jury pour l'honneur qu'ils nous ont fait en acceptant d'évaluer notre travail.

Nos remerciements vont également à l'ensemble de l'effectif du département de Génie électrique et à l'ensemble de nos enseignants qui ont contribués à notre apprentissage.

Nous tenons à exprimer toutes nos reconnaissances à tous ceux qui ont contribués de prêt ou de loin à la réalisation de ce modeste travail.

Dédicaces

Nous dédions ce modeste travail :

À nos très chers parents pour leurs efforts et sacrifices afin que nous puissions aujourd'hui réaliser notre rêve en travaillant sur ce projet remarquable et instructif.

À nos frères et sœurs, et à l'ensemble des membres de nos familles qui nous ont encouragés pour fournir plus d'efforts et le meilleur de nous-même.

À nos ami(e)s et camarades pour avoir passés de très bons moments lors des périodes de travails et pendant le cursus universitaire.

Tables des matières

Liste des abréviations

IA : Intelligence **A**rtificielle.

V5 : Version **5**.

CPU: Central **P**rocessing **U**nit (Microprocesseur).

LED: *L**ight-*E*****mitting *D****iode***. (DEL: Diode Electro-Luminescente)

IR: Infra**R**ed (Infrarouge)

RAM: Random **A**ccess **M**emory (Mémoire à Accès Aléatoire)

SDHC: Secure **D**igital **H**igh **C**apacity

FSR: Force **S**ensitive **R**esistors

SOC : System **O**n a **C**hip (Système embarqué sur une seule puce)

Wi-Fi : Wireless **F**idelity

IEEE 802.11 : Institute of **E**lectrical and **E**lectronics **E**ngineers.

USB : Universal **S**erial **B**us

$\boldsymbol{mW/sr}$ **:** Milliwatts par Stéradian.

LFsr : Left Fsr, **RFsr** : Right Fsr, **FL** : Front Left, **FR** : Front Right, **RL** : Rear Left, **RR** : Rear Right

HD: High **D**efinition (**H**aute **D**éfinition)

CMOS : Complementary **M**etal **O**xide **S**emiconductor

Fps : Frame per second (Image par seconde (ips)).

Ips : Image par seconde.

mV/Pa : millivolt par Pascal

dB : décibel

PC : Personal **C**omputer.

IP : Internet **P**rotocol.

2D : Bidimensionnel (Deux dimensions)

OpenCV : Open **C**omputer **V**ision

µm : micromètre

Mp : Méga **p**ixels (unité de mesure courante de la définition d'un capteur d'imagerie numérique)

V/Lux-sec : Volt par **L**ux **S**econdes

YUV : Y : Luma (Luminance)**, U** et **V :** Chrominance (Couleur)

DFOV : Display **F**ield **O**f **V**iew (Affichage de champ), **HFOV : H**orizontal **F**ield **O**f **V**iew (Champ de vision horizontal), **VFOV : V**ertical **F**ield **O**f **View** (Champ de vision vertical)

VGA : Video **G**raphics **A**rray (Standard d'affichage pour ordinateurs).

QVGA : Quarter **V**ideo **G**raphics **A**rray

ERS : Electronic **r**oller **S**hutter (Volet roulant électronique)

px : Pixel

RGB : Red (Rouge), **G**reen(Vert), **B**lue(Bleu).

HSY : H : Hue (Teinte), **S : S**aturation, **Y :** Luma (Luminance)

HSL : H : Hue (Teinte), **S : S**aturation, **L : L**ightness **(L**uminosité**)**

HSV : H : Hue (Teinte), **S : S**aturation, **V : V**alue (Valeur)

MRE : Magnetic Rotatry Encoders (Codeur rotatif).

Introduction générale

Introduction générale

L'évolution de la technologie est au cœur de l'actualité, et ce, depuis plusieurs années. Elle occupe une place importante dans tous les domaines, scientifique, sociale, éducatifs et bien plus encore. Elle modifie notre vision des choses et notre comportement vis-à-vis de ce qui nous entoure. Parmi elles, on distingue le développement de la robotique et de l'intelligence artificielle, à l'instar des robots humanoïdes, qui sans aucuns doutes, seront un jour nos futurs compagnons.

L'objectif de notre travail étant de réaliser une étude sur les facultés de reconnaissances et de développer des applications sur le robot humanoïde *NAO*. Dans un premier temps, on a pris connaissance des ressources hardware et software pour garantir une manipulation optimale et sécurisé du robot. Après, on s'est familiariser avec le logiciel du robot ainsi que le langage de programmation Python. La dernière étape était la réalisation et l'implémentation des applications.

Pour une bonne présentation du travail fait, on a divisé le manuscrit en quatre chapitres. Le premier chapitre traite les généralités sur les robots. Le second chapitre met en avant le robot *NAO* ainsi que son logiciel *Choregarphe*, permettant de le commander, avec une description détaillée, et consacre aussi une partie à la boite script Python. Ensuite, on a illustré dans le troisième chapitre les différentes caractéristiques de la caméra de *NAO*. Le dernier chapitre sera consacré aux applications réalisées avec Chorégraphe et le langage de programmation Python.

Chapitre I : Généralités sur les robots humanoïdes

I.1. Introduction

La robotique est l'ensemble des domaines scientifiques et industriels en rapport avec la conception et la réalisation de robots, ces derniers étant des dispositifs mécatronique, alliant mécanique, électronique et informatique, qui peuvent accomplir automatiquement soit des tâches qui sont généralement dangereuses, pénibles, répétitives ou impossibles pour les humains, soit des tâches plus simples mais avec une meilleure précision. **[Web 5]**

L'origine du mot robot vient du tchèque *robota* qui signifie le travail forcé **[3].**

Le terme *robotique* quant à lui vient de l'anglais *robotics*, imaginé par le romancier Isaac Asimov et popularisé par un livre publié en 1942, (*Runaround*) dans lequel il décrit les « trois lois de la robotique ».

Dans le domaine industriel, la robotique produit des automates réalisant des fonctions précises sur des chaînes de montage, aussi des engins capables de se mouvoir dans différents milieux : dangereux (pollués, radioactifs...), aérien, sous-marins, spatiaux, etc…, Outre l'industrie, la robotique concerne ainsi aujourd'hui la recherche scientifique, l'exploration spatiale et les activités de défense militaire ou de maintien de l'ordre. Elle intéresse également le secteur médical, pour les prothèses, les assistances aux chirurgiens ou aux infirmiers. Ainsi, on comprend tout l'intérêt de ce domaine vue les possibilités qu'il offre **[Web 2].**

I.2. Historique

Au cours de l'histoire et au fil de l'évolution de la technologie, la robotique a connu un développement considérable. Si on veut mettre en exergue cette évolution on peut alors distinguer trois types de robots.

Les premiers robots sont les « Automates » équipés de capteurs qui sont généralement programmés à l'avance et permettent d'effectuer des actions répétitives.

Ils sont entre autre utilisés dans le secteur automobile. Ils exécutent leurs actions peu importe l'environnement, le moment ou quelconque facteur extérieur. Ces automates sont composés :

- D'une structure mécanique articulée (bras).
- De moteurs qui permettront le mouvement.

- Et enfin d'un calculateur qui lancera directement l'action de façon répétitive.

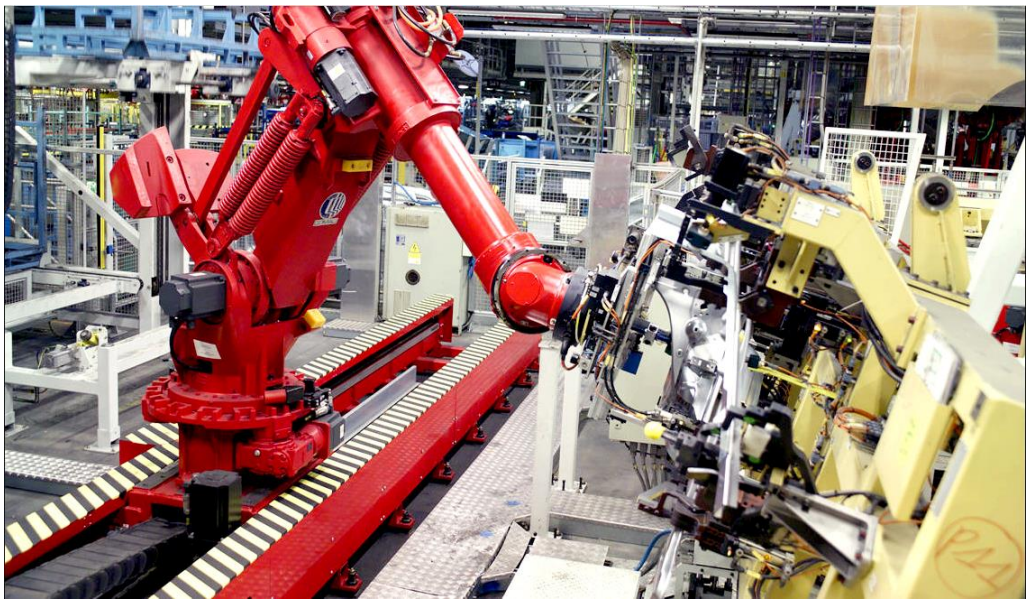


Figure I.1 : Automate employé dans le secteur automobile. **[Web 3]**

Le second type de robots correspond à ceux qui sont équipés de capteurs (de température, photo-électronique, à ultrasons etc…) leur permettant d'apercevoir l'environnement pour réaliser des tâches telles que l'évitement des obstacles et/ou le suivi de trajectoires. Ces capteurs permettent au robot une relative adaptation à son environnement afin de prendre en compte des paramètres aléatoires qui n'aurait pu être envisagés lors de leur programmation initiale. Ces robots sont donc bien plus sophistiqués que les automates, mais nécessitent un investissement en temps de conception et en coût de réalisation.

Figure I.2 : Robot du second type.

Le troisième type de robots correspond à ceux disposant d'une intelligence dite "artificielle" et reposant sur des modèles mathématiques complexes, tels que les réseaux de neurones. En plus de capteurs physiques comme les robots du second type, ces robots peuvent prendre des décisions beaucoup plus complexes et s'appuient également sur un apprentissage de leurs erreurs comme peut le faire l'être humain. Bien sûr, il faudra attendre encore longtemps avant que le plus "intelligent" des robots ne soit égal, tant par sa faculté d'adaptation que par sa prise de décisions, à l'Homme **[Web 1].**

Un exemple marquant de robots du troisième type est *SOPHIA*, un robot humanoïde mis au point par l'entreprise *Hanson Robotics*.

Figure I.3 : Robot humanoïde *SOPHIA*.

SOPHIA est une preuve réelle de l'avancée technologique que l'être humain a pu atteindre et sert comme une plateforme de recherche pour la robotique et la recherche sur l'IA, en particulier pour comprendre les interactions homme-robot et leurs applications potentielles de service et de divertissement. **[Web 4]**

I.3. Robots humanoïdes

Un robot anthropomorphique est un robot qui ressemble à un être humain, d'où le nom de robot humanoïde. Ces robots sont dotés de la bipédie (aptitude à marcher sur deux pieds), mais aussi de deux bras et d'une tête, qui seront, suivant l'application et l'environnement de travail, plus au moins opérationnels.

Le robot humanoïde est la principale motivation des roboticiens, dus entre autre aux hauts potentiels qu'ils présentent dans l'assistance aux personnes et l'aide qu'ils peuvent procurer dans différentes situations. Plusieurs robots humanoïdes ont été mis au point à des fins différentes, de l'aide aux astronautes de la station spatiale internationale à ceux qui peuvent jouais des instruments **[1].** On présente quelques exemples de ces robots.

I.3.1. ASMIO

ASIMO est un robot humanoïde bipède développé par *HONDA*, *ASIMO* est avant tout un robot de recherche dont l'objectif à terme sera de développer des robots qui coexisteront et qui seront capables de venir en aide aux personnes handicapées, âgées ou malades. *ASIMO* a été introduit en 2000. Sa dernière version date de 2011 cette version possède notamment 57 degrés de liberté et peut courir à une vitesse pouvant aller jusqu'à 9km/h, ce qui en fait le robot humanoïde le plus rapide actuellement.

ASIMO possède plusieurs caractéristiques, notamment :

- Une autonomie accrue
- Un comportement continu sans intervention humaine
- Une bonne adaptation aux différentes situations, grâce aux différents capteurs équivalents aux sens visuels, auditifs et tactiles d'un être humain.
- Il peut marcher sur des surfaces inégales.
- Effectuer des tâches avec dextérité, telles que ramasser une bouteille en verre et tourner le capuchon, ou tenir une tasse en papier doux pour verser un liquide sans l'écraser.
- *ASIMO* est capable de créer des expressions en langage des signes qui nécessitent le mouvement complexe des doigts. **[Web 7]**

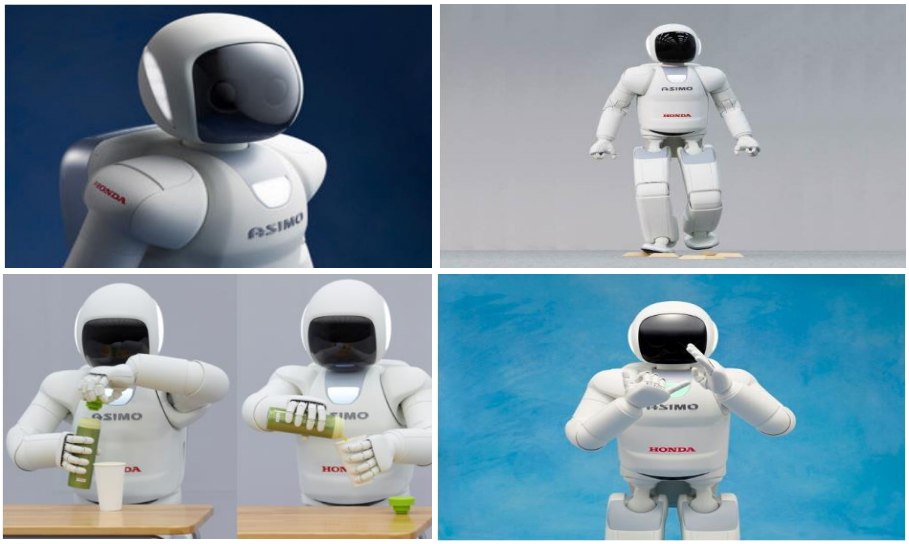


Figure I.4 : *ASIMO* et ses différentes capacités.

ASIMO, comme d'autres robots de sa catégorie, sont des robots à usage plus général, mais il existe des robots spécialisés comme *PEPPER*.

I.3.2. PEPPER

PEPPER est un robot humanoïde conçu par la société *SoftBank Robotics*. Il est capable d'identifier les visages et les principales émotions humaines. Il possède entre autre 20 degrés de liberté, parlent 15 langue (Anglais, Français, Espagnols, Néerlandais etc…) et l'écran tactile sur son torse affiche du contenu pour renforcer les messages et seconder la parole.

PEPPER peut interagir avec les humains à travers le dialogue et son écran tactile. Il est utilisé par des entreprises pour assister dans l'accueil, l'information et l'orientation des visiteurs **[Web 8].**

Figure I.5 : Robot humanoïde *PEPPER*.

Des robots plus petits et accessibles en termes de couts sont également développés pour travailler sur leurs caractéristiques mécaniques et/ou de perception et de reconnaissance, on trouve parmi cette catégorie le robot humanoïde *NAO*.

I.3.3. DARwin-OP

DARwIn-OP est un robot humanoïde programmable conçu par la société *Robotis*. *DARwIn-OP* a été créé à la base par l'université américaine de Virginie. Il est entièrement open-source, tant au niveau matériel que logiciel. Il possède 20 degrés de libertés et peut être programmé avec différents langage (C++, Python …etc.). Ce qui le rend une plateforme idéale pour l'enseignement et la recherche. **[Web 9]**

Figure I.6 : *DARwin-OP*.

I.4. Architecture des robots humanoïdes

Le robot humanoïde est constitué d'organes de perceptions, d'actionneurs et de capteurs de contact gérés par une centrale inertielle.

Les systèmes robotiques peuvent être décomposés en trois couches :

1. **La couche de perception** qui analyse l'environnement autour du robot et construit un modèle de cet environnement,
2. **La couche de décision** qui tentera de résoudre la tâche donnée au système tout en prenant en compte l'état actuel de l'environnement,
3. **La couche d'action** qui utilise les capacités du robot pour réaliser la tâche proprement dite.

En effet, cette catégorisation est issue en grande partie de la présence dans un système robotique de boucles lentes et de boucles rapides.

Les couches perception et décision nécessitent une grande puissance de calcul pour pouvoir assurer une réactivité suffisante. Cette puissance de calcul est parfois déportée sur un ordinateur spécifique. Cette architecture, largement décrite par la littérature, est encore d'actualité pour les robots humanoïdes **[2].**

I.4.1. La couche de perception

La perception d'un robot est en général un processus lent, dépendant de la vitesse à laquelle ces capteurs peuvent fonctionner. Il arrive parfois que les capteurs acquirent des données plus vite qu'elles sont traitées et certaines données sont alors ignorées silencieusement. De ce fait, le succès d'un robot humanoïde dépend du choix de ses capteurs (et de ses effecteurs), ils doivent être appropriés par rapport à la tâche.

L'autonomie du robot dépend aussi de sa perception de l'environnement. Dans cette couche, on classifie la perception par :

- Extéroceptive : localisation, obstacles…,
    1. Comme les capteurs passifs (caméras) qui sont de vrais observateurs de l'espace qui l'entoure. Ils capturent les signaux qui sont émis par les autres sources de l'environnement.
    2. Ou capteurs actifs (sonar, télémètre laser…), (Emetteur/Récepteur).
- Proprioceptive : attitude, vitesse, articulations, ….
    1. Ils informent le robot sur son propre état.
- Capteurs de contact :
    1. Ils informent si le robot est en contact avec un objet (obstacle) **[Web 11].**

I.4.2. La couche de décision

La prise de décision est aussi un processus lent qui peut prendre plusieurs secondes. La planification de mouvements en est un exemple. Tout mouvement est défini par ce qu'on appelle le module de génération de trajectoires. La trajectoire est générée dans un espace de configuration, et c'est ce qu'on appelle la planification de mouvement. Une fois la trajectoire faite, il y a la commande de mouvement, la trajectoire de référence doit être corrigé en fonction de l'environnement, des obstacles…etc. par exemple si le sol est incliné d'un certain

angle qui pourrait rendre le robot instable ou si un obstacle se présente sur la trajectoire du robot, le robot réagira en fonction : c'est ce qu'on appelle l'asservissement **[Web 10].**

I.4.3. La couche d'action

Le robot dispose d'actionneurs, des dispositifs permettant au robot d'exécuter une action comme marcher ou lever son bras. Nous avons entre autre, le vérin pneumatique qui sert à créer un mouvement mécanique (responsable de la levée ou du maintien de charges), mais aussi du moteur électrique qui permet au robot de convertir l'énergie électrique en énergie mécanique.

La couche action contrôle les actionneurs et nécessite d'être extrêmement réactive et impose des contraintes importantes en termes de temps réel. Il faut pouvoir garantir que la boucle de contrôle peut être évaluée à une fréquence donnée. Ce qui contraint à la fois le type de programme informatique pouvant être exécuté dans cette boucle ainsi que le volume de données pouvant être traité. **[2]**

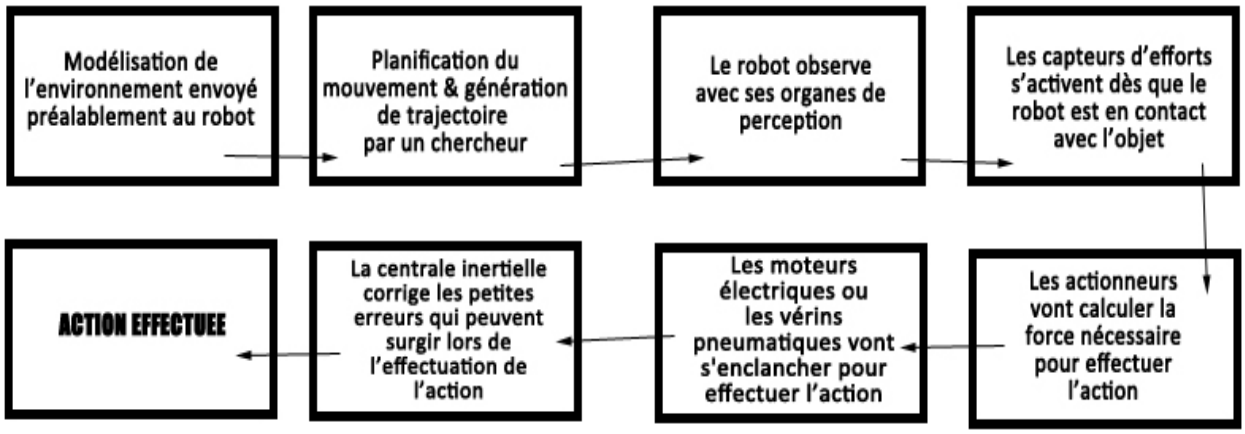


Figure I.7 : Processus d'action pour un robot humanoïde

I.5. Conclusion

Dans ce premier chapitre, on a abordé l'histoire de la robotique, les différentes générations de robots, décrivant l'évolution de cette discipline. On a présenté les différents types de robots humanoïdes et on a fini par leurs caractéristiques, ainsi que leurs architectures.

Chapitre II : Le robot humanoïde NAO et le logiciel Choregraphe

II.1. Introduction

Ce 2eme chapitre présente en détaille le robot humanoïde *NAO V5*, les différents composants électroniques qui le constituent, ainsi que leur principe de fonctionnement. Ensuite, on donne une description de la mise en marche du robot et sa configuration. La dernière partie est consacrée à l'essentiel du logiciel *Choregraphe* et la boîte *script Python*.

II.2. Historique et présentation du robot NAO

NAO est un robot humanoïde, entièrement programmable, autonome et interactif, conçu et développé en 2008 par la société *Aldebaran Robotics*, une start-up française créée en 2005, rachetée par le groupe japonais *SoftBank Group* en 2015 qui la renomme en *SoftBank Robotics*.

Figure II.1. Le robot humanoïde *NAO* et le logo *SoftBank Robotics*. **[Web 14]**

Passé par plusieurs versions depuis sa création, il est aujourd'hui à sa 6ème qui a été lancée en 2018. Formidable outil de programmation, il est devenu une référence dans l'éducation et la recherche, capable d'interagir avec l'être-humain et l'environnement, *NAO* peut voir, entendre, parler, sentir et communiquer. Pour ce faire, les concepteurs lui ont intégrés différents types de technologies : CPU, capteurs, actionneurs, caméra et LEDs.

Figure II.2. Evolution de *NAO* **[Web 14]**

II.3. Architecture et composants de NAO

Figure II.3 illustre bien les différents éléments de *NAO*. Ses caractéristiques techniques essentielles sont regroupées dans Tableau II.1.

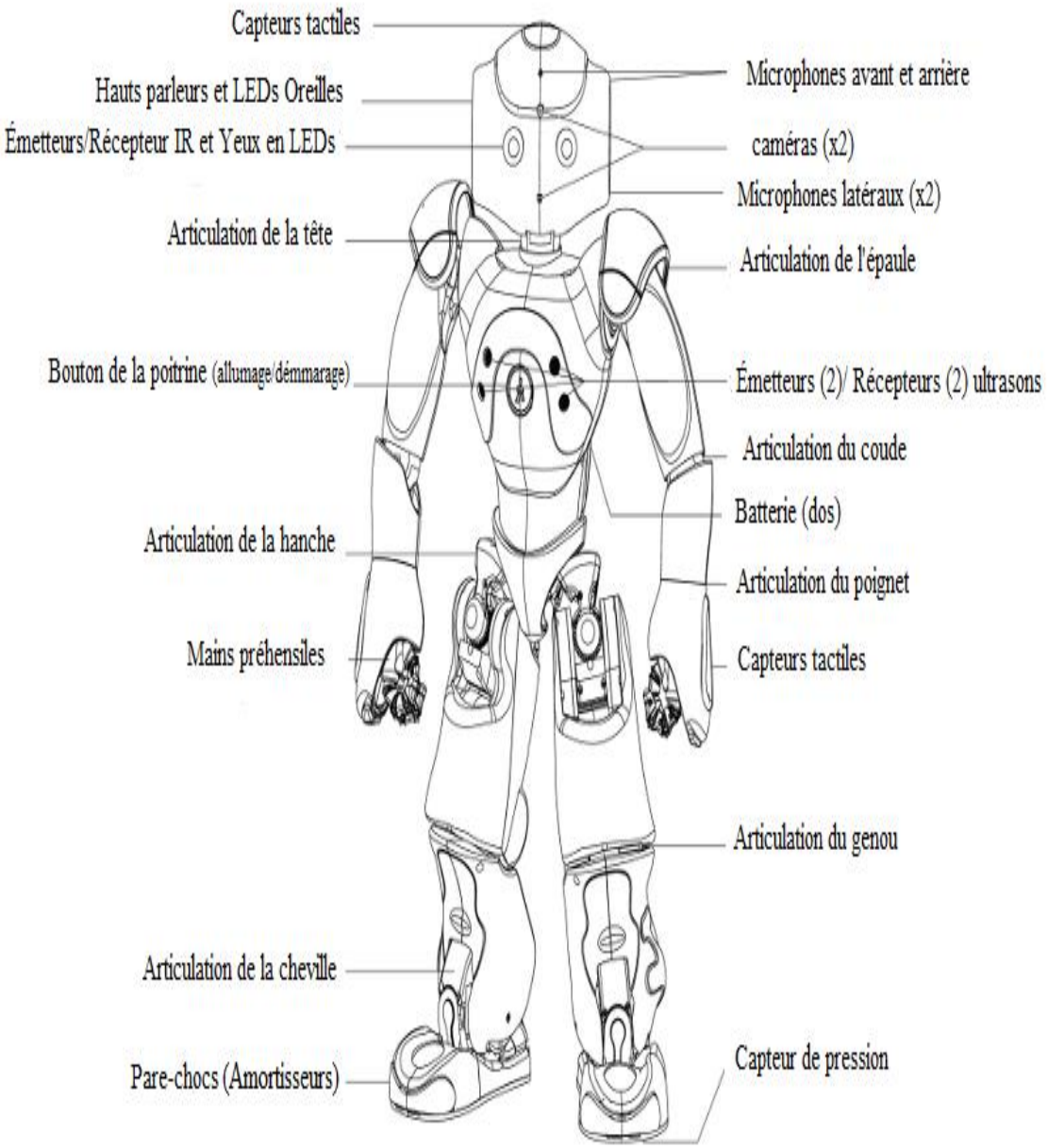


Figure II.3. Illustration détaillant l'ensemble des éléments qui sont inclus dans NAO. **[5]**

Tableau II.1. Composants électronique de *NAO*. **[6]**

Éléments	**Types**	**Spécifications**
Carte mère	**ATOM Z530**	CPU 1.6 GHz
	RAM	1 Go
	Mémoire flash	2 Go
	Mémoire SDHC	8 Go
Capteurs	Ultrason	2 Emetteurs /2 Récepteurs
	Infrarouge	2 (Yeux)
	Capteurs Tactiles	Emplacement : Tête, mains et pieds
	Résistance à capteur de force (FSR)	Emplacement : Pieds
	Capteurs de position d'articulation	Capteur à effet de Hall 36 à 2 MRE
	Caméra (capteur d'image SOC)	HD, $1280\text{x}960, 30fps$
LEDs	/	Disposition : Tête, yeux oreilles, mains et pieds
Haut-Parleurs	/	Nombre : 2 Stéréo
Microphones	/	Nombre : 4 sur la Tête Sensibilité : 20mV/Pa +/- 3dB à $1\ KHz$ Fréquence : $150Hz$ à $12kHz$
Batterie	Lithium ion	Durée : 60 à $90mn$ Temps de recharge : 3 heures
Connectivité	Wifi	IEEE 802.11 a/b/g/n
	Ethernet	1×RJ45 - 10/100/1000 base T
	USB	2.0

II.4. Description des capteurs [11]

NAO a été conçu pour s'adapter à son environnement. Pour cela, il a été équipé de plusieurs capteurs afin de rendre sa perception, de ce qui l'entoure, précise et optimale.

II.4.1 Disposition des capteurs ultrasons

NAO possède deux émetteurs et deux récepteurs ultrasons comme illustré dans la figure II.4.

➢ Spécifications :

- **Fréquence :** 40 kHz.
- **Résolution :** 1 cm à 4 cm.
- **Portée de détection :** De 0.20 m à 0.80 m. Au-dessous de 20 cm, la distance ne peut être mesurée, la présence d'un objet est la seule information donnée. Au-dessus de 80 cm, la valeur renvoyée est une estimation.
- **Cône d'environ :** 60°.

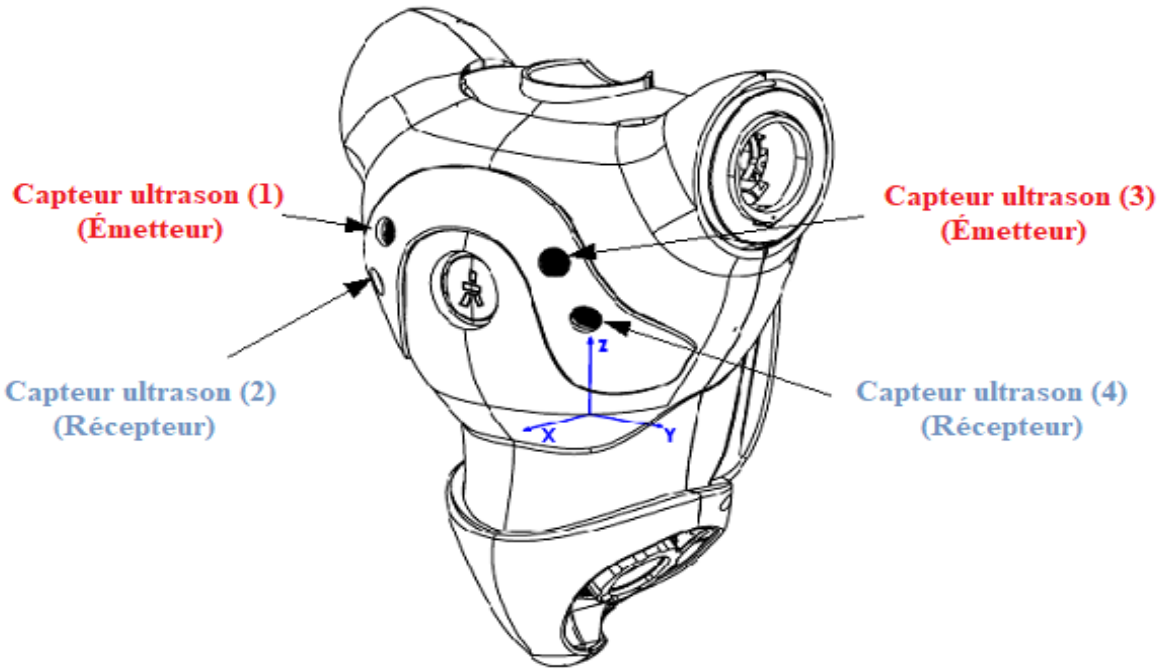


Figure II.4. Emplacement des capteurs ultrasons ainsi que leur ordre de numérotation. **[11]**.

II.4.2 Disposition des capteurs infrarouges

Les deux capteurs infrarouges (IR) de NAO se situent au niveau de ses yeux comme indiqué sur la figure II.5.

➢ Spécifications :

- **Longueur d'onde :** 940 nm.
- **Angle d'émission :** $+/-$ 60°.
- **Puissance :** 8 mW/sr.

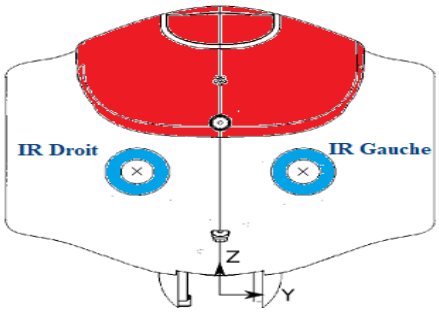


Figure II.5. Emplacement des capteurs infrarouges indiqués par les deux croix. **[11]**

II.4.3. Capteurs tactiles

NAO possède plusieurs capteurs tactiles, répartis sur toute sa structure, 03 réparties sa tête, 03 autres sur chaque mains et 02 sur chaque pied comme illustré sur la figure II.6. Ce qui offre une autre possibilité de communiquer avec l'utilisateur.

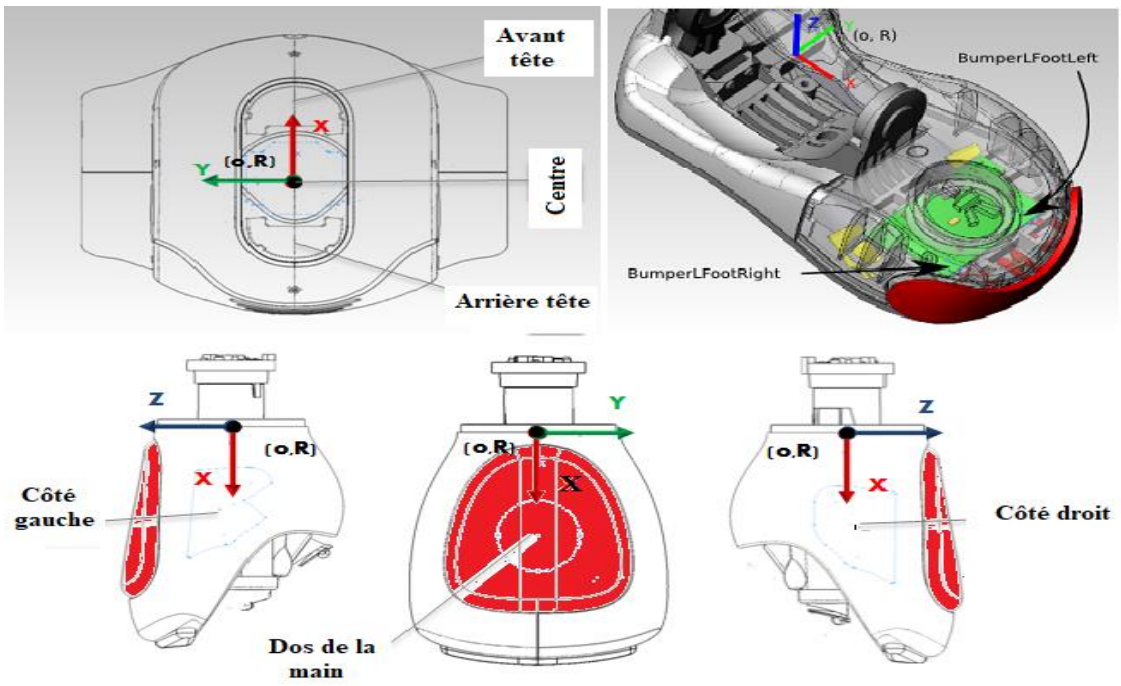


Figure II.6. Illustration indiquant la disposition des capteurs tactiles de *NAO* **[11]**.

II.4.4. Capteur de force résistif (FSR)

Les capteurs FSR (*Force Sensitive Resistors*) sont placés en dessous de chaque pied de l'humanoïde pour faciliter l'équilibre comme le montre la figure II.7. Ce type de capteurs mesure la variation de la résistance selon la pression appliquée. L'intervalle de fonctionnement est de [0 à 25 N].

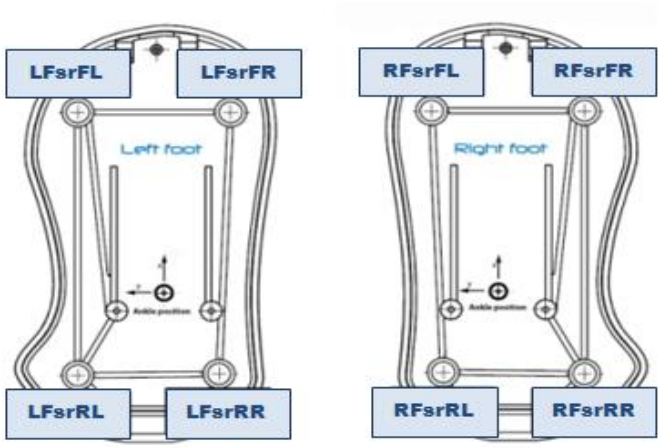


Figure II.7. Vue d'en haut de la position des capteurs Fsr (**LFsr** : capteur pied gauche, **RFsr** : capteur pied droit, **FL** : Front Left, **FR** : Front Right, **RL** : Rear Left, **RR** : Rear Right) **[11]**.

II.4.5. Caméra

NAO possède 2 caméras HD 1280 x 960 pixels à capteurs CMOS capables de capter 30 images par seconde ($30fps$), qui sont utilisées pour l'identification d'objets présents dans son champ de vision tel qu'un ballon ou une carte de jeu.

.

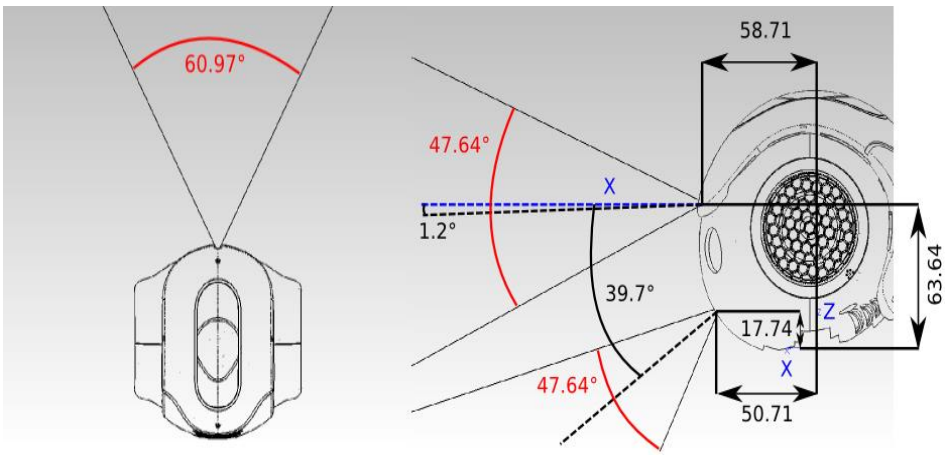


Figure II.8. Champ de vision de *NAO* sur l'horizontale et la verticale (unité distance : mm, angle) **[6]**.

II.5. Les moteurs (Actionneurs) [6]

Pour que *NAO* ait 25 degrés de libertés et soit capable de se mouvoir et de reproduire certains mouvements de l'être humain, les concepteurs l'ont équipé de plusieurs moteurs jouant le rôle d'articulations, comme l'illustre la figure II.9. **[7]**

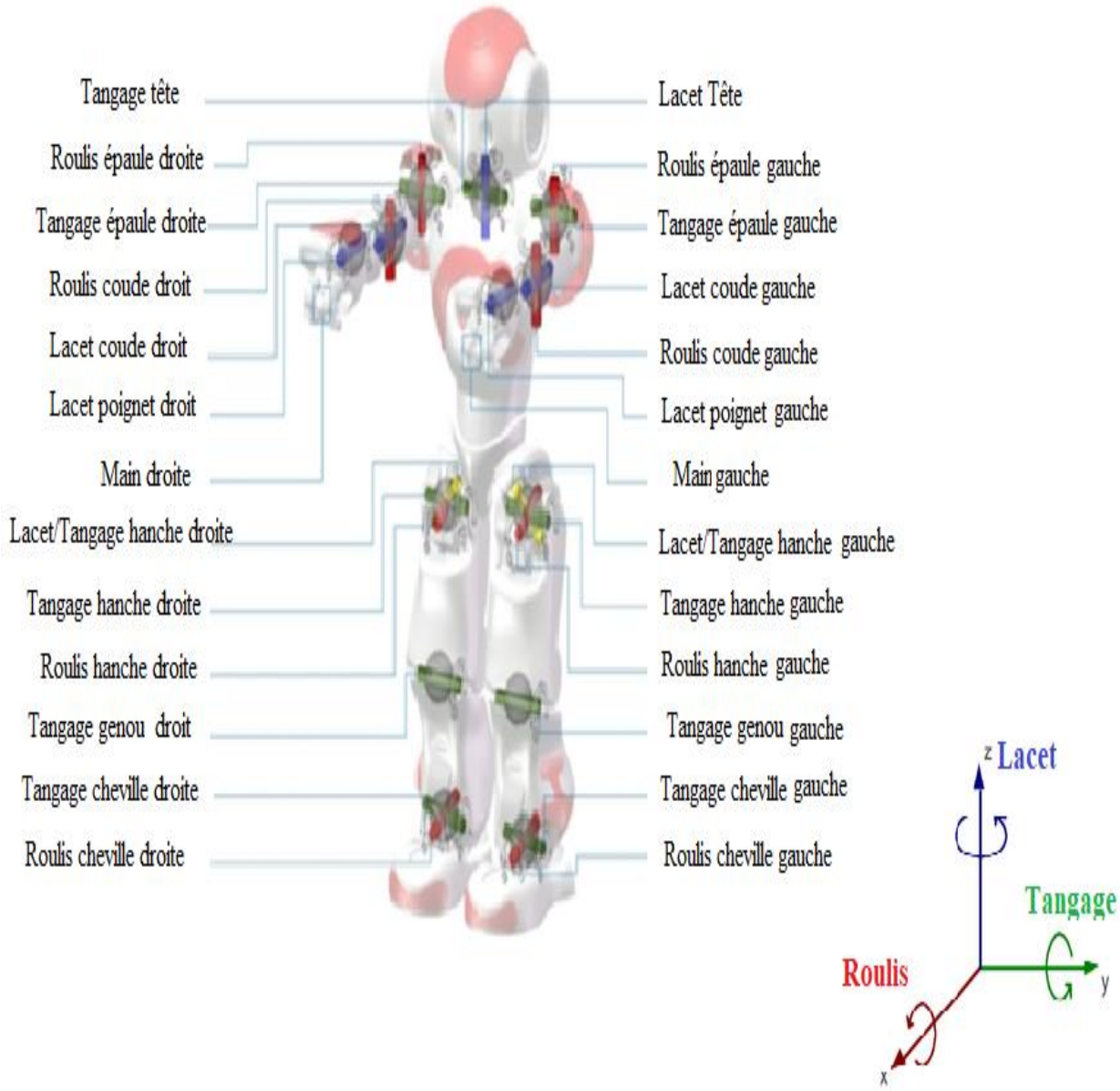


Figure II.9. Ensemble des articulations incluses dans *NAO* qui fonctionnent selon le repère indiqué dans la figure.

II.6. Autres composants [5]

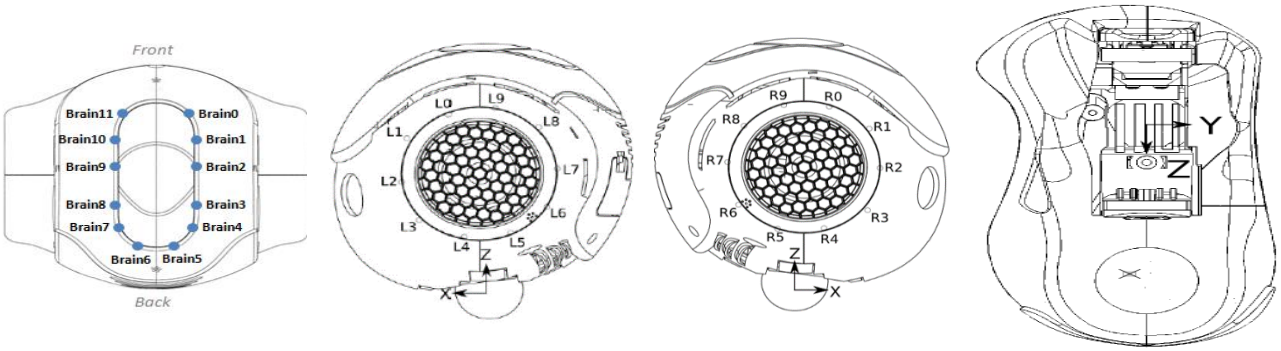


Figure II.10. Répartition des LEDs selon leurs numéros (Tête, oreilles, pied droit et gauche).

II.6.1. LEDs

NAO est un robot très élégant, notamment grâce à ses LEDs qui lui donnent un aspect esthétique. Mais pas que, en effet, selon la couleur de ces dernières, *NAO* pourra nous fournir des informations sur son état en temps réel. Nous développerons tout ça dans la partie **« Mise en marche »**.

II.6.2. Haut-parleurs

Appareil qui transforme des courants électriques (détectés par un récepteur et amplifiés par un amplificateur), en ondes sonores de manière à permettre l'écoute (collective) à distance. C'est donc un transducteur électroacoustique. Situés au niveau de ses oreilles, ils permettent à *NAO* de communiquer un message ou de jouer de la musique.

II.6.3. Microphones

Un microphone est un autre transducteur électroacoustique, ce qui signifie qui peut convertir un signal acoustique en signal électrique (l'inverse du haut-parleur). *NAO* possède quatre microphones situés dans sa tête.

Le microphone est aussi un transducteur électroacoustique, mais contrairement au haut-parleur, il convertit ondes sonores acoustiques en impulsions électriques. *NAO* possède quatre microphones situés dans sa tête.

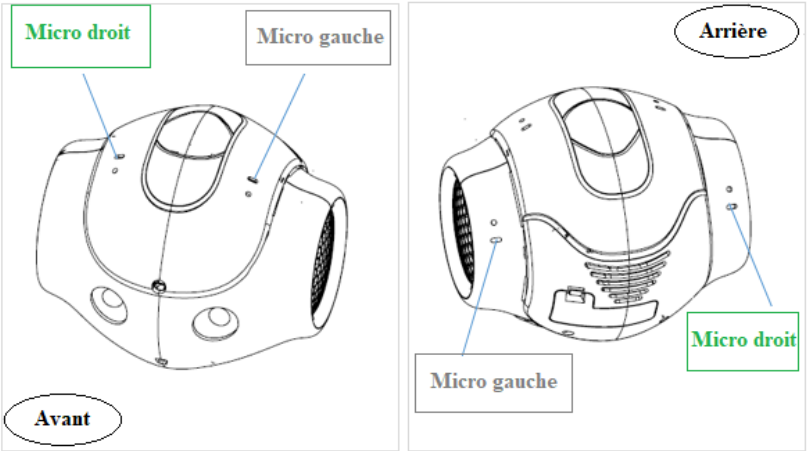


Figure II.11. Microphones intégrés dans *NAO*.

II.7. Mise en marche (connectivité) [4]

Avant son utilisation, *NAO* doit être manipulé avec précaution. Pour cela, il est primordial de suivre les consignes fournies par le constructeur. Voici les étapes à suivre.

II.7.1. Connexion et initialisation de NAO

Pour une toute première utilisation, il faut d'abord mettre le robot en position de sécurité et ne pas mettre les mains sur ses articulations. Puis, brancher l'alimentation avec le chargeur distribué. Ensuite, connecter *NAO* au PC via un câble Ethernet (ou bien vers un routeur auquel est connecté le PC). Cette étape permet de créer et de configurer un réseau Wifi entre le robot et le PC et offrir une meilleure expérience d'utilisation, en se débarrassant des câbles encombrants et dangereux durant les déplacements du robot. D'après les tests qu'on a effectués, il n'est pas exigé que le routeur (Modem) soit connecté à Internet afin d'établir une connexion. Par contre, si on veut télécharger une application, la connexion Internet est obligatoire.

Figure II.12. Position de sécurité et méthode de connexion à *NAO*.

Une fois la liaison établie, on peut procéder à la mise en marche de l'humanoïde et cela en appuyant sur le bouton du torse (indiqué par la figure au-dessus). Le temps de démarrage peut prendre de 3 à 5 mn. On peut suivre la progression de démarrage à travers les LEDs qui entourent les haut-parleurs. Après un moment, le robot prononcera l'expression « **OGNAK GNOUK** », puis, il va commencer à regarder autour de lui et reconnaitre des visages, c'est un signe que *NAO* est prêt à l'utilisation.

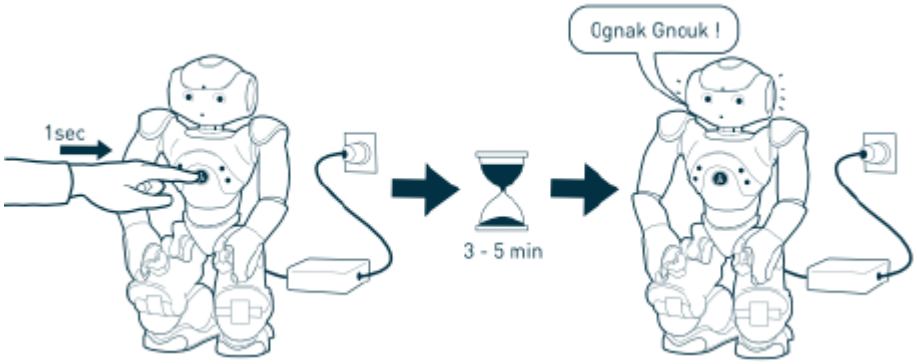


Figure II.13. Mise en marche de l'humanoïde.

À la fin de l'utilisation, la procédure de l'extinction se fait ainsi : il faut appuyer sur le même bouton de démarrage pendant cinq secondes. Les lumières des yeux, du torse et de la tête seront toutes éteintes quand le processus d'extinction sera terminé.

Lorsqu'on communique avec *NAO*, ses yeux s'illuminent avec différentes couleurs :

- Jaune, signifie qu'il vous écoute et traite les sons.
- Bleu, en attente de réception de messages.
- Vert, signifie que le message est reconnu.
- Il alterne entre le blanc et le bleu lorsqu'il parle **[4].**

II.7.2. Configuration Wi-Fi

D'abord, il est nécessaire de créer un compte sur *Aldebaran Community* en raison de la synchronisation. Pour configurer le Wi-Fi pour la première fois, il faut garder le câble Ethernet branché. Par la suite, on appuie sur le bouton de mise en marche après que *NAO* soit démarré afin d'obtenir son adresse IP (Internet Protocol). Si on n'a pas bien retenu les chiffres, on appuie une nouvelle fois pour qu'il répète le message moins vite.

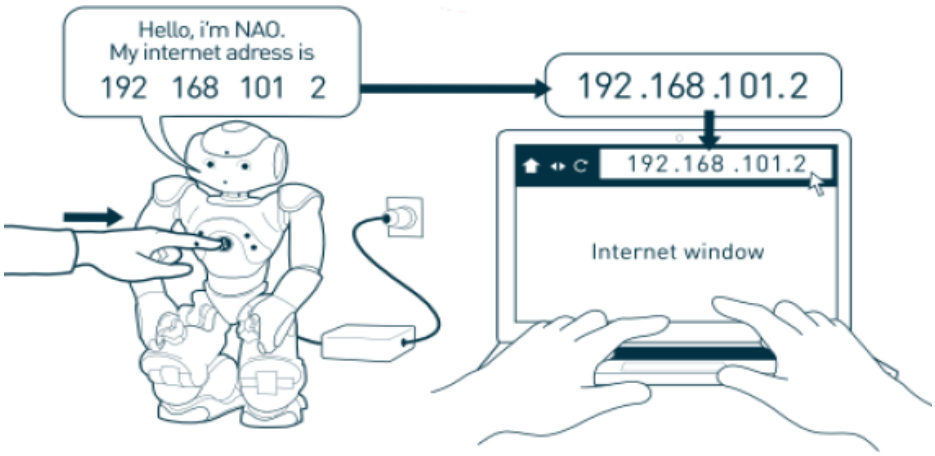


Figure II.14. Obtention de l'adresse IP.

A présent qu'on a récupéré l'adresse, on l'insère dans la barre d'adresse d'un navigateur web (Firefox ou Chrome) et on valide la recherche. Une boite de dialogue d'authentification va apparaître qui demande de saisir le nom d'utilisateur et son mot de passe qui sont par défaut « nao, nao » respectivement **[4].** On les a modifiés par « nao, scanao ».

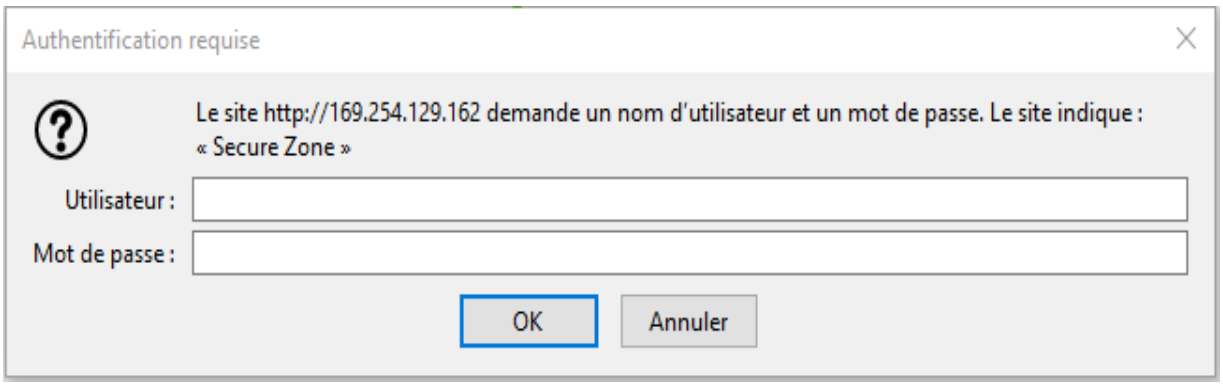


Figure II.15. Boite de dialogue d'authentification.

Après cela, l'assistant « *Getting Started* » va démarrer pour guider l'utilisateur.

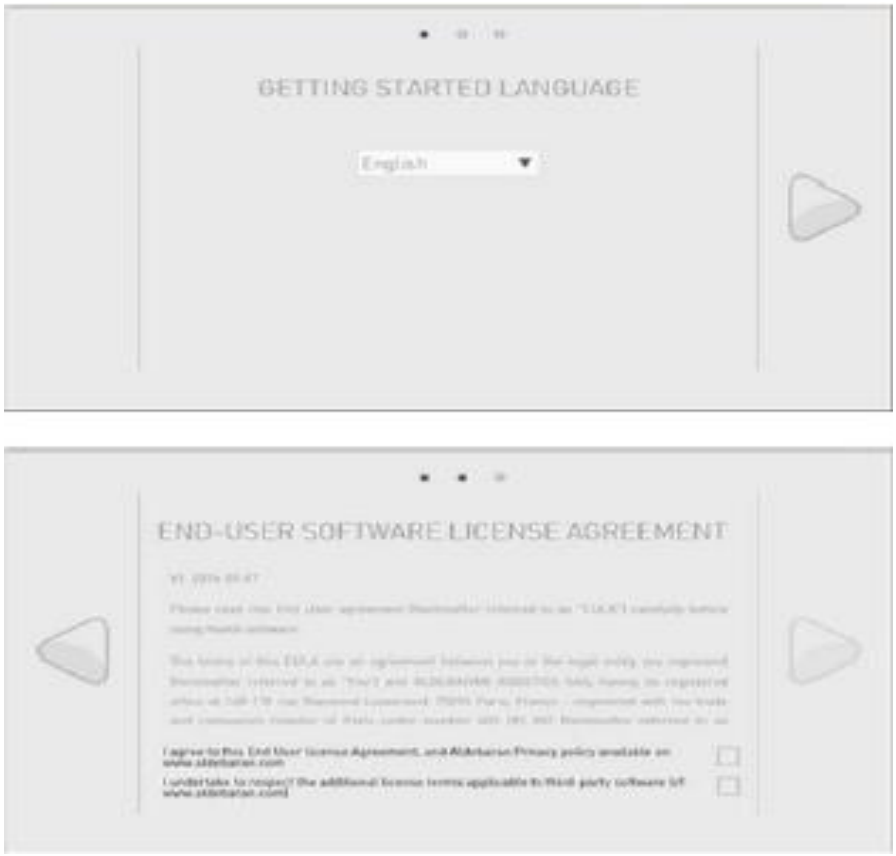


Figure II.16. L'assistant de configuration Wi-Fi

Arrivé à l'écran de configuration Wi-Fi, on peut en sélectionner un puis saisir son code d'accès, ou bien cliquer sur "*Skip*" ou "*Ignore*" pour laisser Nao sans Wi-Fi [4]. On pourra toujours le connecter à un réseau Wi-Fi plus tard. Mais dans les deux cas, le câble Ethernet doit rester brancher pour poursuivre la configuration.

Figure II.17. Réseaux Wi-Fi détectés à proximité.

On a la possibilité de changer le nom du robot ainsi que le mot de passe. (Par-contre, il faut noter précisément les changements, ils seront demandés à chaque authentification).

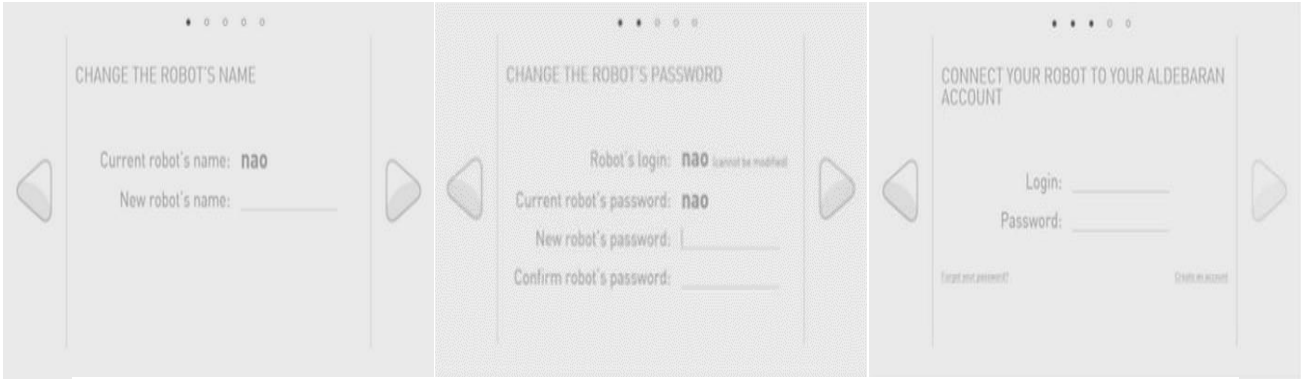


Figure III.18. Fenêtres de configuration du nom, du mot de passe et de la synchronisation du compte.

Si le nom du robot est modifié, alors un reboot est nécessaire. Il est recommandé qu'avant le reboot, la batterie soit chargée et pour une meilleure précaution, que le chargeur soit branché.

Si ces deux conditions sont respectées, alors on peut redémarrer *NAO* en cliquent sur "*Reboot now*". En cliquant sur « *Finish* », on obtient l'interface suivante **[4]** :

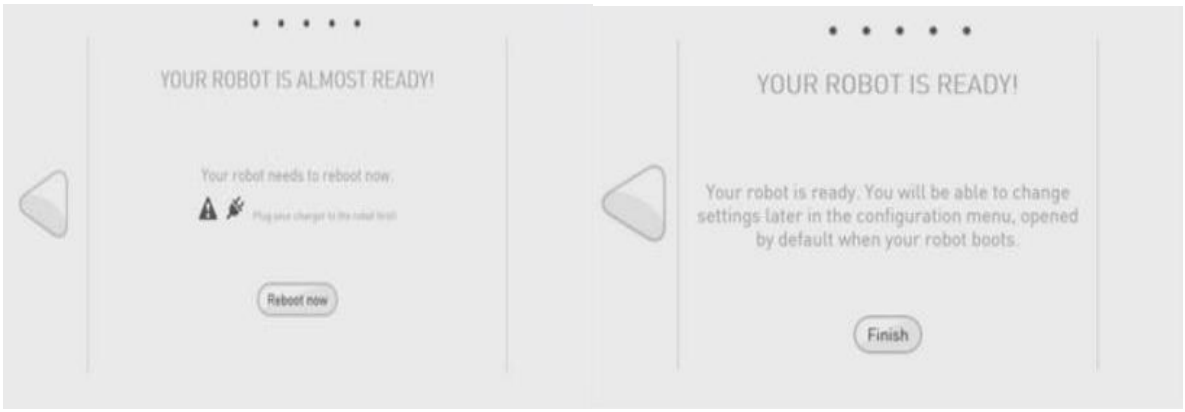


Figure II.19. NAO est prêt pour le redémarrage

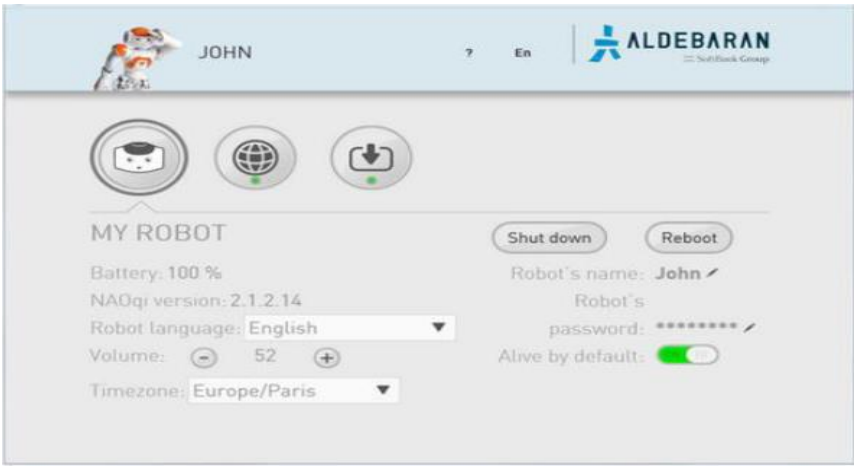


Figure II.20. Interface Web de *NAO*.

Une fois la configuration terminée, on lance l'application Choregraphe et on introduit l'adresse IP de *NAO*. Car, cette fois-ci, on peut débrancher le câble Ethernet et le chargeur de batterie (après la charge) et donner plus de liberté de mouvements à *NAO* et une certaine autonomie.

II.8. Le logiciel Choregraphe [4][5]

Choregraphe est un logiciel multiplateforme, permettant de créer des animations, comportements et dialogue, de les tester dans un simulateur ou directement sur *NAO*, et de surveiller et contrôler le robot. On peut aussi compléter nos programmes en codant avec Python par exemple pour enrichir l'expérience utilisateur.

II.8.1. Installation de Choregraphe

Pour installer *Choregraphe*, on suit les étapes ci-dessous :

- D'abord on télécharge la dernière version de *Choregraphe* sur le site web *Aldebaran Community (SoftBank Robotics)* : https://community.aldebaran.com/en/resources/software
- Ensuite on vérifie que la version de *Choregraphe* corresponde bien à notre version de *NAOqi*.
- On coche la case *NAO* sous la section *MY ROBOTS* à gauche
- On clique sur la version correspondant à notre ordinateur (Windows, Linux ou Mac) pour notre cas sa sera Windows.

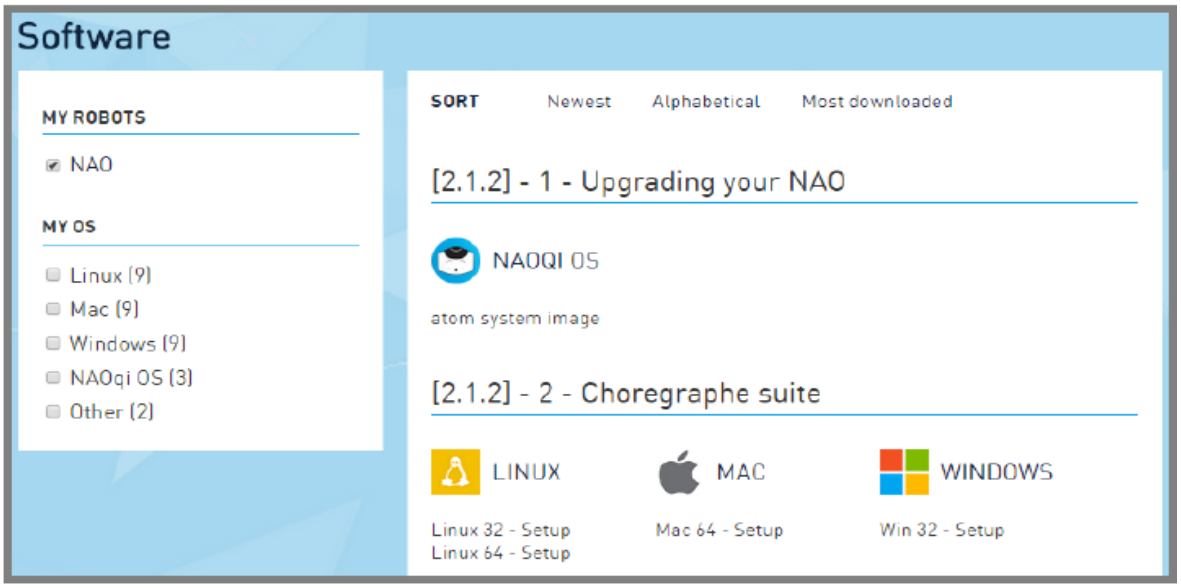
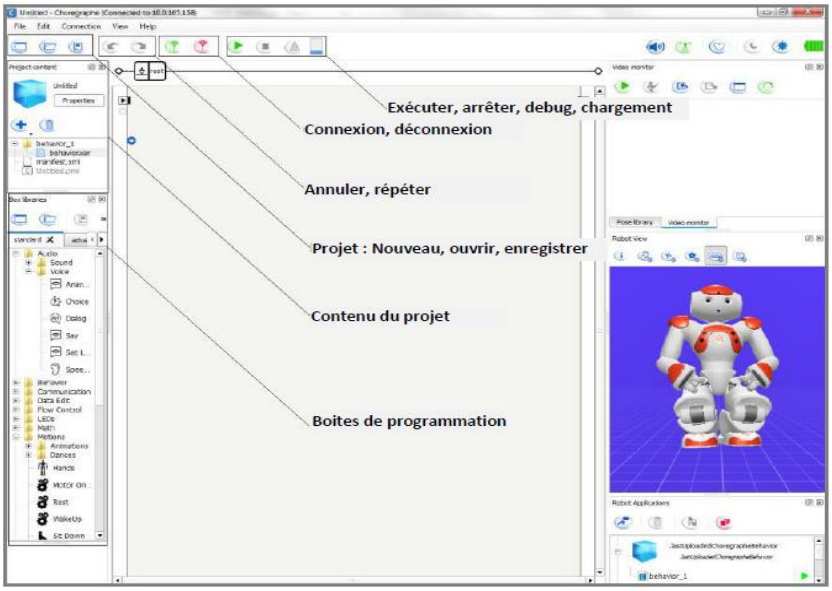


Figure II.21. Fenêtre du site web *SoftBank Robotics* pour l'installation le *Choregraphe*

- On exécute le fichier récupéré et on suit les instructions d'installation.
- Si le démarrage de *Choregraphe* se fait pour la toute première fois, il sera demandé de saisir une clé de licence. Elle doit être reçu avec NAO, soit à partir du représentant de *SoftBank Robotics* (Inexistant en Algerie) ou par un e-mail de *SoftBank Robotics*. Cependant, il y'a possibilité d'utiliser la version 90 jours en attendant. (Pour notre cas la clé a été téléchargé par internet via un site web tiers).

II.8.2. Description de l'interface de Choregraphe

Par défaut, l'interface du logiciel *Choregraphe* se présente comme indiqué par la Figure II.22.



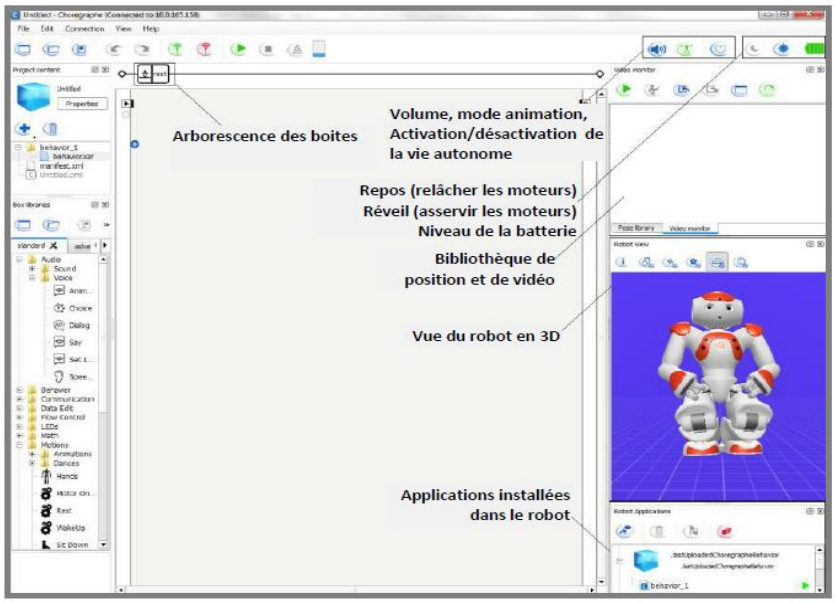


Figure II.22. Interface du logiciel *Choregraphe.*

II.8.3. Connection à NAO et démarrage d'application

- On ouvre *Choregraphe*
- On clique sur "connection" puis "connect to…" pour ouvrir le gestionnaire de connection

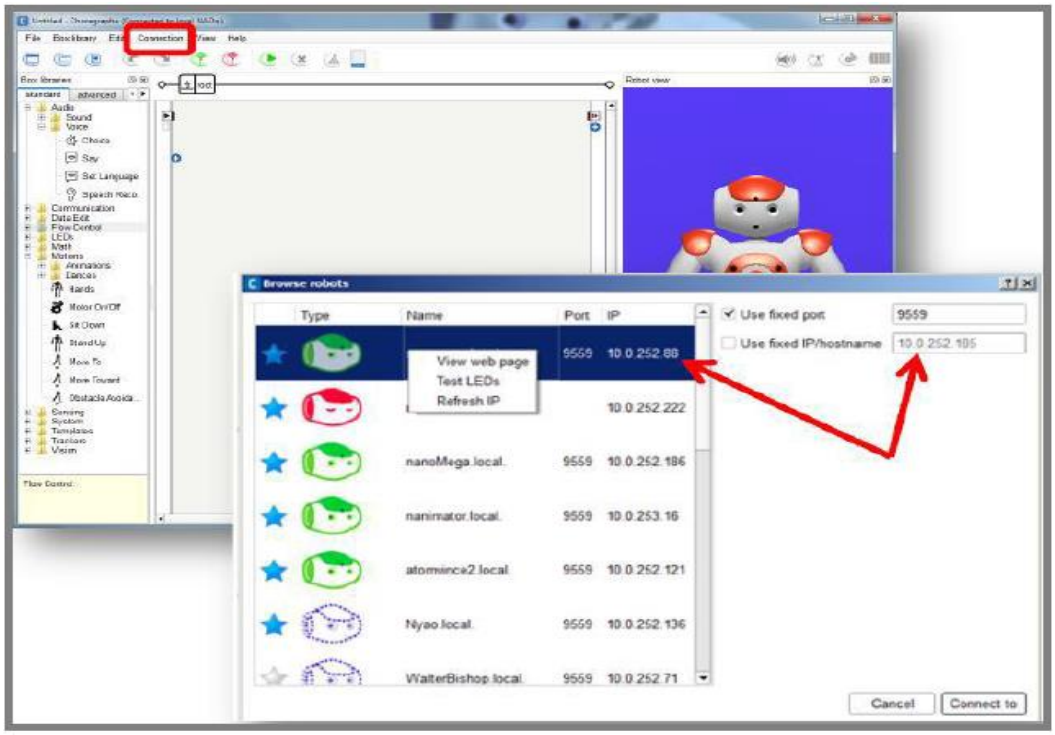
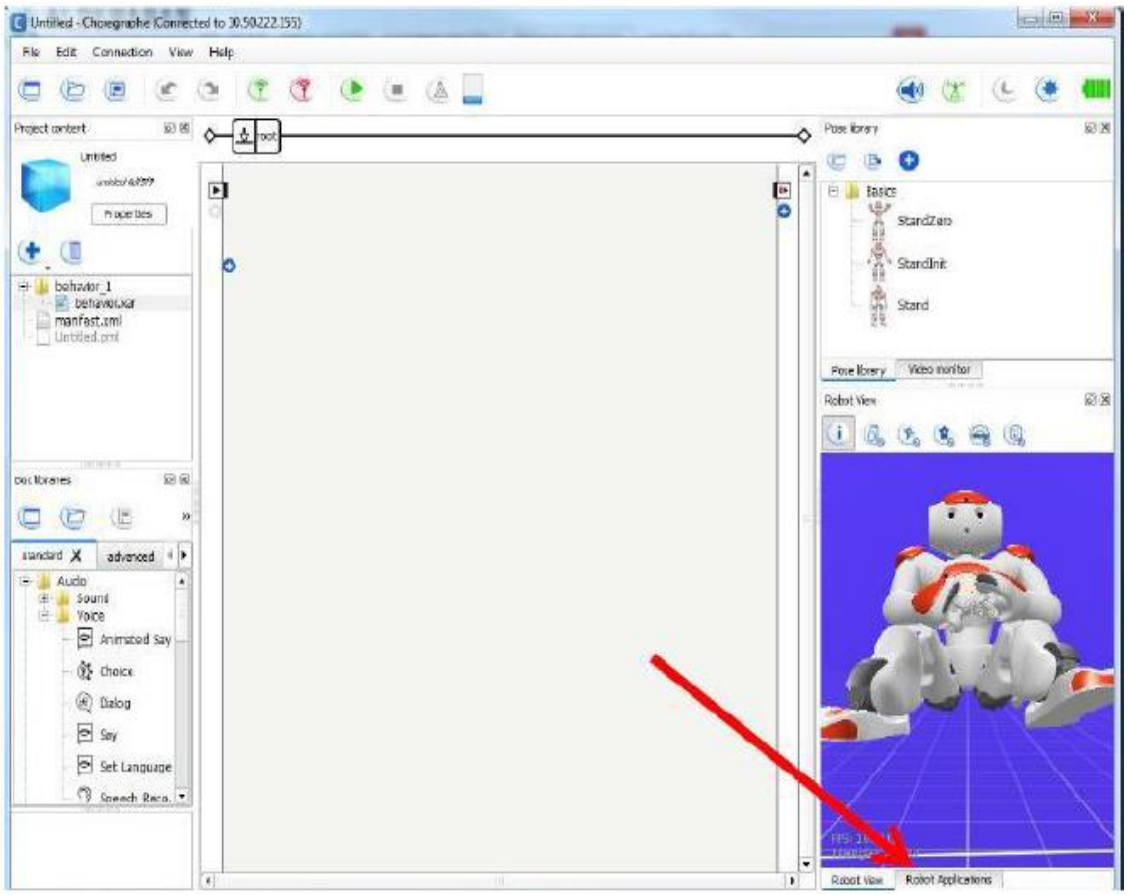


Figure II.23. Fenêtre de connexion à *NAO* via *Choregraphe.*

- On recherche notre robot. On peut s'y connecter en le sélectionnant ou en rentrant son adresse IP (sur la partie de droite) puis on clique sur le bouton "*Connect to*" (Le robot doit être sur le même réseau que l'ordinateur).
- Une fois *NAO* et *Choregraphe* connectés, on ouvre le panneau "*Robot Applications*".



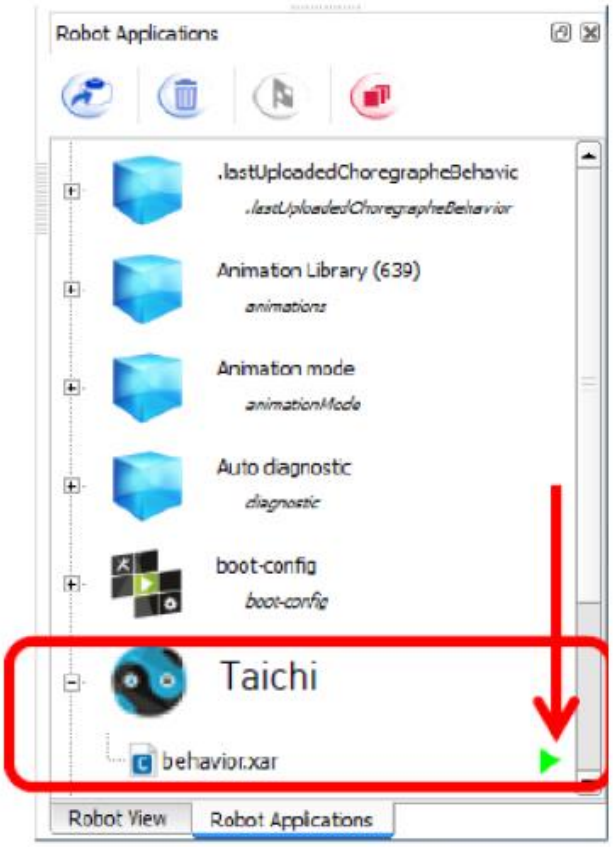


Figure II.24. Panneau « *Robot Applications* » illustrant les applications implémenté dans *NAO*.

Chaque ligne est un programme qui est actuellement installé. Les applications sont affichées avec leur logo et nom.

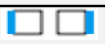
Pour chaque programme (donc chaque ligne), la flèche verte ▶ étant le bouton d'exécution, signifie que le programme est arrêté. A l'inverse, le carré rouge ■, étant le bouton d'arrêt, signifie que le programme est en cours d'exécution.

Si on souhaite avoir un programme qui démarre automatiquement, on sélectionne ce dernier et on clique sur le bouton avec un drapeau

II.8.4. Description des différentes entrées/sorties de Choregraphe

Les différentes entrées/sorties permettant de créer des comportements et animations en utilisant *Choregraphe* sont décrite dans le tableau ci-dessous :

Tableau II.2. Description des symboles d'entrées / sorties.

Symbole	Nature	Description
Entrées		
	onStart	Si l'entrée est stimulée, la boite est démarrée.
	onStop	Si cette entrée est exécutée, la boite est arrêtée.
	onEvent	C'est une entrée qui n'a pas d'effet spécifique sur la boîte, il ne démarre et ne stoppe rien. Quand il est stimulé : La fonction *onInput_<input_name>* du script de la boîte est appelée. Le signal reçu sur l'entrée est transmis au diagramme de la boîte.
	ALMemory Input	C'est une entrée qui n'est visible que dans le schéma. Donc, vous ne pouvez pas la stimuler de l'extérieur de la boîte. Elle est stimulée à chaque fois que la valeur des données stockées dans *ALMemory* correspondant à cette entrée est mise à jour et qu'un événement est déclenché (mis à jour).
	onLoad	C'est une entrée qui n'est visible que dans le schéma et quand la boîte est de type *Timeline*. Donc, on ne peut pas la stimuler de l'extérieur de la boîte. Elle est seulement stimulée lorsque le diagramme a été chargé.
Sorties		
	onStopped	Lorsque cette sortie est stimulée (à partir du *Timeline*,du *Dialog* ou à partir du diagramme), la boîte s'arrête (désactivée). Par contre elle n'a aucun effet sur la boite *Python*.
	punctual	Cette sortie transmet seulement le signal entre les diagrammes et les sous-diagrammes.
Types d'entrées / sorties		
	Bang	Représente un évènement simple et ne porte que l'information stimulée.
	Number	Représente un évènement portant des informations de types float (réel), int ou bien sous forme d'un tableau de nombres.
	String	Représente aussi un évènement portant des informations mais de type *String* ou d'un tableau de type chaines de caractères.
	Dynamic	Il représente les deux types d'évènements : simple ou porteur d'informations.

II.9. Boîte de scripte Python

Une boîte de scripte Python est une boite qui comprend uniquement un scripte. Pour en créer une, on clique sur le bouton droit de la souris où l'on veut sur le diagramme de flux et on choisit « *Create a new box* » après « *Python* ». Une fenêtre s'affichera, elle contient toutes les informations de la boîte comme son nom, le nombre d'entrée, de sortie, leur type ainsi que les paramètres.

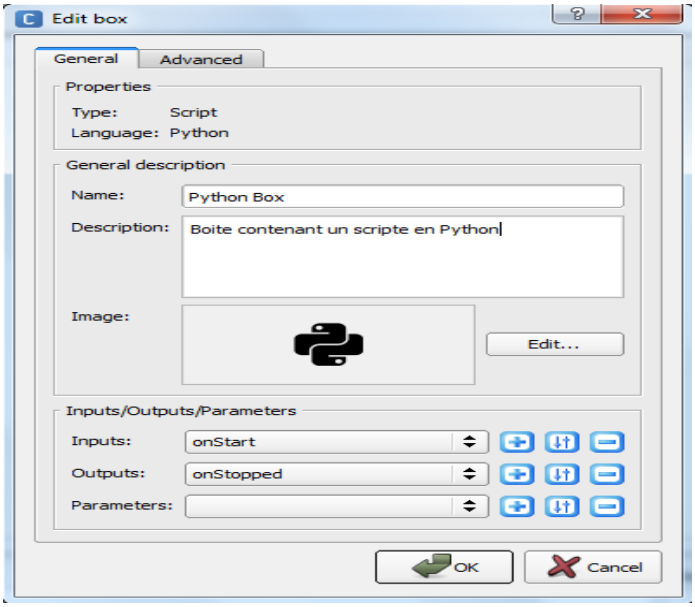


Figure II.25. Fenêtre d'édition de la boîte de scripte.

Après avoir configuré les informations nécessaires, on clique deux fois sur la boîte pour voir son contenu et éventuellement le modifier. Celle-ci contient quelques lignes de code en Python illustré dans la figure II.26.

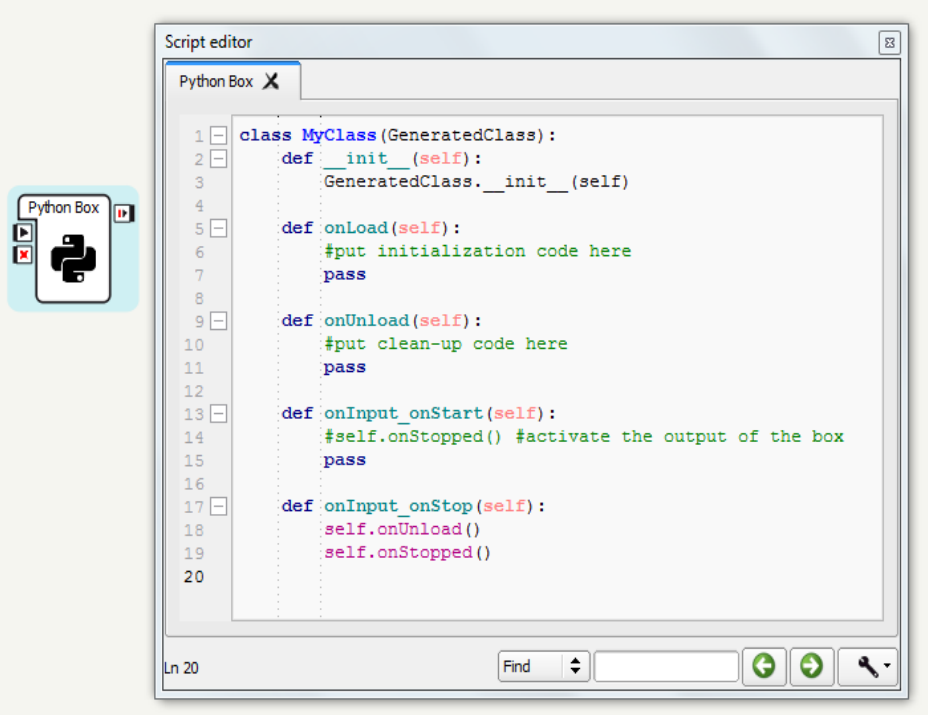


Figure II.26. Contenu de la boîte de scripte.

Le script définit une classe nommée « *MyClass* ». Cette classe hérite de « *GeneratedClass* » et représente la boîte. Elle a déjà quelques méthodes définies par défaut. « *GeneratedClass* » est une classe qui est automatiquement générée lors de l'exécution du scripte. Elle comprend toutes les informations nécessaires concernant la boîte de script (entrées, sorties, paramètres, etc.). Elle définit également toutes les fonctions intégrées supplémentaires du scripte box qui pourraient être utiles dans le script. Le fait que « *MyClass* » hérite de « *GeneratedClass* », nous permet d'utiliser toutes ces fonctions intégrées dans le script. De plus, comme « *MyClass* » est modifiable, on peut éventuellement ajouter nos propres fonctions. [Web 12]

Dans le script d'une boîte, de nombreuses méthodes sont disponibles permettant de gérer :

- Entrées.
- Sorties.
- Paramètres.
- Chronologie (*Timeline*).
- Événements de chargement et de déchargement.
- Des ressources.
- Proxies.
- Journaux (Logs).

II.9.1. Entrées [Web 12]

Les méthodes correspondant aux entrées de la boîte doivent être définies dans le script avec la syntaxe suivante : « *onInput_ [input-name]* »

- La définition de la méthode correspondant à une entrée de type « *Bang* » :

```
def onInput_myInput(self):
    # Le code de ma méthode
    pass
```

- La définition de la méthode correspondant à une entrée de type « *Number* » :

```
def onInput_myInput(self, number):
    # Le code de ma méthode
    pass
```

L'argument « *number* » est la valeur du signal envoyé sur l'entrée.

- La définition de la méthode correspondant à une entrée de type « *Dynamic* ».

```
def onInput_myInput(self, number = None):
    # Le code de ma méthode
    pass
```

Ici l'argument « *number* » peut être :

- Egal à la valeur du signal envoyé sur l'entrée ;
- Ou égal à « *None* », s'il est stimulé avec un signal de type « *Bang* ».

II.9.2. Sorties [Web 12]

Les méthodes correspondant aux sorties de la boîte ne sont pas présentes dans le script, mais sont déjà définies avec la syntaxe suivante : [*nom-sortie*].

La sortie est stimulée chaque fois qu'une méthode correspondant à une sortie est appelée.

- Stimulation de la sortie « *myOutput* » de type « *Bang* ».

```
self.myOutput()
```

- Stimulation de la sortie « *myOutput* » de type « *Number* » avec la valeur « *myNumber* ».

```
myNumber = 24
self.myOutput(myNumber)
```

II.9.3. Paramètres [Web 12]

La fonction « *Get parameter* » permet d'obtenir la valeur de n'importe quel paramètre de la boîte. Par l'acquisition de la valeur du paramètre « *My parameter* » de la boîte se fait comme suit :

```
value = self.getParameter("My parameter")
```

Cette fonction lèvera une exception si aucun paramètre de ce type n'existe**.**

II.9.4. Chronologie (*Timeline*) [Web 12]

Ces méthodes permettent d'affecter la chronologie de la boîte à partir de son script ou du script d'une boîte intérieure. Pour accéder à un objet *Timeline* en Python, on utilise :

- « *self.getTimeline()* » pour gérer la chronologie de la boîte actuelle ;
- Ou « *self.getParentTimeline()* » pour gérer la chronologie de la boîte parent.

II.9.5. Événements de chargement et de déchargement [Web 12]

Ces méthodes sont appelées automatiquement lorsque la boîte est chargée ou déchargée.

« *onLoad* » est appelé lorsque la boîte est chargée. Elle est définie comme suit :

```
def onLoad(self):
    # Mettre le code d'initialisation ici
    pass
```

« onUnload » est appelé lorsque la boîte est déchargée. Elle est définie comme suit :

```
def onUnload(self):
    # Mettre le code de réinitialisation ici
    pass
```

Il est conseillé d'appeler « *onUnload* » dans le code de la méthode « *onStop* » pour que la boîte de scripte soit réinitialisée lorsqu'elle est arrêtée.

II.9.6. Ressources

Pour définir les ressources que peut utiliser une boîte, on sélectionne celle-ci et on clique sur le bouton droit de la souris et on choisit « *Edit resources* ». Une fenêtre s'affichera qui contiendra les ressources disponibles et les actions à faire si celles-ci font l'objet d'appel d'une autre boîte :

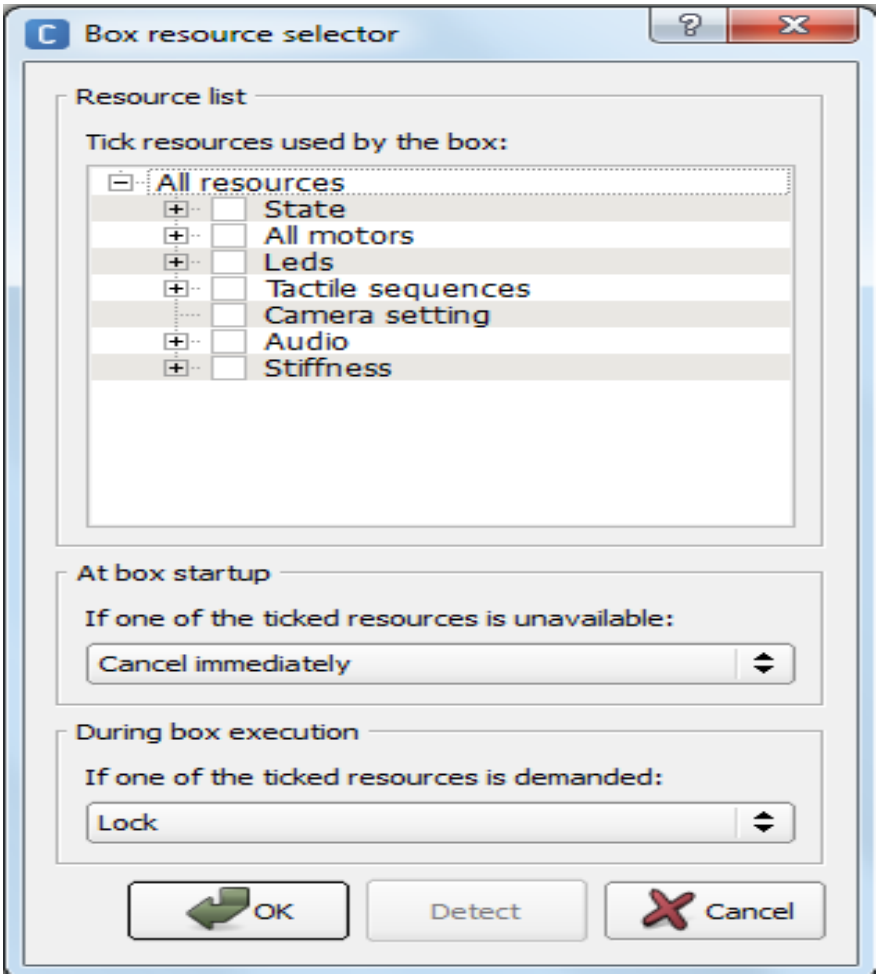


Figure II.27. Fenêtre des ressources.

La méthode « *onResource* » est appelée lorsque les ressources de la boîte sont définies sur « Rappel à la demande » et que les ressources sont demandées par une autre boîte. Pour appeler cette fonction, on doit la définir comme suit : [Web 12]

```
def onResource(self, resource):
    # Le code de ma méthode
    pass
```

L'argument « ressource » correspond au nom de la ressource demandée.

II.9.7. Proxies [Web 12]

« *ALProxy* » permet de créer un proxy sur un module de « *NAOqi* » puis d'utiliser facilement n'importe quelle méthode de ce module.

Par exemple, si on veut faire appel au module « *ALLed* », qui gère les leds de NAO, et utiliser l'une de ses méthodes, on procède comme suit :

```
leds = ALProxy("ALLeds")
t = 5.0 # temps de l'animation en seconde
leds.rasta(t)
```

Ce code permet de lancez une animation des leds, [vert, jaune, rouge], sur tout le corps.

II.9.8. Journaux (Logs) [Web 12]

Ces méthodes permettent d'afficher des messages sur la fenêtre débogage (diagnostique) fournissant des informations sur le déroulement et l'évolution du programme exécuté.

L'affichage d'un message dans le langage Python se fait avec « *Print* ». Mais, dans notre cas, on utilise des fonctions spécifiques détaillées dans le tableau II.3.

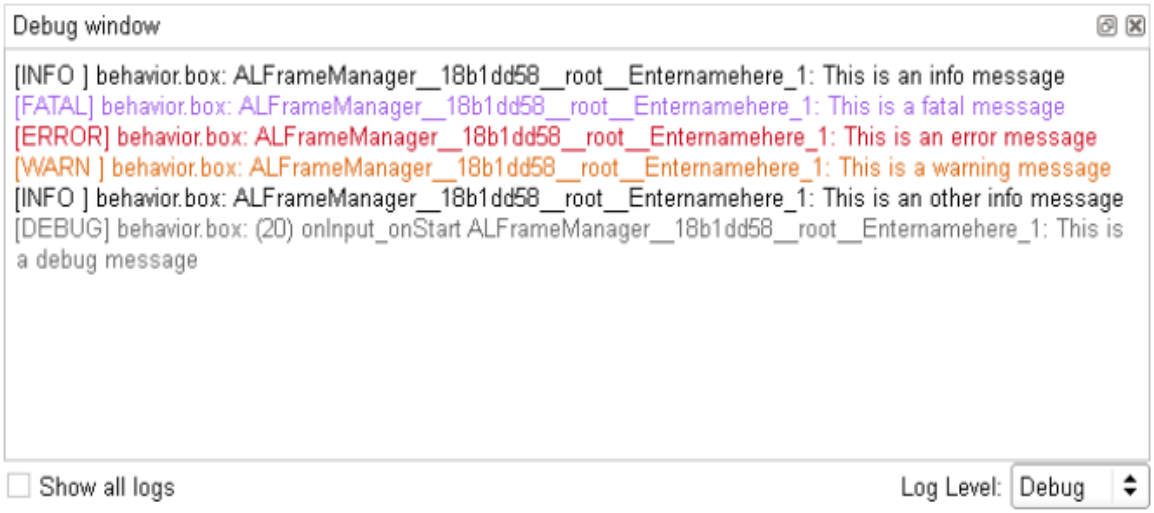
Tableau II.3. Description des fonctions d'affichage.

Fonction	La description
self.log("mon message")	Message avec un niveau d'**information** de gravité.
self.logger.fatal("mon message")	Message avec un niveau de gravité **fatal**.
self.logger.error("mon message")	Message avec un niveau d'**erreur** de gravité.
self.logger.warning("mon message")	Message avec un niveau d'**avertissement** de gravité.
self.logger.info("mon message")	Message avec un niveau d'**information** de gravité.
self.logger.debug("mon message")	Message avec un niveau de gravité de **débogage**.

- Exemple d'utilisation des méthodes de « logs » :

```
def onInput_onStart(self):
    self.log("This is an info message")
    self.logger.fatal("This is a fatal message")
    self.logger.error("This is an error message")
    self.logger.warning("This is a warning message")
    self.logger.info("This is an other info message")
    self.logger.debug("This is a debug message")
```

- Résultat lors l'exécution du code :



II.9.9. Importation des modules Python

Si on veut faire un traitement spécifique des données dans une boîte de scripte, tel qu'un calcul mathématique par exemple, on peut utiliser des modules de Python en les important avec l'instruction « *import* » au début du script.

- Exemple d'importation du module « math » dans la boîte de script :

```
# -*- encoding: UTF-8 -*-
import math #Importer le module math

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        # Mettre le code d'initialisation ici
        pass

    def onUnload(self):
        # Mettre le code de réinitialisation ici
        pass

    def onInput_onStart(self):
        self.res = math.sqrt(16)#Calcul de la racine carré
        self.onStopped(self.res)#Affectation du résultat à la sortie de boîte
        pass

    def onInput_onStop(self):
        self.onUnload()
        self.onStopped()

```

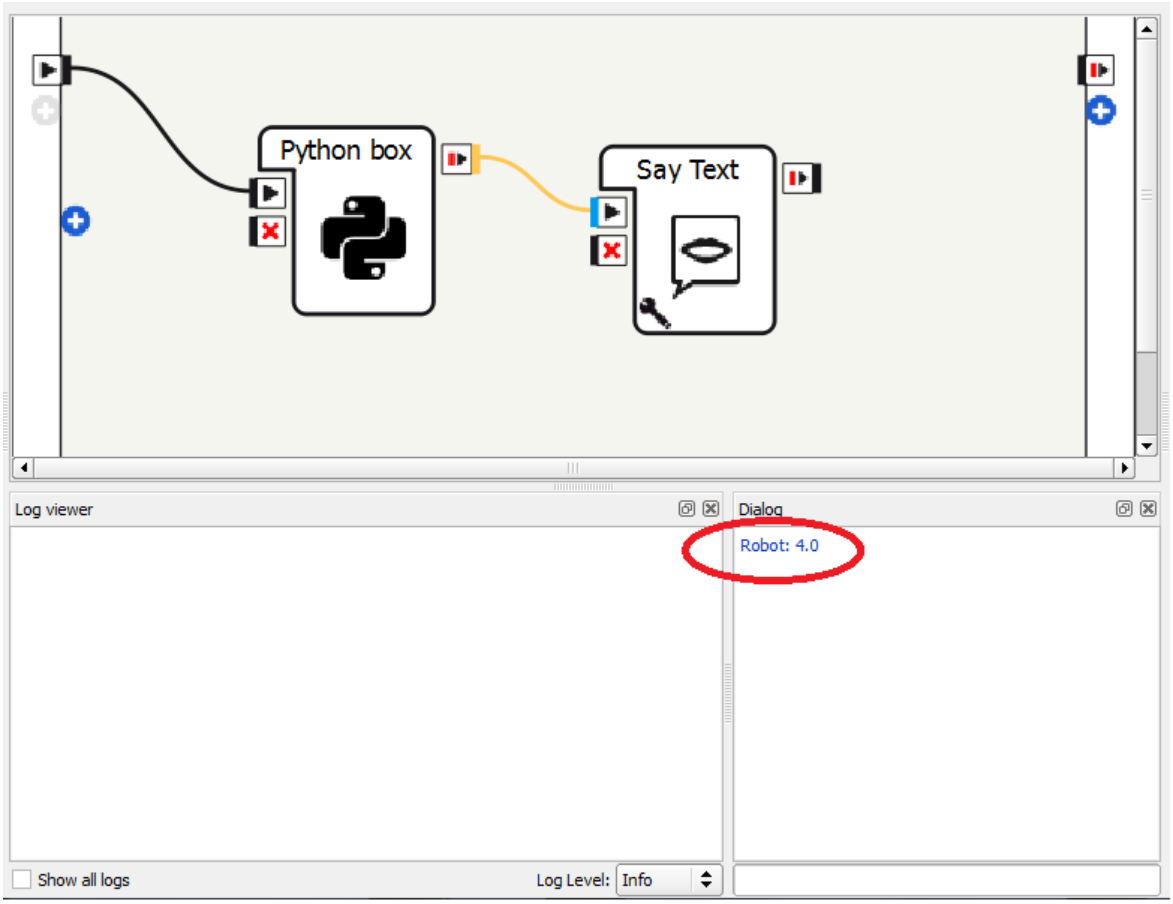


Figure II.28. Résultat d'exécution du script de l'exemple.

II.10. Conclusion

Dans ce deuxième chapitre, on a introduit le robot humanoïde NAO, objet de notre étude. D'abord on a commencé par présenter la société SoftBank Robotics, conceptrice de NAO, et décrit l'architecture et les composants du robot, ses capteurs et ses actionneurs et leurs caractéristiques techniques, ainsi que la mise en marche et la configuration du robot. Enfin, on s'est intéressé au logiciel Choregraphe, outil nous permettant de le manipuler et d'exploiter ses ressources, ainsi qu'à la boite script qui contient des lignes de code en langage Python.

Chapitre III : La caméra 2D de NAO

III.1. Introduction

Dans ce chapitre, nous nous focaliserons sur les deux caméras de *NAO* et consacrerons une partie aux espaces colorimétriques pris en charge par celles-ci, ainsi qu'une brève description de la bibliothèque *OpenCV*. Enfin, nous terminerons par la présentation du *Monitor*.

III.2. Caméra 2D

Comme mentionné dans le chapitre précédent, le robot *NAO* possède deux caméras identiques sur la face (au niveau du front et de la bouche). Elles offrent une résolution jusqu'à 1280x960 à 30 images par seconde (**ips**) et peuvent être utilisées pour l'identification de différents objets présents sur leur champ visuel respectif ou bien encore pour l'enregistrement vidéo.

L'image de l'objet, fournie par la caméra 2D de *NAO* pour le traitement, est en plan bidimensionnelle plane. Autrement dit, on aura les deux données x et y, mais pas la donnée de profondeur de champ sur l'axe z.

III.3. Spécifications [web 12]

Tableau III.1 résume les caractéristiques importantes de la camera 2D.

Tableau III.1. Spécification de la caméra 2D de NAO.

Caméra	**Modèle**	**MT9M114**
	Type	Capteur d'image SOC
Réseau d'imagerie	Résolution	1,22 Mp
	Format optique	1/6 pouce
	Pixels actifs (HxL)	1280 x 960
Sensibilité	Taille de pixel	1,9 µm * 1,9 µm
	Plage dynamique	70 dB
	Rapport signal / bruit (max)	37 dB
	La réactivité	2,24 V / Lux-sec (550 nm)
Production	Sortie caméra	1280 * 960 à 30 images par seconde
	Format des données	(Espace colorimétrique YUV422)
	Type d'obturateur	Volet roulant électronique (ERS)
Vue	Champ de vision	72,6 ° DFOV (60,9 ° HFOV, 47,6 ° VFOV)
	Plage de mise au point	30 cm ~ infini
	Type de mise au point	Mise au point fixe

III.4. Résolutions [web 12]

Tableau III.2 résume les résolutions qui sont prise en charge par la caméra MT9M114.

Tableau III.2. Différentes résolutions de la caméra 2D de NAO.

Nom de l'ID de paramètre	Valeur ID	La description
AL :: kQQQQVGA	8	Image de 40 * 30 px
AL :: kQQQVGA	7	Image de 80 * 60 px
AL :: kQQVGA	0	Image de 160 * 120 px
AL :: kQVGA	1	Image de 320 * 240 px
AL :: kVGA	2	Image de 640 * 480 px
AL :: k4VGA	3	Image de 1280 * 960 px

III.5. Résolutions Vidéo

La résolution vidéo (frames rates) est le nombre d'images par seconde (*ips* ou *fps* en anglais pour *frame per second*) prise en charge par une caméra. Celle de *NAO* se limite à 30 ips, pour toutes les résolutions citées précédemment.

Tableau III.3. Fréquences vidéo prises en charge en fonction de la résolution définie **[Web 12]**

Résolution	Frame rate pris en charge
Image de 40 * 30 px	de 1 à 30 fps
Image de 80 * 60 px	de 1 à 30 fps
Image de 160 * 120 px	de 1 à 30 fps
Image de 320 * 240 px	de 1 à 30 fps
Image de 640 * 480 px	de 1 à 30 fps
Image de 1280 * 960 px	de 1 à 30 fps

III.6. Connexion aux caméras de NAO

Lorsqu'on se connecte à *NAO*, on voit instantanément ce que sa caméra frontale voie. Mais, comme il possède une autre caméra au niveau de la bouche, on peut changer alors les images qui apparaissent sur *Choregraphe*. Pour cela, on va simplement utiliser la boîte « *Select Camera* ».

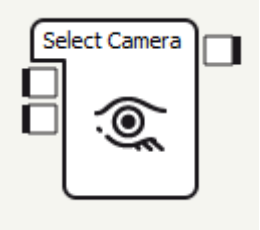


Figure III.1. Boîte « Select Camera ».

On ne peut faire qu'un seul traitement à la fois, et donc, on doit être conscient de la caméra choisie.

A l'intérieure de cette boîte, on trouve les quelques lignes de code en Python de Figure III.2.

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)
        self.kCameraSelectID = 18
        self.cameraModule = ALProxy( "ALVideoDevice" )

    def onLoad(self):
        pass

    def onUnload(self):
        pass

    def onInput_onUseTopCamera(self):
        self.cameraModule.setParam( self.kCameraSelectID, 0 )
        self.onReady()

    def onInput_onUseBottomCamera(self):
        self.cameraModule.setParam( self.kCameraSelectID, 1 )
        self.onReady()
```

Figure III.2. Code Python de la boîte « Select Camera ».

Comme on peut le voir, on a deux fonctions, « *onInput_onUseTopCamera* » et « *onInput_onUseBottomCamera* » qui permettent de choisir la caméra haute ou basse respectivement. Elles correspondent aux entrées de la boite « *Select camera* ».

L'identifiant de la caméra frontale est « 0 », et la caméra en bas est « 1 ».

Tableau III.4. Indices des caméras de *NAO* **[Web 12]**

Nom de l'ID de paramètre	Valeur ID	La description
AL :: kTopCamera	0	caméra sur le dessus.
AL :: kBottomCamera	1	caméra en bas.

Si on veut créer notre propre programme en python pour le traitement d'images, il est nécessaire de se connecter aux caméras de *NAO*, pour récupérer les images et faire les manipulations. Pour ce faire, on fait appel au module « *ALVideoDevice* » propre à « *Naoqi* », qui est le *framework* du robot *NAO*. On définit la caméra qu'on veut utiliser via son indice, la résolution, l'espace colorimétrique et le nombre d'ips.

La connexion à la caméra 2D de NAO avec Python se fait comme suit :

```
def connectToCamera(self):
        """
        Se connecter a la camera de NAO
        """
        try:
            self.avd = ALProxy("ALVideoDevice")
            strMyClientName = self.getName()
            nCamera = 0  # "0" pour la caméra en haut et "1" for la caméra en bas
            nResolution = 2  # "2": Resolution KVGA 640*480
            nColorSpace = 13  # "10" : YUV, "11" : BGR, "13" : RGB
            nFPS = 30  # de 1 à 30 ips

            self.strMyClientName = self.avd.subscribeCamera(strMyClientName, nCamera, nResolution, nColorSpace, nFPS)
        except BaseException, err:
            self.log("Err: connectToCamera : catching error: " + str(err))

```

Figure III.3. Code Python de connexion à une caméra de Nao.

Après avoir terminé le traitement, On déconnecte les caméras pour ne plus recevoir les images prise par celles-ci. Pour cela, on se désabonne de « *ALVideoDevice* » selon le code de Figure III.4.

```
def disconnectFromCamera(self):
        """
        Se deconnecter de la camera de NAO
        """
        try:
            self.avd.unsubscribe(self.strMyClientName)
        except BaseException, err:
            self.log("Err: disconnectFromCamera : catching error: " + str(err))

```

Figure III.4. Code Python de déconnexion à la caméra de Nao.

III.7. Espace colorimétrique

Un espace colorimétrique est un modèle décrivant la manière de représenter la couleur sous la forme d'une liste de nombres (vecteur). Par exemple, on peut représenter la couleur de chaque pixel d'un écran sous la forme d'une liste de trois éléments R, V et B, qui sont respectivement les valeurs du Rouge, du Vert et du Bleu généralement stockées dans un octet (de 0 à 255).

La caméra 2D de NAO prend en charge plusieurs espaces de couleur : YUV, RGB et HSY. Ces derniers peuvent être utilisés de plusieurs façons, selon le besoin, et dépendent de la valeur du paramètre de l'espace colorimétrique comme indiqué dans le tableau III.5.

III.7.1. Espace de couleur YUV

L'espace colorimétrique YUV a trois composantes. La composante Y détermine la luminosité de la couleur (appelée luminance), tandis que les composantes U et V déterminent la couleur elle-même (la chrominance). Y va de 0 à 1 (ou 0 à 255 dans les formats numériques), tandis que U et V vont de -0,5 à 0,5 (ou -128 à 127 sous forme numérique signée, ou 0 à 255 sous forme non signée).

III.7.2. Espace de couleur RGB

L'espace RGB est l'espace de couleurs le plus utilisé, puisqu'il intervient dans la plupart des appareils de prise d'images couleurs, ainsi que dans les moniteurs couleur. La plupart des formats de codage d'images utilisent cet espace de couleurs. Il a trois composante R pour le rouge, G pour le vert (green en anglais) et B pour le bleu.

Tableau III.5. Indces des espaces colorimétriques **[Web 12].**

Nom de l'ID de paramètre	Valeur ID	Nombre de canaux	La description
AL :: kYuvColorSpace	0	1	Le tampon ne contient que le Y (composant luma)
AL :: kyUvColorSpace	1	1	Le tampon ne contient que le U (composant Chrominance)
AL :: kyuVColorSpace	2	1	Le tampon ne contient que le V (composant Chrominance)
AL :: kRgbColorSpace	3	1	Le tampon ne contient que le R (composant rouge)
AL :: krGbColorSpace	4	1	Le tampon ne contient que le G (composant vert)
AL :: krgBColorSpace	5	1	Le tampon ne contient que le B (composant bleu)
AL :: kHsyColorSpace	6	1	Le tampon ne contient que le H (composant Hue)
AL :: khSyColorSpace	7	1	Le tampon ne contient que le S (composant de saturation)
AL :: khsYColorSpace	8	1	Le tampon ne contient que le Y (composant de luminosité)
AL :: kYUV422ColorSpace	9	2	Format natif, 0xY'Y'VVYYUU.
AL :: kYUVColorSpace	10	3	Le tampon contient un triplé au format 0xVVUUYY
AL :: kRGBColorSpace	11	3	Le tampon contient un triplet au format 0xBBGGRR
AL :: kHSYColorSpace	12	3	Le tampon contient un triplet au format 0xYYSSHH
AL :: kBGRColorSpace	13	3	Le tampon contient un triplet au format 0xRRGGBB
AL :: kYYCbCrColorSpace	14	2	Format TIFF, quatre caractères non signés pour deux pixels.
AL :: kH2RGBColorSpace	15	3	H de «HSY à RVB» dans de fausses couleurs.
AL :: kHSMixedColorSpace	16	3	HS et (H + S) / 2.

Figure III.5. Modèle RGB.

III.7.3. Espace de couleur HSY [Web 12]

Ii est inspiré des espaces colorimétriques HSV et HSL et optimisé pour la vitesse sur les systèmes embarqués.

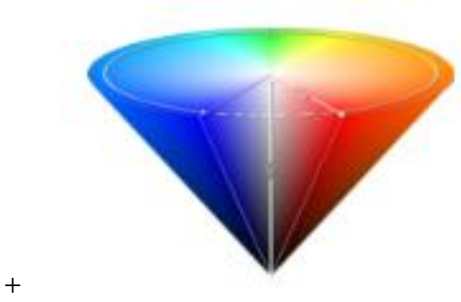


Figure III.6. Modèle HSY.

- Dans l'espace colorimétrique HSY, la composante H (teinte) est codée de 0 à 255 pour représenter les 360 °. Ainsi, chaque unité représente 1,406 ° et la teinte peut être stockée sur 8 bits.
- La saturation (S) utilisée est celle du système HSV, car elle est plus rapide à calculer.
- Comme la caméra possède la caractéristique de luminance qui est un composant natif de celle-ci, alors il a été préféré à la valeur (V) et à la luminosité (L).

III.8. OpenCV

OpenCV, pour « *Open Computer Vision* », est une librairie graphique spécialisé dans le traitement d'images pour la photo ou la vidéo. Elle est disponible sur la plupart des systèmes d'exploitation et existe pour les langages Python, C++ et Java.

Figure III.7. Logo de la bibliothèque *OpenCV*.

On l'a notamment utilisée pour développer quelques applications de reconnaissance avec le robot *NAO* en langage Python, qu'on verra dans le prochain chapitre.

Pour faire appel à cette bibliothèque en Python, il suffit simplement de l'importer et pas besoin d'exécuter une quelconque installation car « *NAOqi* » prend en charge *OpenCV*.

Il est à noter que comme le module « *ALVideoDevice* », la bibliothèque *OpenCV* ne peut être utilisée avec le robot virtuel, bien qu'il soit possible de faire des manipulations en se connectant à une webcam, chose qu'on a faite pour faire des tests.

```python
import cv2
from naoqi import ALProxy

ip = raw_input('Adresse Ip : ')
port = int(raw_input('Port : '))

camProxy = ALProxy("ALVideoDevice", ip, port)

vc = cv2.VideoCapture(0)

while vc.isOpened():
    check, frame = vc.read()
    frame = cv2.resize(frame, (640, 480))  # (source, dimension)
    b, g, r = cv2.split(frame)  # obtenir b,g,r
    rgb_img = cv2.merge([r, g, b])  # b,g,r --> r,g,b
    """
    putImage(Index, Largeur, Hauteur, ImageBuffer)
    """
    camProxy.putImage(0, 640, 480, rgb_img.tobytes())
    key = None
    if key == 27:
        break
```

Figure III.8. Programme en Python.

Ce programme permet d'utiliser la webcam et d'afficher l'image sur le **Vidéo monitor** du *Choregraphe* grâce à *OpenCV*. Ainsi, on a pu faire appel au module « *ALVideoDevice* » qui n'est normalement utilisé qu'avec le robot réel.

III.9. Monitor [Web 12]

Monitor est dédié à donner un retour élémentaire de votre robot et un accès simple à ses paramètres de caméra. Il est notamment installé avec *Choregraphe*.

Son architecture modulaire permet de charger des plugins (paquets qui complètent le logiciel hôte) dans différents widgets mobiles, chacun d'entre eux étant connecté au robot de notre choix. C'est-à-dire qu'on peut se connecter à plusieurs robots à la fois.

Trois (03) plugins sont disponibles :

- Moniteur laser.
- Visionneuse de la mémoire.
- Visionneuse de la caméra.

Figure III.9. Fenêtre d'accueil de Monitor.

III.9.1. Monitor laser [Web 12]

Cette fonction n'est disponible que pour les propriétaires de têtes laser NAO. Ce plugin permet d'afficher ce qui est vu par le télémètre laser.

III.9.2. Monitor de la mémoire [Web 12]

Ce plugin permet de visualiser les données détenues par le module « *ALMemory* » d'un « *NAOqi* » donné. L'évolution des données de type numérique peut également être tracée. Ceci est très utile pour diagnostiquer les comportements à travers les données internes qu'ils utilisent.

III.9.3. Monitor de la caméra [Web 12]

Le visionneur de la caméra permet de :

- Configurer les caméras robot.
- Afficher ce qu'ils voient.
- Prendre des photos.
- Ou d'enregistrer des vidéos.

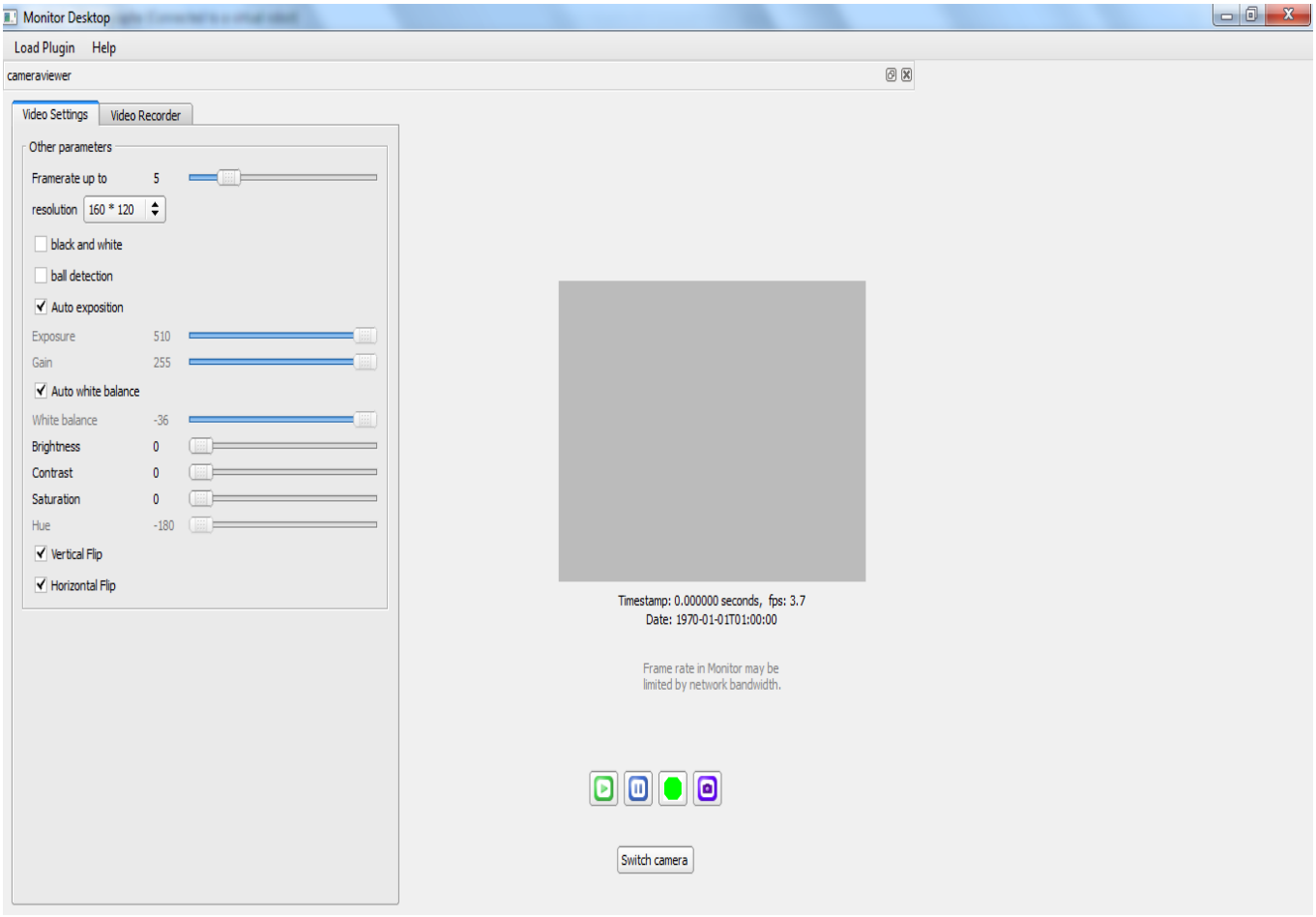


Figure III.10. Fenêtre d'accueil du visionneur de la caméra.

- Pour démarrer la récupération et l'affichage des images de *NAO* dans le widget vidéo, on clique sur « **Play** »  .
- « **Pause** »  met en pause l'affichage vidéo et arrête de demander des images.
- **« Record »**  démarre l'enregistrement d'une vidéo.  Arrêtez l'enregistrement de la vidéo et demande un fichier de destination.

- **« Snapshot »**  prend trois photos (a, b, c) de ce que voit le robot. Les fichiers qui seront enregistrés dans :
  /home/nao/local/share/naoqi/vision/{Timestamp} {a,b,c}.jpg

Avec :

Timestamp : horodatage des photos.

Ce plugin s'est avéré très utile pour diagnostiquer des « petits » problèmes. Par exemple, si on crée un module qui permet de détecter la position d'un ballon sur un terrain de football et que lorsqu'on exécute le programme rien ne se passe, on lance le plugin et on clique sur « **Play** » pour voir à travers les yeux du robot. On constate que c'est la balle, qui est aux pieds de votre robot, ne peut pas être vue aussi facilement par la caméra supérieure. À la place, la caméra inférieure serait certainement utile.

III.10. Conclusion

Après avoir pris connaissance des différentes caractéristiques de la caméra 2D du robot *NAO* et de la méthode de connexion à celles-ci ainsi que du logiciel *Monitor* qui lui est dédié, nous consacrerons le chapitre IV aux différentes applications réalisées avec le langage Python et le logiciel *Choregraphe*.

Chapitre IV : Applications

IV.1. Introduction

Dans ce chapitre, nous présenterons les applications réalisées avec le robot *NAO* et sa caméra 2D. Ces applications ont été faites avec le langage Python ainsi qu'avec les blocs préprogrammés fournis par le logiciel *Choregraphe*. Bien évidemment, les applications faites ne nécessitent pas la mise en marche des moteurs du robot, donc le mode repos « *Rest* » a été activer pour économiser sa batterie.

IV.2. Application 1 : Apprendre à NAO à reconnaitre des objets

NAO possède un système d'apprentissage d'objets, de livres et d'emplacements. Pour cela, il suffit d'utiliser le panneau du moniteur vidéo.

Nous avons suivi des étapes pour réaliser cette application. D'abord, il est question de faire apprendre à *NAO* à reconnaitre un ou (des) objet(s), livre(s), ou emplacement(s).

IV.2.1. Partie apprentissage

1. On vérifie d'abord que *NAO* est bien connecté au logiciel *Choregraphe*
2. Dans « **View** » on sélectionne « **Video monitor** » le panneau Moniteur vidéo apparaît et affiche ce que la caméra active voit.
3. Sur le « **Vidéo monitor** » on appuie sur l'icône « **Learn** ».
4. Un compte à rebours de 4 secondes commence. Il donne le temps de placer correctement l'objet que l'on veut faire apprendre à *NAO*. À 0 secondes, l'image passe en résolution QVGA et est capturée.

Figure IV.1. Emplacement de l'icône « **Learn** » sur le panneau « **Video monitor** ».

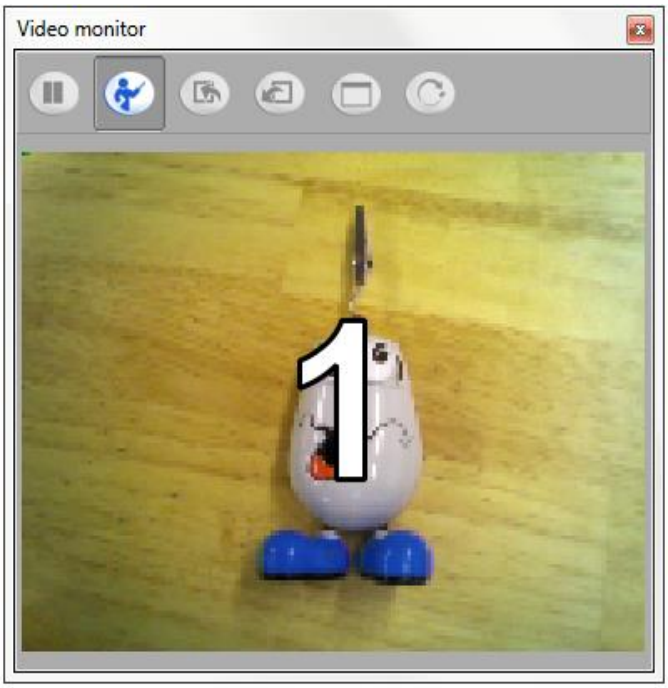


Figure IV.2. Compte à rebours lancé.

5. On clique sur une extrémité de l'objet en question pour dessiner, segment par segment, le contour de l'objet que l'on veut faire apprendre à *NAO*.

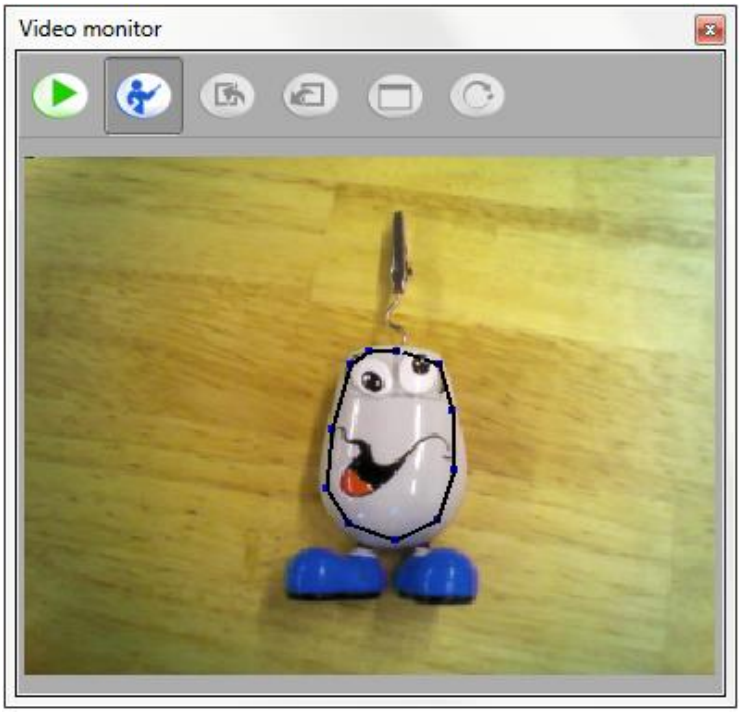


Figure IV.3. Dessin du contour.

6. Dès qu'on clique à nouveau sur le premier point pour fermer la forme, une fenêtre pop-up s'ouvre.

Figure IV.4. Définition de l'étiquette de l'objet.

On saisit les informations concernant l'objet, son nom et le coté dans lequel on l'a pris en photo

7. Ensuite on clique sur OK. Un message de retour confirme si l'apprentissage est réussi ou non (due en raison à la mauvaise qualité de l'entrée).
8. Si on veut faire apprendre de nouveaux objets à *NAO*, il suffit alors de cliquer sur le bouton « **Play** » sur le panneau du « **Video monitor** » et de refaire les mêmes opérations à partir de la troisième étape.
9. Lorsque nous en avons fini avec l'apprentissage de *NAO*, on a deux possibilités qui s'offrent à nous :
    - Soit on clique sur le bouton signifiant que l'on souhaite envoyer la base de données actuelle sur la reconnaissance d'objet à *NAO*.
    - Ou on choisit d'exporter (envoyer) ladite base de données en cliquant sur le bouton dans le but de la sauvegarder sur notre PC pour une utilisation ultérieure.

IV.2.2. Partie reconnaissance d'objet

Une fois la base de données sur la reconnaissance d'objet crée et lancée sur le robot, *NAO* peut alors reconnaitre les objets définis dans la base de données. Pour exécuter la reconnaissance, on suit ces quatre étapes :

1. D'abord on s'assure que la connexion avec *NAO* est faite.

2. On vérifie que l'on a lancé une base de données de reconnaissance visuelle sur le robot.

   Si ce n'est pas fait :

   - On crée une base de reconnaissance visuelle ou on importe une précédemment enregistrée dans le PC. L'importation dans *NAO* se fait en cliquant sur le bouton  se trouvant sur le panneau du *Video monitor*.
   - On clique sur le bouton afin d'envoyer la base de données sur la reconnaissance visuelle à *NAO*.

3. On insère la boite (*Box*) « *Vision Reco* » sur l'interface de travail de *Choregraphe*, qui permet à *NAO* de reconnaître les livres, objets et les emplacements définie dans sa base de données.

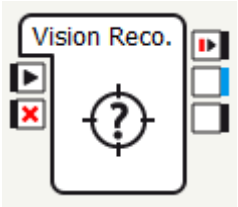


Figure IV.5. Boîte « *Vision Reco* ».

4. On lance l'application afin que *NAO* reconnaisse l'objet en se basant sur la base de données implémentée.

Notre application *Choregraphe* implémentée dans *NAO* est représentée par Figure IV.6.

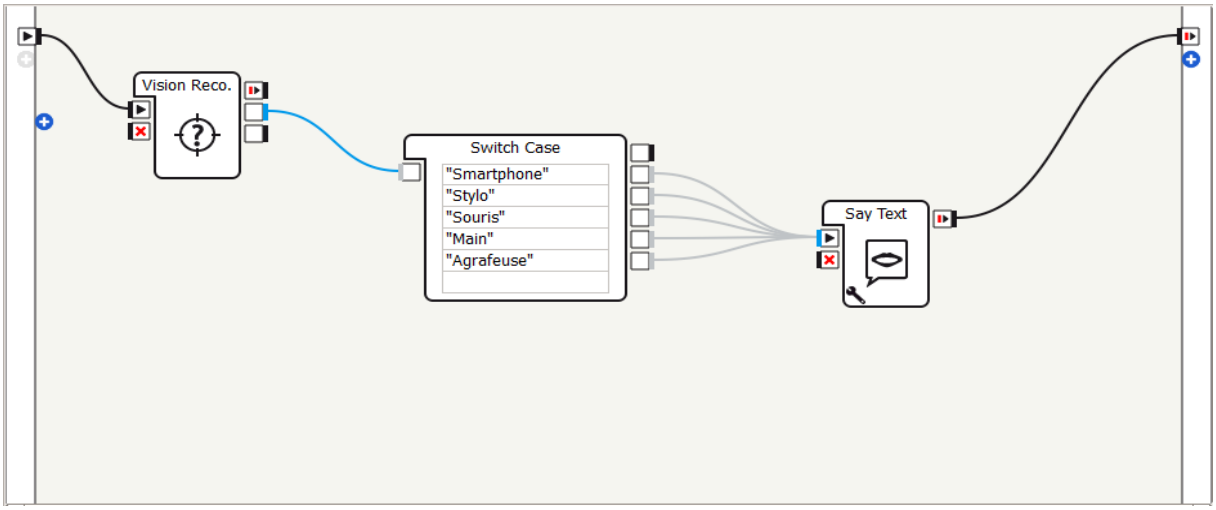


Figure IV.6. Programmation de reconnaissance d'objets.

IV.2.3. Remarques

- Dans la partie apprentissage, l'objet ne doit pas être uniforme en termes de couleurs et doit être différent de l'arrière-plan, il est donc conseillé d'avoir une même couleur. Dans le cas contraire un message d'erreur s'affichera comme l'illustre la figure IV.7.

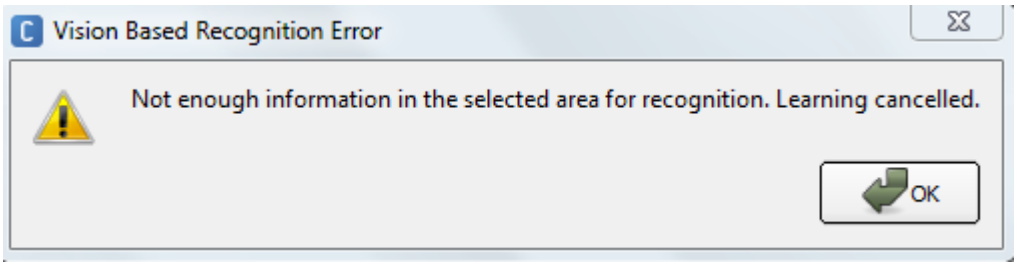


Figure IV.7. Message d'erreur.

- La luminosité de la salle doit être maîtrisée et l'objet vu par NAO doit être le plus claire possible donc il doit être à proximité de sa caméra.
- Lors de la partie reconnaissance, on a constaté que si l'objet est retourné ou incliné de sa position d'apprentissage, le robot ne le reconnait pas.

IV.3. Application 2 : Détection des expressions faciales

NAO peut déterminer l'expression faciale d'une personne ou d'une image. Pour cela, Choregraphe dispose de la boite « *get expression* » qui renvoie l'expression faciale détectée de la personne qui se trouve devant le robot.

Toutefois la détection ne marche que sur une personne à la fois. Il est possible de définir le seuil de confiance et les paramètres de temporisation pour cette boite.

Dans notre application, le but est que NAO détecte l'expression d'un visage humain. Pour ce faire, on utilise tout d'abord la boite « *Basic Awarness* ».

Figure IV.8. Boîte « *Basic Awarness* ».

Pour que NAO ne traque que les visages humains, on ne sélectionne dans les paramètres de la boite « *Basic Awarness* » que la case *People Stimulus*. On met le mode d'engagement en

complètement engagé. Comme la détection du visage ne va concerner que les caméras de NAO, on choisit la tête du robot comme mode de suivi.

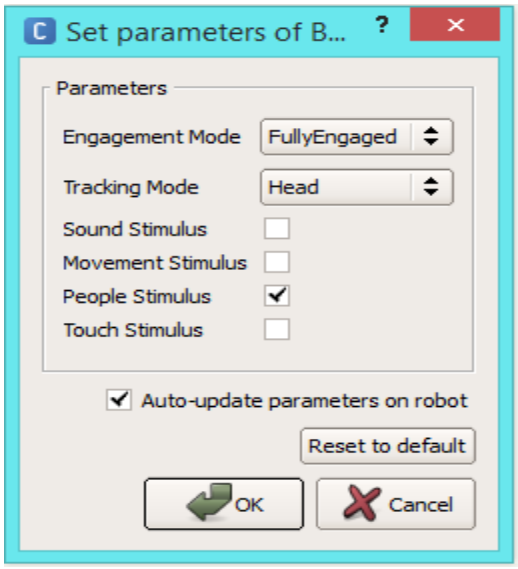


Figure IV.9. Paramètres de la boîte « *Basic Awarness* ».

A travers la boite « *Get Expression* » NAO tente d'identifier l'expression faciale de la personne ou de l'image de la personne qui se trouve devant lui.

Figure IV.10. Boite « *Get Expression* ».

Dans la partie paramètre de la boite en bas à gauche, On sélectionne les émotions que l'on souhaite détecter.

Les émotions possibles déterminables par *NAO* sont :

- La neutralité
- La joie
- L'étonnement
- La colère
- La tristesse

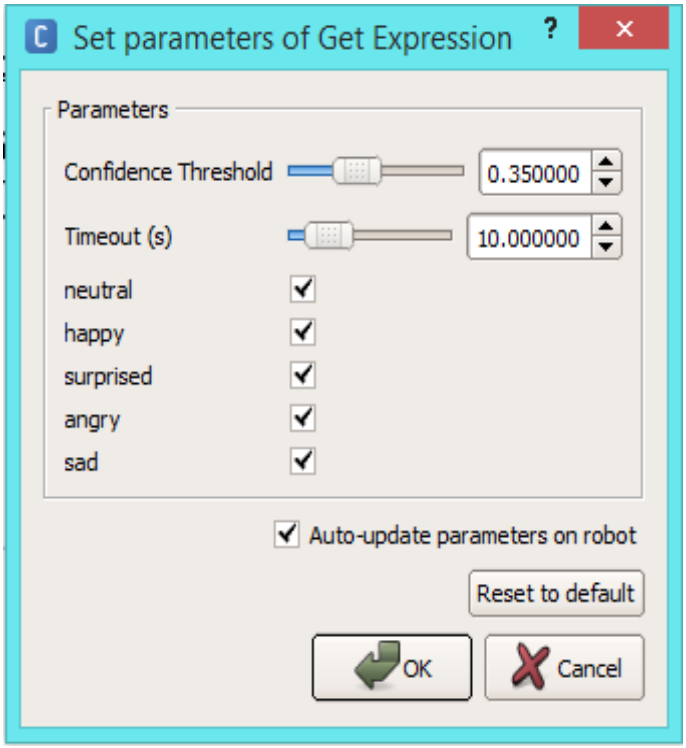


Figure IV.11. Gestion des paramètres de la boîte « *get expression* ».

L'application se présente sous la forme de Figure IV.11.

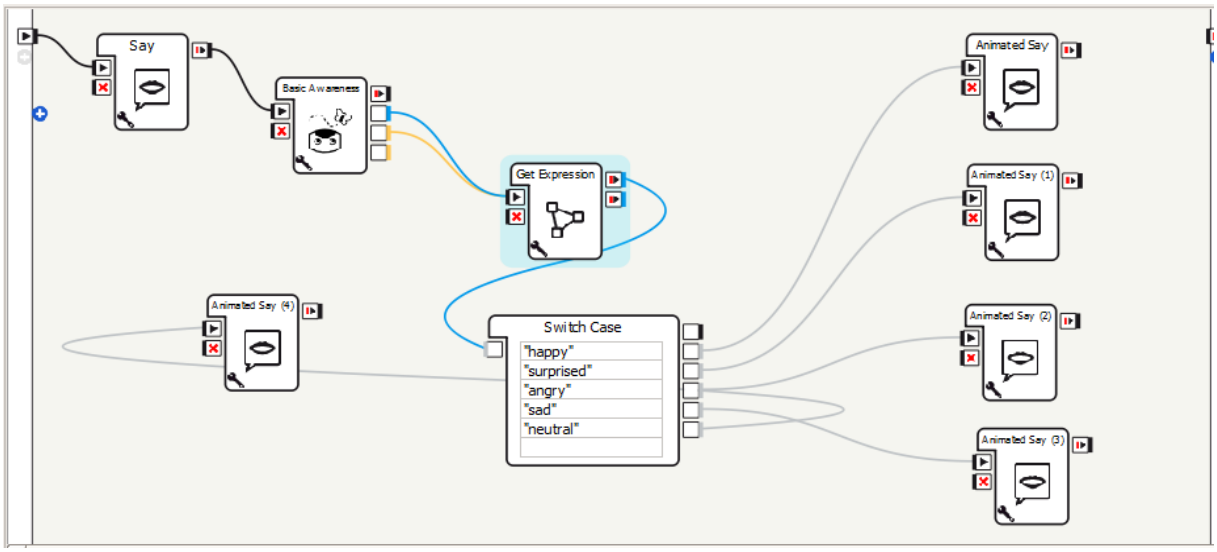


Figure IV.12. Détection des expressions faciales sous Choregraphe.

Nous avons illustré le contenu des boites « *Animated Say* » dans l'annexe A.1.

IV.3.1. Remarques

Cette application fonctionne aussi bien qu'avec une personne réelle qu'avec une photo sur papier ou numérique.

IV.4. Application 3 : Reconnaissance de couleurs

Cette application consiste à détecter un objet et à définir sa couleur. Pour se faire, on a utilisé *OpenCV* pour la conversion de l'image récupérée vers le système HSV, pour faire l'équilibrage et définir l'intervalle de chaque couleur en fonction des valeurs de la teinte (H), de la saturation (S) et de la valeur (V). On a implémenté 4 couleurs : Rouge, vert, bleu et jaune.

A chaque fois qu'un objet se présente dans le champ de vision de la caméra choisie pour le traitement, *NAO* prononcera sa couleur.

Comme on l'a vu dans le chapitre précèdent, on peut faire le test avec une webcam. Le code est donné par l'Annexe A.3.

IV.4.1. Remarques

- Le fond doit être différent de la couleur traitée.
- La luminosité ambiante influence le résultat final.
- La résolution choisie est de 320*240 pixels.
- On a fait le traitement sur 04 couleurs. Nous encourageons les prochaines générations à concevoir un programme qui sera capable de différencier plus.

IV.5. Application 4 : Tracker Detection

Cette application permet à *NAO* de reconnaître des points de repère spéciaux avec des motifs spécifiques. Ces repères sont appelés les *Naomarks*. Pour cela, on doit utiliser le module « *ALLandMarkDetection* » qui est un module de vision.

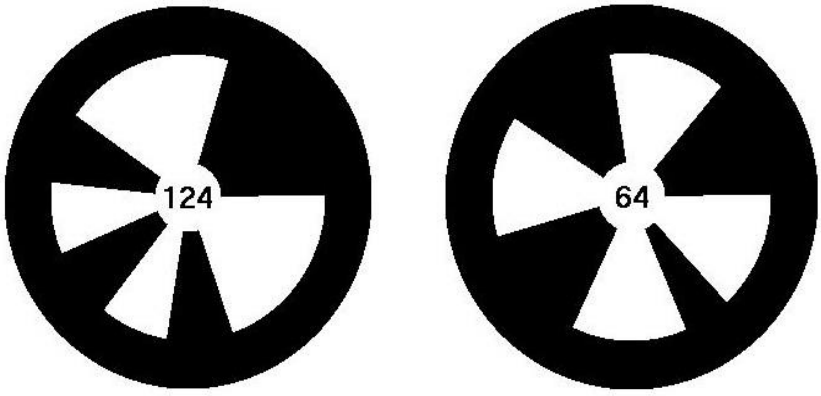


Figure IV.13. Exemples de repères spéciaux *Naomarks*.

Dans un premier temps, nous devons nous abonner à « *ALLandMarkDetection* », pour pouvoir utiliser sa fonction de détection, en précisant le nom de l'abonné, la période qui indique la fréquence à laquelle « *ALLandMarkDetection* » tente d'exécuter sa méthode, ainsi que la précision.

Le module de détection des repères stocke les résultats obtenus dans la variable « *LandmarkDetected* » se trouvant dans « *ALMemory* », qui est une mémoire centralisée utilisée pour stocker toutes les informations clés liées à la configuration matérielle de *NAO*.

Pour pouvoir utiliser les résultats de détection directement dans notre code, on vérifie régulièrement la variable « *ALMemory* » en créant simplement un proxy vers celui-ci et on récupère la valeur de la variable avec l'instruction « *getData("LandmarkDetected")* ».

On aura deux possibilités :

1. Si aucun repère n'est détecté, alors la variable retournée sera vide. Comme c'est un tableau, alors le résultat en Python est un vecteur vide : [ ].
2. Si maintenant N *Naomarks* sont détectées, alors la variable sera définie comme suit :

Var = [[TimeStampField], [Mark_info_0, Mark_info_1, . . . , Mark_info_N-1]]

Avec :

TimeStampField = [TimeStamp_seconds, Timestamp_microseconds], qui représente l'horodatage de l'image qui a été utilisée pour effectuer la détection.

Mark_info = [ShapeInfo, ExtraInfo], pour chaque marque détectée, nous avons un champ *Mark_info*.

- **ShapeInfo = [1, alpha, bêta, sizeX, sizeY, angle_de cap],** alpha et beta représentent l'emplacement du « *Naomark* » en termes d'angles de caméra, sizeX et sizeY sont la taille de la marque en angles de caméra et l'angle de cap décrit comment le *Naomark* est orienté autour de l'axe vertical par rapport à la tête du robot.
- **ExtraInfo = [MarkID]**, l'ID de la marque est le numéro inscrit sur le *Naomark* qui correspond à son motif. **[Web 12]**

Maintenant que nous avons présentés la méthode d'approche, le code en Python est illustré dans l'Annexe A.4. Voici le résultat de l'implémentation de l'application dans NAO en Figure IV.13.

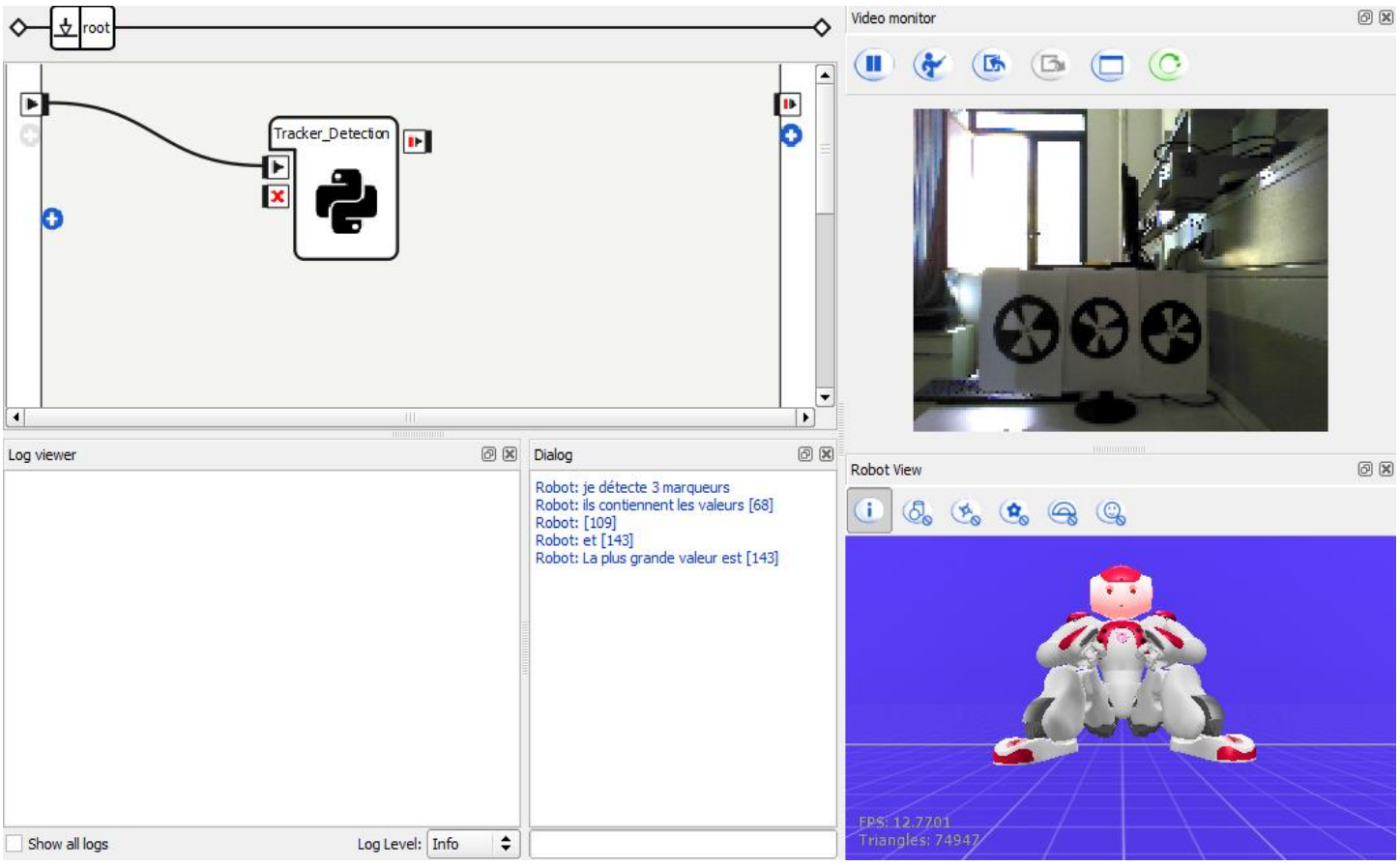


Figure IV.14. Reconnaissance des points de repères spéciaux.

IV.5.1. Remarques

- Pour que NAO puisse reconnaître les repères, il faut augmenter la résolution à 640*480 au minimum.
- Hormis le réglage de la résolution, cette application ne connaît aucuns autres inconvénients.

IV.6. Application 5 : Reconnaissance de formes

L'application 5 consiste en la détection et la reconnaissance de la forme géométrique d'un objet. Pour ce faire, l'algorithme procède en suivant trois étapes. D'abord détecter l'objet à travers sa couleurs en utilisant les ressources de la bibliothèque *Opencv* pour la conversion de l'image vers le système HSV, afin de définir l'intervalle de couleur, ensuite définir le contour de l'objet en utilisant la fonction de *Opencv* qui est « *cv2.findContours()* ».

Enfin, l'algorithme calcule, à travers la fonction « *cv2.approxPolyDP*() », le nombre de cotés et renvoie le résultat à travers la variables « *approx* ». L'implémentation s'est faite sur des formes de couleur bleue.

L'application a été implémentée sur *Choregraphe* à travers la *Python Box,* selon le code de l'Annexe A.5.

IV.6.1. Remarques

- L'utilisation de la résolution de 1280*960 est recommandé.
- La forme géométrique doit avoir un contour linéaire.
- s, en utilisant d'autres méthodes de traitement d'images plus performantes.

IV.7. Conclusion

Dans ce quatrième chapitre, dédié aux applications, nous avons pu tester les capacités de reconnaissance de *NAO*.

Le développement de l'application 1 nous a permis de tester les capacités de *NAO* à l'apprentissage et à la reconnaissance de différents objets. L'inconvénient est la latence du système à la reconnaissance de l'image préenregistrer.

Ensuite, on a testé la détection des expressions faciales du visage, avec différentes personnes. Puis, on s'est intéressé à la programmation avec le langage Python, en développant dans un premier temps, une application de détection de couleurs. Par la suite, on a testé l'application 4 permettant à NAO de reconnaitre des points de repère spéciaux en utilisant des motifs spécifiques. À la fin, on a testé l'application de détection de formes, l'inconvénient a été l'imprécision dans la détection des différentes formes, due entre autres à la variable « area » représentant le contour qui est imprécise, à la résolution qui est inversement proportionnelle au nombre d'*fps* reçu mais aussi au processeur de *NAO*, se situant dans sa tête, chauffant lors du traitement de l'application.

Notant aussi la faible autonomie de la batterie de NAO, qui nous a contraints à ne pas utiliser ses moteurs lui permettant de se mouvoir.

Conclusion générale

Ce projet de fin d'étude concerne une exploration des capacités de reconnaissances du robot humanoïde NAO, à travers le développement de diverses applications utilisant ses capacités détection visuelle.

On a d'abord commencé par aborder la robotique humanoïde en présentant les réalisations les plus avancées mondialement dans ce domaine.

Par la suite, on s'est intéressé à l'objet de notre étude en présentant la société SoftBank Robotics conceptrice de NAO, puis on a décrit l'architecture et la technologie embarqué dans celui-ci, on a aussi montré comment le mettre en marche et le connecter au réseau, et présenter le logiciel Choregraphe et fini par une description des fonctions de la boîte Python qui a été utile dans la partie applications.

L'outil de reconnaissance principal de NAO étant sa caméra, on s'est penché sur ses caractéristiques techniques, mais aussi aux espaces colorimétriques pris en charge par la caméra de l'humanoïde, on a présenté les outils et méthodes nécessaires pour une utilisation optimale de celle-ci.

On a fini par illustrer les applications implémentées sur NAO, toute traitant sur la reconnaissance visuelle du robot, programmées avec les boîtes Choregraphe et la programmation graphique ainsi que le langage de programmation Python.

Le choix porter de traiter uniquement sur la reconnaissance visuelle a été dicté par le fait que c'est une bronche importante et très intéressante en robotique, ce domaine n'est qu'à ses débuts donc présente encore un grand défi pour les ingénieurs, son avenir dans le secteur des humanoïdes pourra donner une évolution rapide de ce secteur porteur pour l'avenir.

Ce travail de fin d'étude nous a permis de consolider les connaissances acquises durant notre cursus, on a entre autre appliqué nos connaissances en programmation informatique, traitement de l'image et protocoles de communication.

Référence bibliographique

Références bibliographiques

[1] Sanchez-Riera, Jordi. "*Developing Audio-Visual capabilities of humanoid robot NAO*." Thèse de doctorat. Université de Grenoble, 2013.

[2] Thomas Moulard. "*Optimisation numérique pour la robotique et exécution de trajectoires référencées capteurs."* Thèse de doctorat. Institut National Polytechnique de Toulouse - INPT, 2012.

[3] Rita Baddoura-Gaugler. " *L'homme et le robot humanoïde : Transmission, Résistance et Subjectivation*. " Thèse de doctorat. UNIVERSITE PAUL VALERY - MONTPELLIER III, 2013

[4] Guide : Démarrer avec NAO version 2.1.4

[5] Joachim.A, Rémi.C, Guillaume.G, Chunhui.L, Wei.G, Benjamin.P, Sébastien.S, Cedric.V. : "*Guide Utilisateur du robot humanoïde NAO.* " Projet d'Ingénierie du Logiciel en Informatique. École Polytechnique de l'Université de Tours. France, 2012.

[6] Documentation du robot NAO *(Aldebaran documentation)* installé avec le logiciel Choregraphe, version : 2.1

[7] Wang, Chaoyue. "*Behaviour Design of NAO Humanoid Robot Playing Tic Tac Toe Game*", Technology and Communication, Vaasan Ammattikorkeakoulu University of Applied Sciences, Finland. 2015.

[Web 1] : http://www-igm.univ-mlv.fr/~dr/XPOSE2002/robotique/chapitres/Historique.htm, consulté le 18 avril 2020.

[Web 2] : https://www.futura-sciences.com/tech/definitions/robotique-robotique-603/, consulté le 18 avril 2020.

[Web 3] : http://robfutur.blogspot.com/2018/02/les-differents-types-de-robots.html, consulté le 18 avril 2020.

[Web 4] : https://www.hansonrobotics.com/sophia/, consulté le 19 avril 2020.

[Web 5] : https://sites.google.com/site/avesnellestechno14k/les-robots-antropormorfique, consulté le 19 avril 2020.

[Web 6] : https://intelligence-artificielle-robotique.weebly.com/les-robots-anthropomorphiques.html, consulté le 19 avril 2020.

[Web 7] : https://global.honda/innovation/robotics/ASIMO.html, consulté le 20 avril 2020.

[Web 8] : https://www.softbankrobotics.com/emea/fr/pepper, consulté le 20 avril 2020.

[Web 9] : https://www.generationrobots.com/fr/401439-robot-humano%C3%AFde-darwin-op-deluxe-edition.html, consulté le 20 avril 2020.

[Web 10] : http://tpe-robotique-exosquelette.e-monsite.com/pages/tpe/robot/robot-humanoide/fonctionnement-du-robot-humanoide.html, consulté le 20 avril 2020.

[Web 11] : http://www.isir.upmc.fr/UserFiles/File/clady_homepage/M2/UE%20Informatique%20Pour%20la%20Robotique%20-%20Intro%20%20Perception.pdf, consulté le 20 avril 2020.

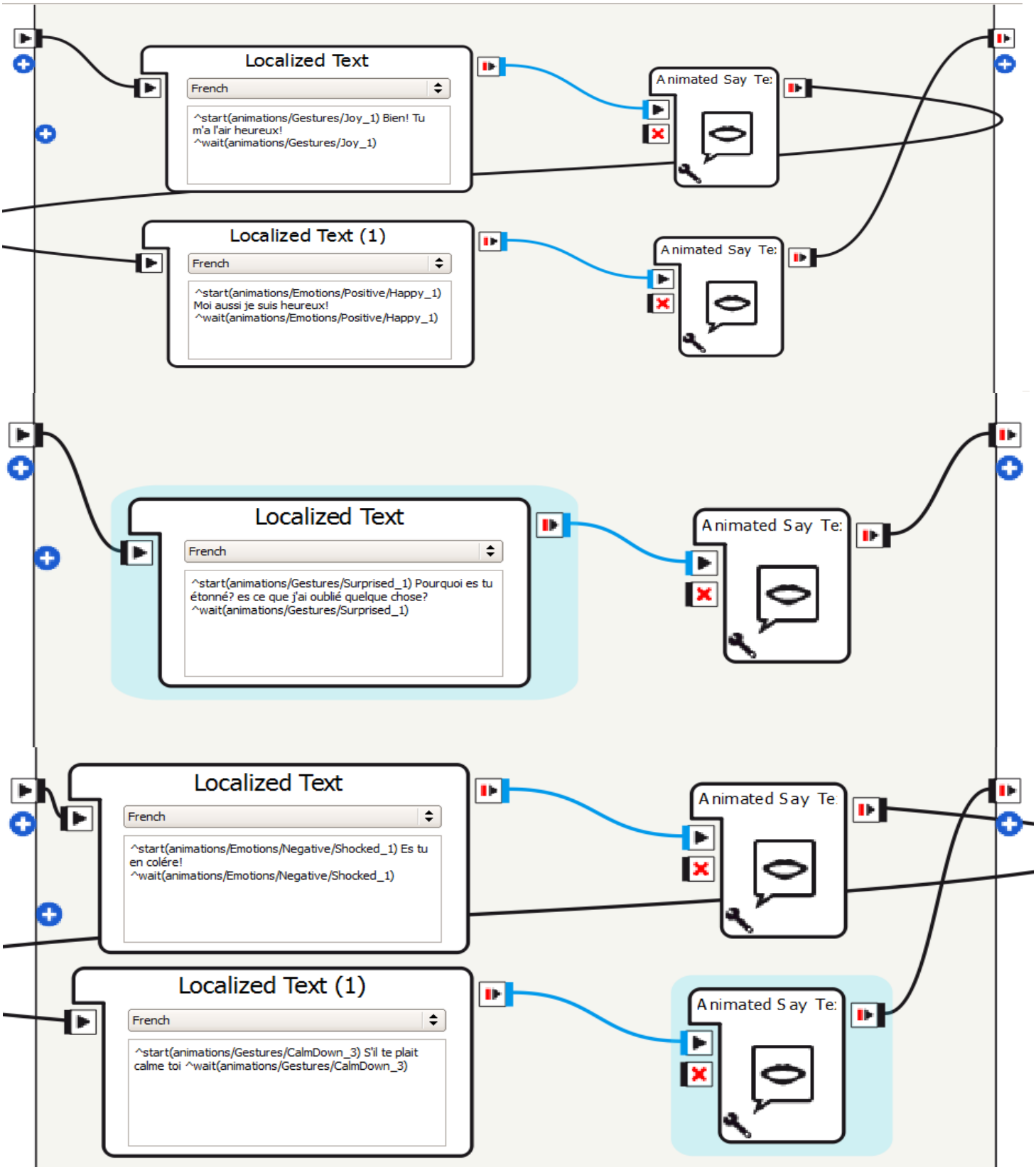
[Web 12] : http://doc.aldebaran.com/2-1/index_dev_guide.html, consulté le 05 juillet 2020.

[**Web 13**] : https://staff.fnwi.uva.nl/a.visser/research/nao/Labbook2017.html, consulté le 05 juillet 2020.

[**Web 14**] : https://www.softbankrobotics.com/emea/fr/nao, consulté le 05 juillet 2020.

Annexes

Annexe A.1. Détection des expressions du visage



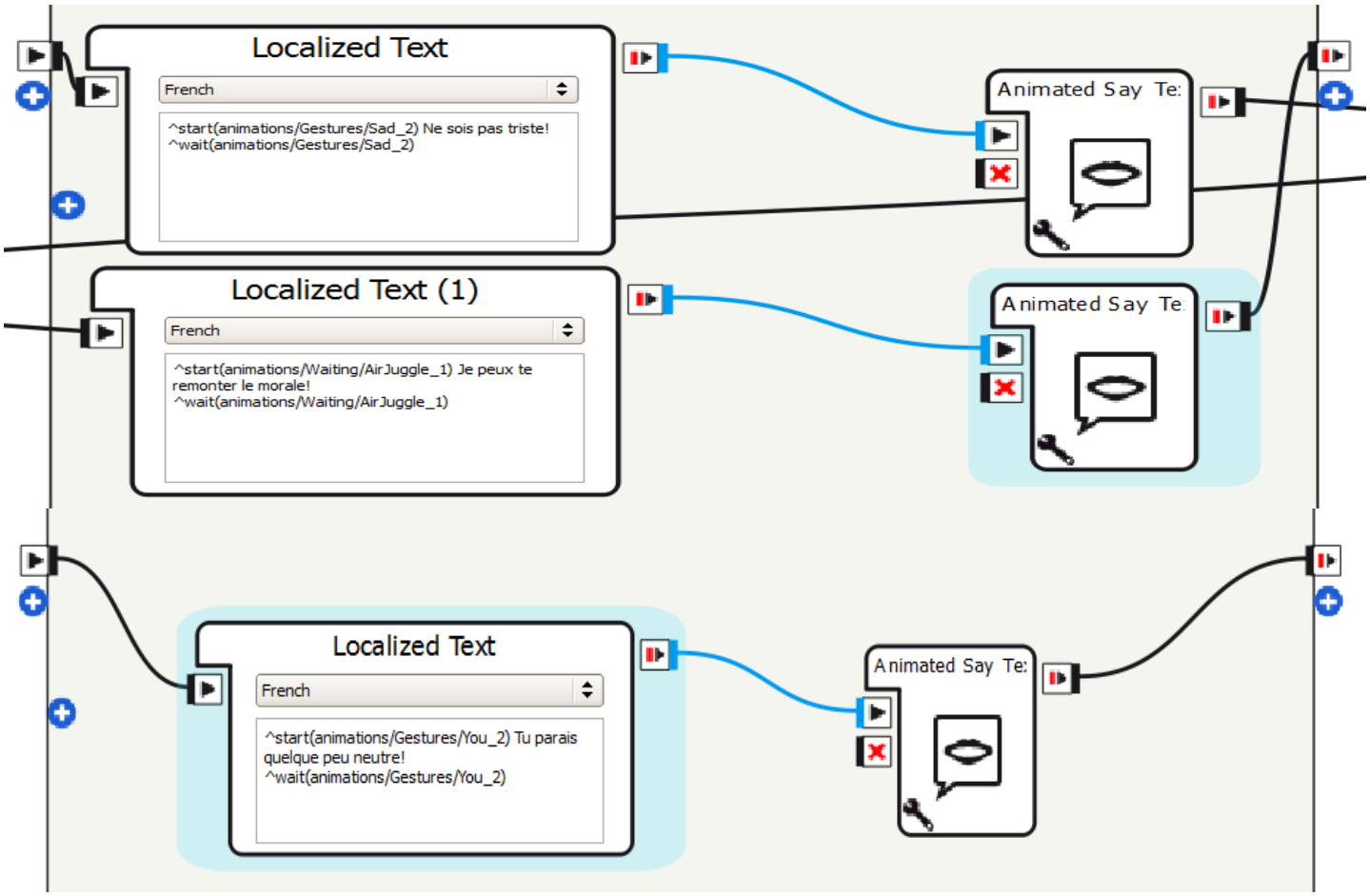


Figure A.1. Illustration du contenu des boites « *Animated Say* ».

Annexe A.2. Script Python de la boite « *Animated Say Text* »

L'Algorithme A.2. Illustre le contenu de la boite « *Animated Say Text* ».

Algorithme A.2. Sous programme de la boite « *Animated Say Text* ».

```
import time

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self, False)
        self.animSpeech = ALProxy('ALAnimatedSpeech')

    def onLoad(self):
        self.bIsRunning = False
        self.ids = [ ]

    def onUnload(self):
        for id in self.ids:
            try:
                self.animSpeech.stop(id)
            except:
                pass

    def onInput_onStart(self, p):
        self.bIsRunning = True
        try:
            sentence = "\RSPD="+ str( self.getParameter("Speed (%)") ) + "\ "
            sentence += "\VCT="+ str( self.getParameter("Voice shaping (%)") ) + "\ "
            sentence += str(p)
            sentence += "\RST\ "
            configuration =\
            {"bodyLanguageMode":self.getParameter("Body language mode")}
             id = self.animSpeech.post.say(str(sentence), configuration)
             self.ids.append(id)
             self.animSpeech.wait(id, 0)
        finally:
            try:
                self.ids.remove(id)
            except:
                pass
        if ( self.ids == [] ) :
            self.onStopped() # activate output of the box
            self.bIsRunning = False

    def onInput_onStop(self):
        self.onUnload ()
```

Annexe A.3. Reconnaissance de couleur

```python
# -*- encoding: UTF-8 -*-
import cv2 as cv
import numpy as np

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def connectToCamera(self):
        """
        Se connecter a la camera de NAO
        """
        try:
            self.avd = ALProxy("ALVideoDevice")
            strMyClientName = self.getName()
            nCamera = 0  # "0" pour la caméra en haut et "1" for la caméra en bas
            nResolution = 2  # "2": Resolution KVGA 640*480
            nColorSpace = 13  # "10" : YUV, "11" : BGR, "13" : RGB
            nFPS = 30  # De 1 à 30 ips

            self.strMyClientName = self.avd.subscribeCamera(strMyClientName, nCamera, nResolution, nColorSpace, nFPS)
        except BaseException, err:
            self.log("Err: connectToCamera : catching error: " + str(err))# Message en cas d'erreur

    def disconnectFromCamera(self):
        """
        Se déconnecter de la camera de NAO
        """
        try:
            self.avd.unsubscribe(self.strMyClientName)
        except BaseException, err:
            self.log("Err: disconnectFromCamera : catching error: " + str(err))

    def getImageFromCamera(self):
        """
        Cette méthode retourne juste l'image fournie par la camera
        pour être utilisée dans le traitement.
        """
        try:
            dataImage = self.avd.getImageRemote(self.strMyClientName)
            if(dataImage != None):
                img = (np.reshape(np.frombuffer(dataImage[6], dtype= '%iuint8' % dataImage[2]),
                       (dataImage[1], dataImage[0], dataImage[2])))
                return img
        except BaseException, err:
            self.log("Err: getImageFromCamera : catching error: " + str(err))
            return None
```

```
    def onInput_onStart(self):

        tts = ALProxy("ALTextToSpeech")
        # Connection à la camera:
        self.connectToCamera()

        r, g, b, y = 0, 0, 0, 0
        while True:
            frame = self.getImageFromCamera()
            if(frame == None):
                # Message d'erreur en cas de non réception de l'image
                self.log("Err: error while getting image from camera : frame is None")
                break
            else:
#             hsv_frame = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
#             rgb_frame = cv.cvtColor(frame, cv.COLOR_YUV2RGB)
            hsv_frame = cv.cvtColor(rgb_frame, cv.COLOR_RGB2HSV)#Conversion de l'image vers le système HSV

            """Red color :"""
            l_r = [160, 34, 0]# les plus petites valeurs de [H_rouge, S_rouge, V_rouge]
            u_r = [255, 255, 255]# les plus grandes valeur de [H_rouge, S_rouge, V_rouge]
            lower_HSV_r = np.array(l_r)
            upper_HSV_r = np.array(u_r)
            mask_r = cv.inRange(hsv_frame, lower_HSV_r, upper_HSV_r)
            result_r = cv.bitwise_and(frame, frame, mask=mask_r)
            if np.average(result_r) > 15 and r == 0:
                tts.say("objet rouge !")
                r = 1
            elif np.average(result_r) < 15 and r == 1:
                r = 0

            """Green color :"""
            l_g = [62, 99, 0]
            u_g = [79, 255, 255]
            lower_HSV_g = np.array(l_g)
            upper_HSV_g = np.array(u_g)
            mask_g = cv.inRange(hsv_frame, lower_HSV_g, upper_HSV_g)
            result_g = cv.bitwise_and(frame, frame, mask=mask_g)
            if np.average(result_g) > 15 and g == 0:
                tts.say("objet vert !")
                g = 1
            elif np.average(result_g) < 15 and g == 1:
                g = 0

            """Blue color :"""
            l_b = [99, 106, 119]
            u_b = [164, 255, 255]
            lower_HSV_b = np.array(l_b)
            upper_HSV_b = np.array(u_b)
            mask_b = cv.inRange(hsv_frame, lower_HSV_b, upper_HSV_b)
            result_b = cv.bitwise_and(frame, frame, mask=mask_b)
            if np.average(result_b) > 15 and b == 0:
                tts.say("objet bleu !")
                b = 1
            elif np.average(result_b) < 15 and b == 1:
                b = 0
```

```
113
114             """Yellow color :"""
115             l_y = [20, 113, 131]
116             u_y = [44, 255, 255]
117             lower_HSV_y = np.array(l_y)
118             upper_HSV_y = np.array(u_y)
119             mask_y = cv.inRange(hsv_frame, lower_HSV_y, upper_HSV_y)
120             result_y = cv.bitwise_and(frame, frame, mask=mask_y)
121             if np.average(result_y) > 15 and y == 0:
122                 tts.say("objet jaune !")
123                 y = 1
124             elif np.average(result_y) < 15 and y == 1:
125                 y = 0
126         # Appel de la fonction pour se deconnecter de la camera:
127         self.disconnectFromCamera()
128
129
130     def onInput_onStop(self):
131         self.onUnload()
132         self.onStopped()
133
```

Figure A.3. Code Python de la détection d'objet et la reconnaissance de sa couleur.

Annexe A.4. Reconnaissance des points de repère « Naomarks »

```python
# -*- encoding: UTF-8 -*-
import time
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        # Créer un proxy à ALTextToSpeech
        self.tts = ALProxy("ALTextToSpeech")
        # Créer un proxy à ALMotion
        self.motion = ALProxy("ALMotion")
#         # Créer un proxy ALMemory
        self.memProxy = ALProxy("ALMemory")
        # Créer un proxy à ALLandMarkDetection
        self.markProxy = ALProxy("ALLandMarkDetection")
        self.period = 500 # period = 0.5 sec
        self.markProxy.subscribe("Test_Landmark", self.period, 0.2 )

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        val = []
        max, j = 0, 0
        # Récupère les données de la marque
        data = self.memProxy.getData("LandmarkDetected", 0)# [[TimeStampField], [markInfo]]
        markInfo = data[1]  # on a data[1] = [mark_info_0, mark_info_1,..., mark_info_N-1]]
                            # avec mark_info_0 = [shapeInfo_0, extraInfo_0]
        for i in markInfo:
            val.append(i[1])# i[1] = markID qui est la valeur de la marque avec ExtraInfo = MarkID
            #Obtenir la plus grand valeur:
            if max < val[j]:
                max = val[j]
                k = val.index(val[j])
            j+=1
        self.tts.say("je détecte " + str(len(val)) + " marqueurs")
        time.sleep(1)
        for i in val:
            if i == val[0]:
                self.tts.say("ils contiennent les valeurs " + str(i))
                time.sleep(1)
            elif i == val[-1]:
                self.tts.say("et "+str(i))
                time.sleep(1)
            else:
                self.tts.say(str(i))
                time.sleep(1)
        alpha = markInfo[k][0][1]#L'angle Yaw de la plus grande valeur
        beta = markInfo[k][0][2]#L'angle Pitch de la plus grande valeur
        self.motion.post.setAngles(["HeadYaw", "HeadPitch"], [alpha, beta], 0.05)
        self.tts.say("La plus grande valeur est "+str(max))

    def onInput_onStop(self):
        self.onUnload()
        self.onStopped()
```

Figure A.4. Code Python de la reconnaissance des points de repères spéciaux.

Annexe A.5. Reconnaissance des formes géométriques

```python
# -*- encoding: UTF-8 -*-
import cv2 as cv
import numpy as np

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def connectToCamera(self):
        """
        Se connecter a la camera de NAO
        """
        try:
            self.avd = ALProxy("ALVideoDevice")
            strMyClientName = self.getName()
            nCamera = 0  # "0" Pour la caméra en haut et "1" pour la caméra en bas
            nResolution = 2  # "2": Resolution KVGA 640*480
            nColorSpace = 10  # "10" : YUV, "11" : BGR, "13" : RGB
            nFPS = 30  # De 1 à 30 ips

            self.strMyClientName = self.avd.subscribeCamera(strMyClientName, nCamera, nResolution, nColorSpace, nFPS)
        except BaseException, err:
            self.log("Err: connectToCamera : catching error: " + str(err))# Message en cas d'erreur

    def disconnectFromCamera(self):
        """
        Se déconnecter de la camera de NAO
        """
        try:
            self.avd.unsubscribe(self.strMyClientName)
        except BaseException, err:
            self.log("Err: disconnectFromCamera : catching error: " + str(err))

    def getImageFromCamera(self):
        """
        Cette méthode retourne juste l'image fournie par la camera
        pour être utilisée dans le traitement.
        """
        try:
            dataImage = self.avd.getImageRemote(self.strMyClientName)
            if(dataImage != None):
                img = (np.reshape(np.frombuffer(dataImage[6], dtype= '%iuint8' % dataImage[2]), (dataImage[1], dataImage[0], dataImage[2])))
                return img
        except BaseException, err:
            self.log("Err: getImageFromCamera : catching error: " + str(err))
            return None

    def onInput_onStart(self):

        tts = ALProxy("ALTextToSpeech") # Module ALProxy pour la parole
        # Connection à la camera:
        self.connectToCamera()

        tri, car, rec, cir= 0, 0, 0, 0
```

```
        while True:
            frame = self.getImageFromCamera() # Fonction permettant de recevoir l'image video sur le Video Monitor de Choregraphe
            if(frame == None):
                self.log("Err: error while getting image from camera : frame is None")# Message d'erreur en cas de non réception de l'image
                break
            else:
               #hsv_frame = cv.cvtColor(frame, cv.COLOR_BGR2HSV) #Conversion de l'image vers le système HSV sous la formecv.cvtColor(image_d'entrée,flag)
                                                                 # ou le flag determine le type de la conversion pour ce cas BGR vers HSV
                rgb_frame = cv.cvtColor(frame, cv.COLOR_YUV2RGB)
                hsv_frame = cv.cvtColor(frame, cv.COLOR_RGB2HSV)
            """Objet de couleur bleu"""
            l_b = [45, 81, 101] # les plus petites valeurs de [H_bleue, S_bleue, V_bleue]
            u_b = [164, 255, 255]# les plus grandes valeur de [H_bleue, S_bleue, V_bleue]
            lower_HSV_b = np.array(l_b)
            upper_HSV_b = np.array(u_b)
            mask = cv.inRange(hsv_frame,lower_HSV_b, upper_HSV_b)# seuiller l'image en HSV pour n'obtenir que des couleurs bleue
            kernel = np.ones((5, 5), np.uint8)            # utilisation d'une matrice unitaire de taille 5 x 5
            mask = cv.erode(mask, kernel)                 # Elimine grace a la matrice Kernel tout les point dont le noyeau est noir

            # Detection de contours
            contours, hierarchy = cv.findContours(mask, cv.RETR_LIST, cv.CHAIN_APPROX_NONE) # On manipule le mask, avec la fonction RETR_LIST
                                                                                            # elle va calculer la hierarchie entre les contours
                                                                                            # La methode cv.CHAIN_APPROX_NONE va se charger
                                                                                            # d'économiser de la mémoire
            for cnt in contours:
                area = cv.contourArea(cnt)  # Calcul une zone de contour, la zone retournée et le nombre de pixels non nuls
                approx = cv.approxPolyDP(cnt, 0.01 * cv.arcLength(cnt, True), True) # La fonction permet l'approximation de l'objet de forme quelconque
                                                                                    # avec une courbe ayant moins de sommets utilisant l'algorithme de
                                                                                    # Douglas-Peucker
                x , y , w , h = cv.boundingRect ( approx )  # Calcul un rectangle des délimitations d'un ensemble de points

                if area > 400:
                    """Forme Triangulaire"""
                    if len (approx) == 3 and tri == 0:
                        tts.say ("Je vois un Triangle !")
                        tri = 1
                    if len(approx) != 3 and tri == 1:
                        tri = 0
                    if len(approx) == 4:
                        arRatio = w / float ( h )
                        """Forme Carré"""
                        if 0.95 < arRatio < 1.05 and car == 0:
                            tts.say ( "Je vois un Carré !" )
                            car = 1
                        if len(approx) != 4 and car == 1:
                            car = 0
                        """Forme Rectangulaire"""
                        if arRatio >= 1.05 and rec == 0:
                            tts.say ( "Je vois un Rectangle !" )
                            rec = 1
                        if len(approx) != 4 and rec == 1:
                            rec = 0
                    """Forme Circulaire"""
                    if len(approx) > 10 and len(approx) < 20 and cir == 0:
                        tts.say ( "Je vois un Cercle !" )
                        cir = 1
                    if len(approx) >= 20 and cir == 1:
                        cir = 0

        # Appel de la fonction pour se deconnecter de la camera:
            self.disconnectFromCamera()
            self.onUnload()
            self.onStopped()

        def onInput_onStop(self):
            self.onUnload()
            self.onStopped()
```

Figure A.5. Code Python de la reconnaissance des formes géométriques.

Annexe A.6. Articulation de la tête

La tête du robot *NAO* possède 2 degrées de libérté. Un rotation autoure de l'axe y qui est le mouvement *Pitch*, et celui autour de l'axe z qui représente quant à lui le mouvement *Yaw*. La figure A.2 illustre la limitation des ses mouvements.

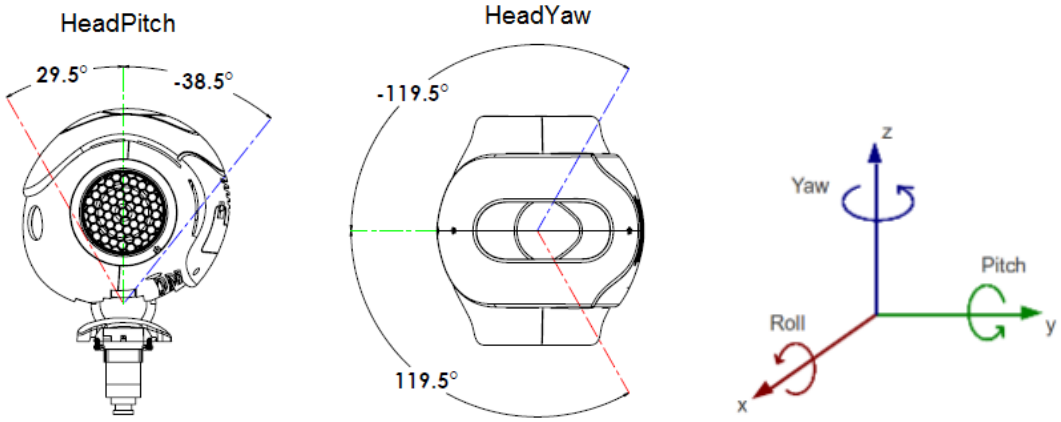


Figure A.6. Mouvement de la tête de *NAO*.

Résumé :

L'objectif de ce travail de Master concerne l'exploration des facultés de reconnaissances du robot humanoïde NAO et le développement d'applications permettant de concrétiser l'étude faite. Notre approche a été de faire une introduction sur la robotique et les humanoïdes en général. Puis, on a présenté l'architecture de NAO et son logiciel Chorgraphe ainsi que les différentes caractéristiques de sa caméra 2D. Enfin, on a conclu par l'élaboration de diverses applications traitant de la reconnaissance avec le Choregraphe et avec le langage Python, on cite : Apprendre à NAO à reconnaitre des objets, Détection des expressions faciales, Reconnaissance de couleurs, Tracker detection et Reconnaissance de formes géométriques.

Mots-clés : NAO, Robotique, Reconnaissance, Robot Humanoïde.

Abstract :

The purpose of this Master's work is to explore the NAO humanoid robot's knowledge faculties and develop applications to make the study a reality. Our approach was to make an introduction to robotics and humanoids in general. The architecture of NAO and its Chorographe software were then presented, as well as the different characteristics of its 2D camera. Finally, the development of various applications dealing with recognition with the Choreographe software and the Python language was concluded, we quote: Teaching NAO to recognize objects, Facial expression detection, Color recognition, Tracker detection and Geometric shape recognition.

Keywords: NAO, Robotics, Recognition, Humanoid Robot.

ملخص:

الهدف من عمل ماستر هذا هو أستكشاف كليات الاستطلاع للروبوت البشري NAO وتطوير التطبيقات اللازمة لإجراء الدراسة. كان النهج الذي اتبعناه هو الأخذ بعين الاعتبار الروبوتات البشرية بشكل عام. ثم قدمت معمارية NAO وبرامجه «Choregraphe» وخصائص كاميراته الثنائية الأبعاد. واختتم حديثه قائلا إنه تم إعداد تطبيقات مختلفة تتعلق بالاعتراف بلغة «Choregraphe» وبلغة «Python»، نقتبس: تعليم NAO للتعرف على الأشياء، واكتشاف تعابير الوجه، والتعرف على اللون، واكتشاف المقتفي، والتعرف على الأشكال الهندسية.

الكلمات الأساسية: NAO، روبوتات، تعرف، مخلوق آلي ذو بنية بشرية .

Agzul:

Iswi n umahil-a n Master yerza asekfel n tzemmar d usteɛref n «Robot humanoïde NAO» d usnerni n usnas ara yeğğen asilaw n tezrawt yemmugen. Axeddim-nneɣ d tazwert ɣef lrubutik akked «les humanoïdes» s umata. Syin akkin, nessenked-d aɣawas n «NAO» d useɣzen-is «Choregraphe» akked tulmisin yemgaraden n lkamira-ines «2D» daɣen. Ɣer taggara, nessegra-d s uxdam n yisnasen yemgaraden yerzan asteɛref s «Choregraphe» d umeslay «python».

Awalen tisura: NAO, Rubutik, asteɛref, humanoïde.