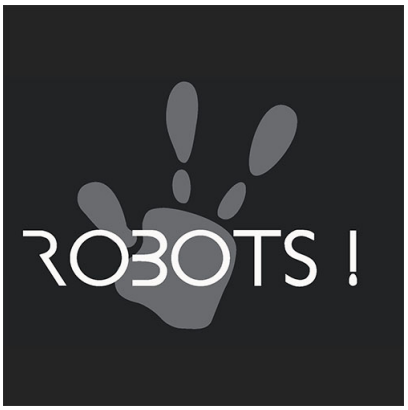
Développement de la téléopération et le contrôle de robots à travers la technique du « Danseur d'Ombres »

Rapport du travail de fin d'études.

Auteur: Víctor RODRÍGUEZ MARTÍ

Maître de stage: Sophie SAKKA

Responsable de l'option: Abdelhamid CHRIETTE



Remerciements

Avant de commencer, je souhaiterais remercier plusieurs personnes qui ont fait possible cet incroyable séjour.

Premièrement, à l'*Association Robots!* pour m'avoir accueilli et posé leur confiance en moi. En particulier à *Sophie SAKKA*, enseignante chercheur à l'École Centrale Nantes et présidente et fondatrice de *Robots!*, pour m'avoir accordé la possibilité de travailler pour l'innovation dans un secteur pointe comme la robotique humanoïde et me transmettre des principes si importants dont: l'anticipation, le travail associatif ou le travail social.

Je remercie aussi *Noémie SPIESSERT* et *Rénald GABORIAU* de m'avoir accompagné pendant les six mois.

Ainsi qu'à *Sylvain BAUD* et *Virginie ROUPENEL* avec qui j'ai eu l'opportunité de travailler pendant la grand partie du stage et ça a été avec grand plaisir.

Résumé

Le projet Danseur d'Ombres consiste à faire intervenir sur le monde un robot humanoïde commandé par téléopération. Les mouvements et la voix de l'opérateur sont envoyés au robot, la vision et le son captés par le robot sont renvoyés vers l'opérateur: il est en **immersion**. Cette technique a été développé sur le robot Pepper[1] afin d'étudier les relations affectives qui peuvent s'établir entre l'être humain et la machine, et définir les règles éthiques d'utilisation de ce type de médiation. Les applications du Danseur d'Ombres sont nombreuses : pédagogie, conférences, accueil, spectacles chorégraphiques, etc...

Ce stage de fin d'études a servi pour étendre la méthode du *Danseur d'Ombres* à d'autres types de robots, humanoïdes ou non, et de systématiser (simplifier) la connexion avec un nouveau type de robot (relation mouvements de l'opérateur et commandes robotiques) par la création d'une interface utilisateur. Pour tester la possibilité (ou non) d'étendre la technique du *Danseur d'Ombres* à d'autres types de robots, deux robots totalement différents intégreront cette technique: le robot humanoïde Nao[1] et un quadricoptère. Dans les deux cas, l'immersion dans le robot est nécessaire.

1. *Robot développé par Softbank Robotics*

Sommaire

Table des figures

Liste des tableaux

Nomenclature

CoM Center of Mass

CoS Center of Support

DGM Direct Geometric Model

DKM Direct Kinematic Model

DoF Degrees of Freedom

DS Double Support

$e.g.$ *exempli gratia*: par exemple

$i.e.$ *id est*: c'est-à-dire

IKM Inverse Kinematic Model

LS Left Support

$ref.$ référence

RS Right Support

$s.t.$ *subject to*

SS Single Support

ZMP Zero Momentum Point

1

Introduction

1.1 Contexte

A l'issue de ma dernière année d'études à l'École Centrale Nantes et ayant choisi l'option disciplinaire **robotique**, j'ai réalisé mon stage de fin d'études (TFE) au sein de l'*Association Robots!*.

1.1.1 Association Robots!

L'*Association Robots!* est basée sur Nantes et est une association 1901 d'intérêt général fondée en 2014 par *Sophie SAKKA*. Son objet est la diffusion de compétences et de connaissances sur les robots et sur leur utilisation. Parmi ses missions, l'association mène des projets de recherche sociétale sur les usages de la robotique pour répondre à une question d'apparence simple : *comment, dès aujourd'hui, les robots peuvent améliorer notre quotidien ?*

Dans ce cadre, *Association Robots!* mène plusieurs types de projets de recherche différents: Rob'Educ, Rob'Zheimer et Rob'Autisme.

Rob'Autisme

Rob'Autisme propose un accompagnement utilisant des robots Nao et basé sur la médiation culturelle et artistique. Les ateliers se déroulent sur 20 séances qui alternent 10 séances de préparation et 10 séances de programmation robotique. Un spectacle est préparé et présenté à un public plus large lors d'une 21ème séance. C'est un projet en collaboration entre l'*Association Robots!* et le CHU de Nantes (et plus précisément son hôpital de

jour, le Centre psychothérapique pour grands enfants et adolescents) dont l'orthophoniste Rénald GABORIAU, qui encadre les enfants et le projet et avec qui j'ai pu avoir des échanges très enrichissants, et trois étudiants orthophonistes.

Le but de ce programme est d'améliorer la vie d'enfants et adolescents présentant des troubles du spectre autistique par le biais du robot Nao. Celui-ci n'est qu'un outil thérapeutique qui ne serve à rien sans l'accompagnement professionnel qui est derrière. En fait, le robot Nao, et en général un robot humanoïde, peut servir à ces enfants comme intermédiaire entre eux et le monde (reste des personnes) et comme accélérateur de leur amélioration de vie.

Les résultats sont impressionnants en court et moyen terme puisque les enfants se retrouvent dans la société où ils habitent (ils arrivent à se projeter dans un futur proche) et réussissent à se communiquer avec l'extérieur sans l'angoisse détectée auparavant. Le travail d'accompagnement thérapeutique a donc pour effet la reconstruction identitaire des participants.

Personnellement, j'ai vécu la restitution de **Rob'Autisme 2019** et pu échanger avec les familles, les enfants et les orthophonistes sur les résultats directs et impressionnants du programme et sur les perspectives futures.

Rob'Zheimer

Rob'Zheimer propose un accompagnement utilisant des robots Nao et basé sur la même logique que **Rob'Autisme**.

Le programme Rob'Zheimer a été initié à Nantes en 2017, collaboration entre l'association Robots !, la maison de retraite « Les Eglantines » et l'école d'ingénieurs Centrale Nantes. Dans cette collaboration, les ateliers robotiques se déroulent à la maison de retraite : l'association Robots ! met en place le protocole expérimental et le réalise, et la recherche est réalisée par Centrale Nantes, elle s'appuie sur les résultats des ateliers.

Le projet Rob'Zheimer a pour objectif l'amélioration des capacités de communication et le développement de la qualité des interactions sociales individuelles et collectives de personnes souffrant de la maladie d'Alzheimer et maladies apparentées (MAMA). Le travail d'accompagnement thérapeutique effectué pendant les séances est donc le même que

Rob'Autisme qui a pour effet la reconstruction identitaire des participants.

Rob'Educ

Une des questions qui se pose l'*Association Robots!* est: comment les robots vont influencer nos métiers? Et c'est dans la recherche d'une réponse qui naisse **Rob'Educ** avec l'expérimentation de robotiser un métier si important comme l'éducation.

Rob'Educ a pour objectif l'analyse de la contribution robotique dans le monde de l'éducation, particulièrement de la pédagogie. Il s'agit ici de mettre en place une expérience d'enseignement réalisée par un robot. Cette expérience n'a pas vocation à remplacer l'enseignant humain, mais à comprendre précisément ce qu'apporte spécifiquement un enseignant humain, ce qu'apporte spécifiquement un enseignant robotique, et comment on peut concilier les deux pour améliorer le message pédagogique.

Personnellement j'ai pu expérimenter *Rob'Educ* pendant un cours à École Centrale Nantes (voir Figure 1.1) dans le cadre de la matière *Robots humanoïdes* où le robot Pepper à réalisé deux séances de cours magistraux dont j'ai participé et j'ai discuté sur la robotisation des métiers avec mes collègues.

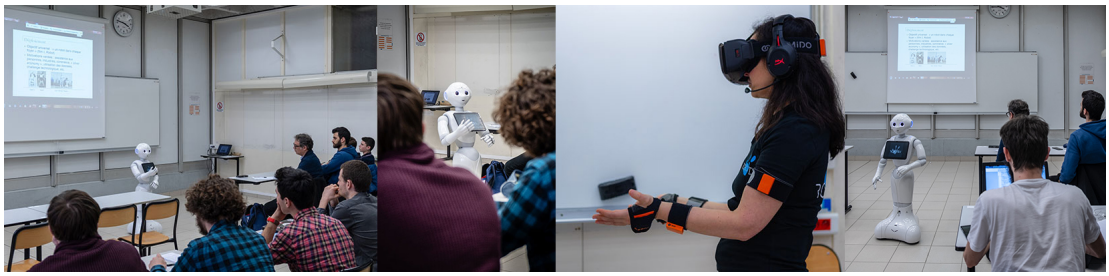


FIG. 1.1: Expérimentation de Rob'Educ à l'École Centrale Nantes.

1.1.2 Le Danseur d'Ombres

Le *Danseur d'Ombres* est une technique qui consiste à faire intervenir sur le monde un robot humanoïde commandé par un humain par téléopération. Les mouvements et la voix de l'opérateur sont envoyés au robot, la vision et le son captés par le robot sont renvoyés vers l'opérateur: c'est **une immersion totale**.

FIG. 1.2: Téléopération de Pepper par la technique du *Danseur d'Ombres*

Cette technique est existante à l'*Association Robots!* grâce à la technologie de *motion capture XSENS* et le robot humanoïde Pepper[1]. Ceci a été développé en 2018 par *Erick Ah-Mouck*, étudiant de l'École Centrale Nantes dans l'option **robotique**. Comment il a été mentionné dans le paragraphe antérieur, un humain (téléopérateur) commande le robot avec son corps, lui envoi sa voix et reçoit la vision et le son détectés par Pepper. La Figure 1.2 nous montre une téléopération de Pepper avec la technique du *Danseur d'Ombres* où le téléopérateur se situe dans le second plan puis Pepper dans le premier plan et on peut remarquer plusieurs aspects:

- **La tenue vestimentaire**: le téléopérateur est équipé avec 17 capteurs inertiels *XSENS* distribués partout dans le corps pour détecter les mouvement et les traiter grâce à leur logiciel *MVN Analyse*.
- **Les périphériques**: le téléopérateur porte des lunettes et des casques qui permettent une immersion totale lors de la téléopération. L'opérateur voit à travers

1. *SoftBank Robotics*

les deux caméras de Pepper, écoute le son détecté par les micros du robot et a un micro avec lequel peut transmettre sa voix filtrée au robot pour la reproduire. Aussi remarquer que le téléopérateur ne voit pas son entourage et donc il doit être situé dans une zone sans obstacles.

Lors de mon arrivée à l'*Association Robots!*, la technique du *Danseur d'Ombres*, étant déjà développée, offrait les possibilités suivantes:

- **Téléopérer** le robot Pepper en temps réel. Ceci implique: imiter le téléopérateur, avancer vers l'avant et tourner.
- **Communication robot-téléopérateur en quasi-temps réel** (une seconde de décalage dans le retour du son).

Étant ces points ci-dessus l'état d'art de la technique du *Danseur d'Ombres*, la mission de mon stage de fin d'études est:

- **Étendre cette technique à d'autres robots (humanoïdes ou pas)**. Pour cela, la technologie du *Danseur d'Ombres* va être implémenté dans deux autres robots: Nao et un quadricopter.
- **Optimiser la technologie suivant une seule logique**. Le *Danseur d'Ombres* étant seulement applicable à Pepper, si on veut l'utiliser sur d'autres robots, il ne doit pas être spécifique à un seul robot mais prendre une seule logique et l'implémenter avec les spécificités de chaque robot. Ceci peut s'expliquer comme une abstraction de la technologie dans un domaine au dessus de la robotique humanoïde (celui de la robotique en générale).

Ainsi qu'une optimisation du l'application déjà existante du *Danseur d'Ombres* en ajoutant:

- Une **interface utilisateur**[2]: facilite le démarrage du *Danseur d'Ombres* et les différentes application issue de cette technique.
- Translation au **système d'exploitation Windows**: l'état de l'art de la technique partageait différentes fonctionnalités entre Linux et Windows ce qui rendait sa logique et sa transportabilité très compliquée.
- Ajouter la possibilité de la **marche arrière** pendant la téléopération.
- Rendre le *Danseur d'Ombres* **une application transportable** à d'autres ordinateur pour que son installation soit possible dans n'importe quel dispositif.

2. GUI: Graphical User Interface

1.2 Positionnement

Mon positionnement par rapport aux missions décrites ci-dessus se résume en trois problématiques:

- Appliquer la technologie à n'importe quel type de robot. **Une abstraction** (ou relativisation) **de la technique** du *Danseur d'Ombres*. Actuellement étant développé que pour Pepper, elle doit suivre une logique qui soie applicable a tout type de robot.
- Tester ceci avec **le robot Nao**.
- Tester ceci avec **un quadricopter**.

1.2.1 Abstraction de la technique.

Pour atteindre ces objectifs, je vais présenter une analyse en détail du hardware utilisé dans le *Danseur d'Ombres*, c'est-à-dire, le système *Xsens Motion Capture* et le robot Pepper avec ces spécificités propres et ces mécanismes créés par *SoftBank Robotics*. En l'occurrence, trouver un unique logique pour mettre en relation un système d'extraction des position articulaires d'un opérateur et un robot se base sur la normalisation des deux système à un même environnement et dans la mise en commun de données qui seront identifiées comme équivalentes.

En effet, la logique suivie est celle de mettre en lien les DoF de l'opérateur avec ceux du robot et le moyen d'extraire les DoF d'un humain est par le biais de la technologie *Xsens*.

En conclusion, si on peut extraire les DoF d'un opérateur et les normaliser, on peut les associer avec une entrée de commande robotique, comme par exemple, une entrée articulaire pour un robot humanoïde ou industriel (type *Baxter*) ou une entrée de commande type *roll, pitch, yaw* ou *throttle*[3] pour un robot aérien. Ceci peut seulement se faire avec la préparation d'un environnement commun entre l'opérateur et le robot: **communication en temps réel**, **normalisation, rectification et pré-traitement des données**, **base de coordonnées relative**, etc...

L'étude et l'analyse profond du *hardware* de Xsens permettra un pré-traitement des données et de l'autre côté, l'implémentation se basera sur l'étude du software des ro-

3. Entrée qui fait varier le couple appliqué dans les quatre moteurs et qui permet de contrôler l'altitude du quadricoptère.

bots, des libraires de communication multimédia ou *streamming* ainsi que les techniques de contrôle de robot et optimisation informatique (calcul matriciel).

Pour compléter la validation de la logique présentée ci-dessus, on va la tester sur deux robots: Nao et un quadricopter.

1.2.2 Application sur le robot humanoïde NAO.

Le premier test pour valider expérimentalement l'abstraction ou relativisation de la technique du *Danseur d'Ombres* à tout type de robot va se réaliser avec le robot humanoïde Nao. Grâce à toute la logique expliquée dans le chapitre 3, la méthodologie d'implémentation d'un nouveau robot dans cette technique est:

- Comprendre la structure morphologique et électronique. (voir section 2.1.2)
- Étudier les spécificités du robot (type de robot, numéro de DoF, nom des articulations, etc...) (voir section 4.1)
- Associer les DoF humain et ceux du robot. (voir chapitre 3)
- Intégrer les librairies *GStreamer* pour l'immersion. (voir section 2.2.2)

Un autre aspect très important est l'analyse et réflexion de l'usage de la technique *Danseur d'Ombres* sur le robot. Dans ce sens, Nao a supposé un grand défi à cause de sa contrainte de l'équilibre. La plus grande spécificité du robot Nao est ses deux jambes et aussi sa plus grande problématique puisqu'elles limitent beaucoup ses mouvements. En faite, chaque déplacement ou configuration articulaire contient la contrainte de l'équilibre qui n'est pas intrinsèque dans le robot comme il l'est pour l'opérateur. Lors de l'imitation (ou la manipulation de Nao), le téléopérateur peut atteindre des posture non équilibrées pour le robot même si elles le sont pour l'humain. Ceci est un problème de sécurité est c'est une contrainte de priorité extrême.

La spécificité de l'équilibre chez le robot humanoïde Nao va supposer qu'on étudie les principes fondamentaux de l'équilibre chez les robots, son implémentation dans un système redondant et sa compatibilité avec l'imitation de l'opérateur. Dans ce sens, je vais me baser beaucoup sur le travail réalisé par Louise Poubel [17] où elle expose l'équilibre chez Nao qui imite à son tour à un opérateur par le biais d'une *Kinect*[4].

4. Microsoft Xbox Kinect

1.2.3 Application sur un drone.

L'extrapolation vers un robot non humanoïde est très important pour valider l'expérimentation. Un quadricopter est un robot aérien, avec quatre moteurs, un contrôleur de vol (CPU) et quelques capteurs pour détecter les informations dynamiques de l'environnement. Il est, en plus, assez abordable économiquement[5]. En particulier, pendant ce stage, j'ai eu le privilège de contacter *Pilgrim Technologies* pour l'achat d'un drone fait à mesure. Nos particularités étaient:

- Le vol indoor: l'expérimentation vise un vol à l'intérieur des bâtiments.
- Établir une communication PC-drone.

Le positionnement est, dans un premier temps, de contrôler les quatre DoF du drone: *roll, pitch, yaw, throttle*, avec l'aide de la technologie Xsens et les articulations humaines et non pas avec la manette de pilotage habituelle.

Avec une communication PC-drone, on a accès aux fonctionnalités Xsens, i.e., téléopération du robot avec le corps humain (communication: capteurs $\Leftrightarrow$ logiciel *MVN Analyse* situé dans le PC). En addition, le type de drone (pour vol indoor) permet un usage plus basique, sans fonctionnalités très complexes, facile à programmer (pas de GPS), moins cher et surtout, idéal pour valider expérimentalement l'abstraction ou relativisation de la technique du *Danseur d'Ombres*. C'est en conclusion, une bonne cible et la logique mène à penser que c'est fesable.

5. En comparaison à d'autres robots non humanoïdes

2

Existence et compétences prérequises

2.1 Hardware

2.1.1 Xsens Motion Capture

Le système de capture inertielle de mouvement Xsens MVN est un système facile à utiliser pour la capture de mouvement du corps humain. MVN est basé sur des capteurs inertiels en miniature, à la pointe de la technologie, et des solutions à une communication sans fil combinées à des algorithmes de fusion de capteurs avancés, utilisant des hypothèses de modèles biomécaniques; c'est-à-dire, un système qui permet à l'utilisateur de saisir la position, la rotation, la vitesse et l'accélération de chaque capteur inertiel.

Pour la correcte utilisation de cette technologie, c'est nécessaire de lire le manuel d'utilisation [30] où sont expliquées les instructions à suivre pour le positionnement des capteurs MVN AWINDA (voir Figure 2.1) dans le téléopérateur (voir Figure 2.2) et l'utilisation du logiciel *MVN Analyse* (voir Figure 2.3).

FIG. 2.1: Capteurs XSENS AWINDA. SOURCE:[30]

FIG. 2.2: Position des capteurs. SOURCE:[30]

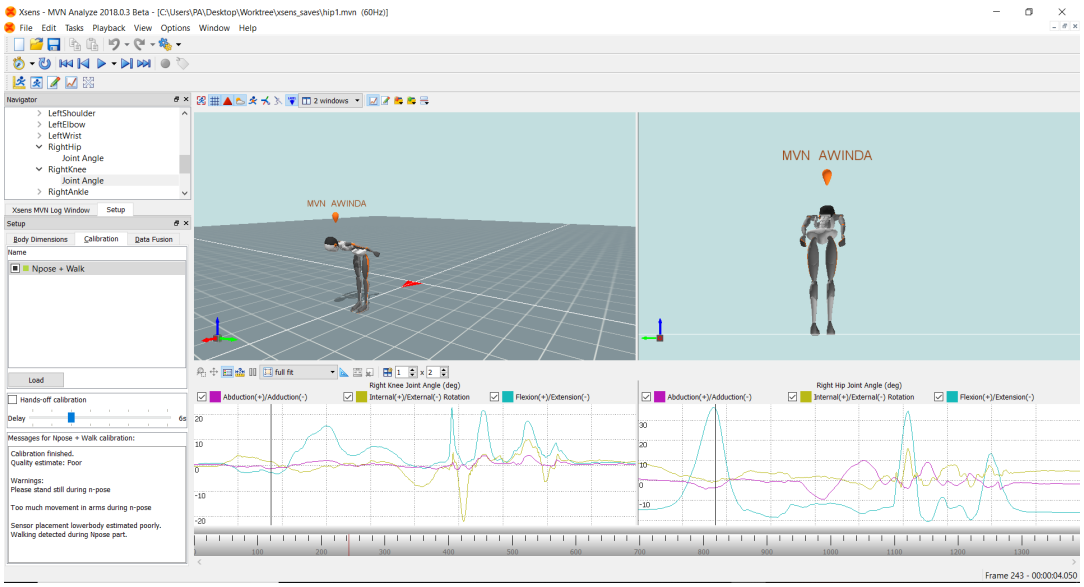


FIG. 2.3: Logiciel MVN Analyse. SOURCE:[30]

Environnement Xsens

Contrairement à l'environnement du robot, Xsens établie son propre environnement pour traiter les données des capteurs. Selon des modèles biomécaniques, Xsens reproduit un squelette humain en connectant les différents segments (par exemple: RightUpperLeg, LeftForeArm, etc...) avec les différentes articulations (par exemple: Right Wrist, Left Knee, etc...) ce qui permet à l'utilisateur d'avoir plusieurs possibilités de repères à saisir et différents paquets d'informations (dont on y reviendra après).

Plus spécifiquement, le modèle anatomique utilisé par Xsens est le suivant (Figure 2.4) : Et le repère le plus intéressant présenté par Xsens est le repère articulaire (voir Figure 2.5)

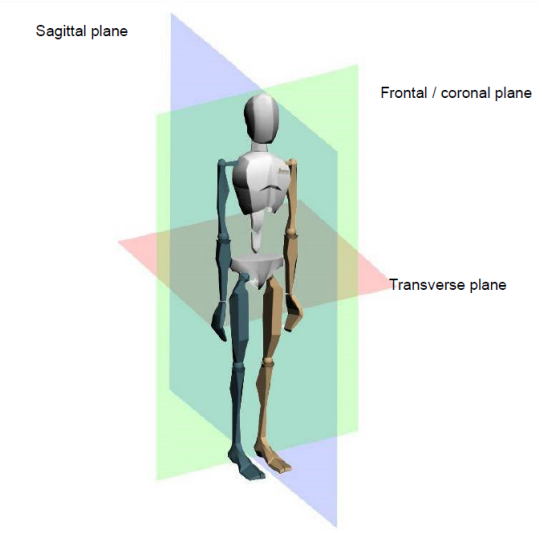


FIG. 2.4: Plans corporels. SOURCE:[30]

où chaque articulation du modèle anatomique[1] a son propre repère relative. On remarque qu'il s'agit de repères *right handed Y-up* et qu'ils sont tous orientés de la même façon. Le figure 2.4 est très important pour comprendre la complexité du système de coordonnées Xsens et la différence avec le repère XYZ traditionnel. En plus, la figure 2.5 montre le modèle anatomique : cinq chaînes et 23 segments :

- **Torso**: Pelvis, Spinal segments (L5, L3, T12), Sternum, Neck, Head.
- **Right Arm**: RShoulder, RUpperArm, RForeArm, RHand.
- **Right Leg**: RUpperLeg, RLowerLeg, RFoot, RToe
- **Left Arm**: LShoulder, LUpperArm, LForeArm, LHand.
- **Left Leg**: LUpperLeg, LLowerLeg, LFoot, LToe

Un segment est l'élément entre les articulations. Chaque segment représente un capteur et permet de construire le modèle de la figure 2.5. Néanmoins, certains segments sont calculées par interpolation, en particulier, les *Spinal segments (L5, L3, T12), Neck, RToe, LToe* sont des segments importants et nécessaires pour construire le modèle anatomique mais ils sont facilement calculées et donc nous n'avons pas besoin des capteurs.

Les articulation sont modélisées par rapport au modèle anatomique et des calculs relatifs entre deux segments liés à une même articulation. Dans ce scénario particulier, où il est important de connaître les valeurs des articulations humaines pour les envoyer au robot, nous n'allons pas travailler avec les segments décrits précédemment, mais directement

1. Modèle humain avec 22 articulation et 23 segments

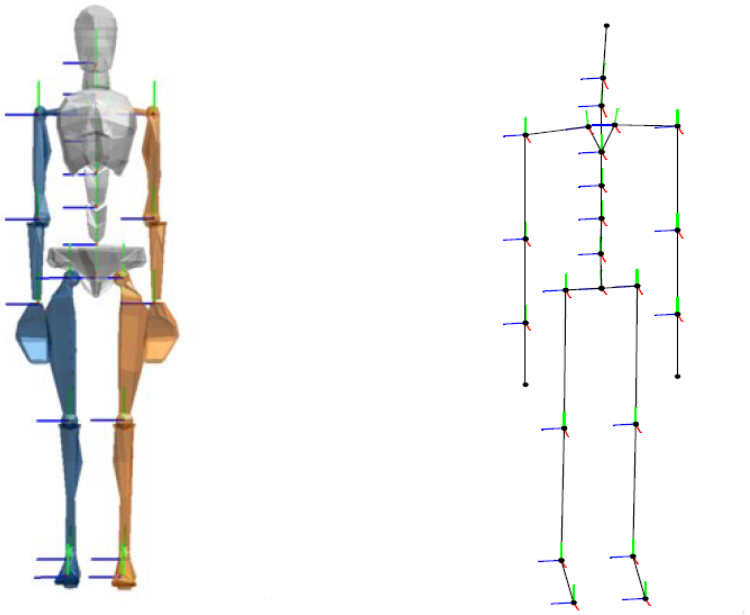


FIG. 2.5: *Gauche*: Modélisation du squelette humain. *Droite*: Repères articulaires. Rouge:X ; Vert:Y ; Bleu:Z. SOURCE:[30]

avec les articulations.

Xsens MVN nous permet donc de récupérer les information des 22 articulations présentes dans la Figure A.1 (voir Annexe A) (le pelvis ne fait pas partie des articulations) et c'est grâce à ces valeurs qu'on peut téléopérer, par exemple, Pepper. En faite, la technologie Xsens et leur logiciel nous permet d'extraire les DoF [2] d'un humain et dans ce sens, ces valeur vont être la base de la nouvelle logique de la téléopération de robots (humanoïdes ou pas) par la technique du *Danseur d'Ombres*. En particulier, les capteurs inertiels *MVN AWINDA* nous apportent, entre autre, leur position en 6D (trois positions et trois rotations). On s'intéresse qu'aux orientations des articulations (on a que des liaisons rotules même si quelques unes sont si parfaites que peuvent être à la fois des liaisons prismatiques) et donc on va extraire trois orientation de chaque articulation: le *roll, pitch et yaw* (voir Figure 2.6). **C'est d'ailleurs cette base qu'on va utiliser comme la base normalisé et tout sera exprimée selon un base du type Figure 2.6**.

En conclusion, **Xsens MVN permet d'extraire 22*3 = 66 DoF de l'opérateur**. La table 2.1 reprend tous les articulation (de la Figure A.1) et segments décrits en-dessus (chaque point rouge représente un segment lié à un capteur - 17 au total) :

2. Degree of Freedom (Degrés de liberté)

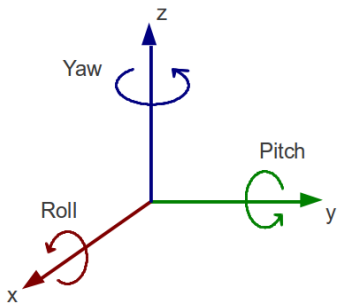


FIG. 2.6: Système de coordonées *right handed Z-up* avec les différentes orientations

Segments	**Joints**	**ID**	**Segments**	**Joints**	**ID**
Pelvis ●	-	-	LShoulder ●	LShoulder	10
L5	L5S1	0	LUpperArm ●	LUpperArm	11
L3	L4L3	1	LForeArm ●	LElbow	12
T12	L1T12	2	LHand ●	LWrist	13
T8 ●	T9T8	3	RUpperLeg ●	RHip	14
Neck	T1C7	4	RLowerLeg ●	RKnee	15
Head ●	C1Head	5	RFoot ●	RAnkle	16
RShoulder ●	RShoulder	6	RToe	RToe	17
RUpperArm ●	RUpperArm	7	LUpperLeg ●	LHip	18
RForeArm ●	RElbow	8	LLowerLeg ●	LKnee ;	19
RHand ●	RWrist	9	LFoot ●	LAnkle	20
-	-	-	LToe	LToe	21

TAB. 2.1: *Noms des articulations et des segments du squelette humain selon Xsens*

Utilisation et fonctionnement de Xsens

Xsens utilise un protocole d'envoi de données expliqué dans le manuel: *MVN real-time network streaming - Protocol Specification* [29]. Son fonctionnement se base sur l'envoi des informations des capteurs à une antenne qui est connectée à l'ordinateur puis le logiciel *MVN Analyse* organise tout sous forme de paquets d'octets (bytes). Ici, nous allons expliquer les paquets principaux et lesquelles on recupére pour la téléopération de Pepper.

Les données de capture de mouvement sont échantillonnées et envoyées à intervalles réguliers dont la longueur dépend de la configuration de MVN Analyze/Animate. Les taux d'échantillonnage habituels se situent entre 60 et 240 Hertz. Le taux de mise à jour

du flux réseau en temps réel peut être modifié indépendamment. Un *datagramme* est un paquet de données dont son contenu est défini par un protocole spécifique. Par exemple, les positions et la rotation de tous les articulations du corps à un instant donné sont envoyées comme une ou plusieurs UDP[3] *datagrammes*.

Chaque paquet de données ou *datagramme* commence par une en-tête (ou *header*) de 24 octets suivi d'un nombre variable d'octets pour chaque articulation, selon le type de *datagramme* sélectionné. À un instant t, selon le protocole choisi, le logiciel *MVN Analyse* envoi une série de *datagrammes* par UDP dont le premier est le *datagram header* et le reste dépendent du protocole.

L'en-tête de 24 octets définit le type de paquet de données que nous utilisons et donc les informations que nous allons recevoir. La Figure 2.7 montre la structure du *datagramme header*:

FIG. 2.7: En-tête du datagramme. SOURCE: [29]

Les 6 premiers octets de la chaîne décrivent l'identification du protocole de données que nous recevons, sous forme de *MXTP* et un numéro à deux chiffres. Il existe un total de 15 types différents de protocoles de datagrammes. Dans cette section, nous allons nous concentrer sur les paquets de données *MXTP01*, *MXTP02* et *MXTP20* qui sont les plus intéressants.

- **MXTP01** :

  Ce paquet, *Segment data Euler*, est composé de 28 octets répartis comme indiqué dans la figure 2.8 et qui définissent les coordonnées 6D des segments du modèle. Les coordonnées sont exprimées sur une base de coordonnées *right handed Y-up*. Chaque vecteur de position et vecteur de rotation est précédé de l'ID de segment

3. User Datagram Protocol : L'un des principaux protocoles de télécommunication utilisés par internet

FIG. 2.8: Segment data Euler. SOURCE: [29]

qui est décrit dans le Tableau 2.1.

(**Attention!** Le segment du bassin prend le numéro d'ID 0 donc avec ce protocole, chaque ID est +1 par rapport au Tableau 2.1) .

L'inconvénient de ce protocole est que le système de coordonnées utilisé est fixe, ce qui ne suit pas la logique définie car on n'exporte pas un DoF de l'opérateur relatif à une liaison rotule, mais une rotation d'un segment par rapport à un repère fixe. Par contre, c'est intéressant dans le sens où ce protocole permet d'extraire les position des segments (par exemple, *RHand*).

– **MXTP02**

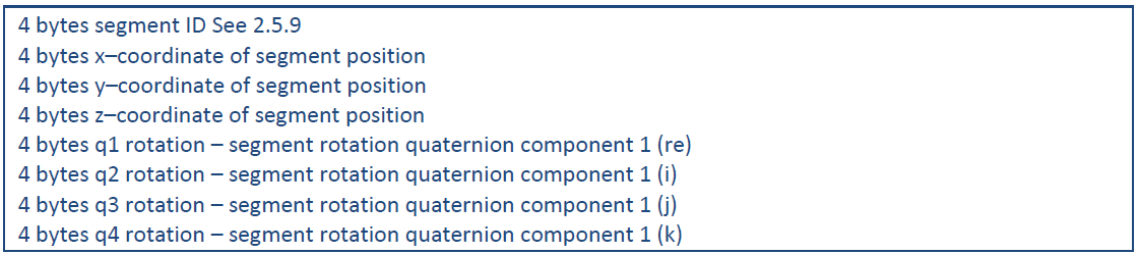


FIG. 2.9: Segment data quaternion. SOURCE: [29]

Ce paquet, *Segment data quaternion*, est composé de 32 octets répartis comme indiqué dans la figure 2.9. Très similaire au paquet précédent, sauf pour l'expression de la rotation des segments, où on extrait le quaternion. Données exprimées selon un système de coordonnées *right handed Z-up* fixe.

– **MXTP20**

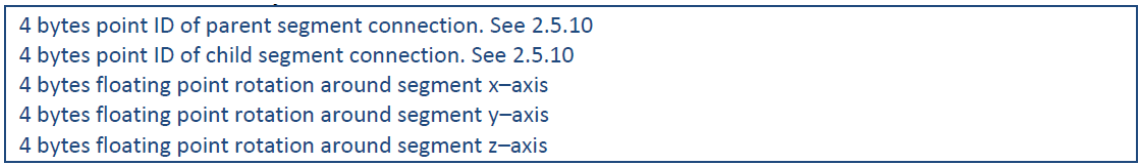


FIG. 2.10: Joint angles data. SOURCE: [29]

Ce paquet, *Joint angles*, est composé de 20 octets répartis comme indiqué dans la figure 2.10. Les 8 premiers octets décrivent le segment enfant et parent ID de

l'articulation, ce qui décrit une liste rangée selon le tableau 2.1. Ce protocole nous envoie le vecteur de rotation dans le système de coordonnées de l'articulation (*right handed Y-up*), ce qui est exactement ce que nous recherchons. Ainsi, ce paquet est celui utilisé pour extraire et stocker les DoF humaines. Par contre, les données seront traitées pour les obtenir sous une base *right handed Z-up*.

2.1.2 SoftBank Robotics

La technique du *Danseur d'Ombres* peut seulement être utilisé avec un robot qui va imiter les mouvements du téléopérateur. En particulier, le robot le plus utilisé au sein de l'*Association Robots!* est ***Pepper*** qui est créé par *SoftBank Robotics* (voir Figure 2.11). Puis à son côté on peut voir un autre robot créé par la même compagnie: **Nao** (Figure 2.12). *SoftBank Robotics* (anciennement *Alderaban*) est une entreprise robotique qui vise à la démocratisation des robots accompagnateurs.

FIG. 2.11: SoftBank Robotics robot PEPPER.

FIG. 2.12: SoftBank Robotics robot NAO.

Ces deux robots ont fait l'objet d'étude pendant mon stage de fin d'études et dans cette section vont être expliqués ses aspects les plus importants.

Pepper

Pepper es un robot humanoïde de 1.2 mètres de large et environ 60 kilos. **Un robot humanoïde est un robot conçu pour avoir une apparence humaine: deux bras, deux jambes, une tête, un tronc qui rassemble des cinq éléments sous forme de poitrine.** En plus, certains robots humanoïdes ont des yeux, une bouche, des doigts ainsi que certains non pas de jambes, comme Pepper, mais une plateforme avec des roues.

La complexité d'avoir deux jambes va être abordé dans le chapitre 4 où le problème fondamental est l'équilibre. Pepper dans ce sens est toujours équilibré et donc offre beaucoup plus d'usages, e.g., en communication: conférences, accueils, démonstrations d'IA [4], etc...

Plus dans la partie technique, Pepper possède 17 DoF dont 2 sont réellement des actionneurs (les deux mains seulement ont deux positions: fermé/ouvert), multiples capteurs répartis dans tout le robot, quatre microphones situés dans la tête, deux haut-parleurs situés dans les côtés de la tête, etc... Toute l'information technique de Pepper est sous libre accès dans [23]. Dans l'étude du hardware robotique nous intéresse surtout:

- Le CPU (dans la tête) quad-core, 2GHz et 4GB de RAM.
- L'information des articulations (voir Figure 2.13 et Table 2.2) [5].

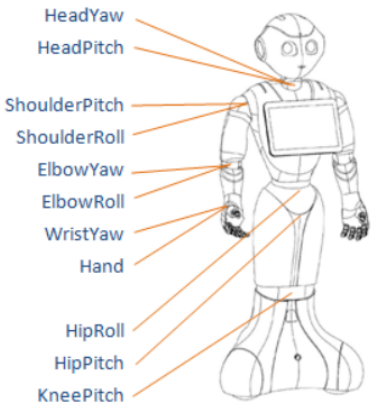


FIG. 2.13: Articulations de Pepper. SOURCE: [22]

Joints	**ID**	**Joints**	**ID**
KneePitch	0	LShoulderPitch	8
HipPitch	1	LShoulderRoll	9
HipRoll	2	LElbowYaw	10
RShoulderPitch	3	LElbowRoll	11
RShoulderRoll	4	LWristYaw	12
RElbowYaw	5	HeadYaw	13
RElbowRoll	6	HeadPitch	14
RWristYaw	7	RHand	15
-	-	LHand	16

TAB. 2.2: *Liste d'articulations de Pepper*

Pepper, et les robots de *SoftBank Robotics*, utilise une base de coordonnées *right handed Z-up* (comme celui de la Figure 2.6) avec x vers l'avant, y vers la gauche et z vers le haut du robot. En plus, dans [22], sont expliquées toutes les articulation (convention de signes pour les *roll, pitch, yaw*) et ses butées articulaires. Par exemple, pour l'articulation *RShoulderPitch* on a $\pm 119.5°$ de limite articulaire (voir Figure 2.14) :

4. Intelligence Artificielle
5. Noter que les articulations des bras sont symétriques.

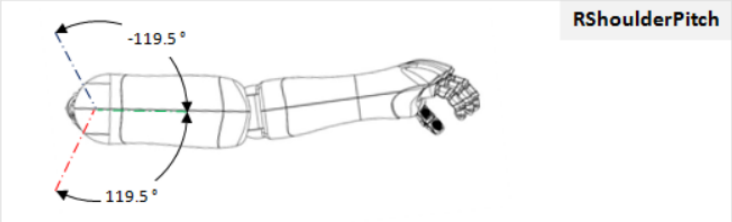


FIG. 2.14: Pepper *RShoulderPitch*. SOURCE: [22]

Nao

Par analogie avec Pepper, Nao a été construit sous la même logique mais sa morphologie lui contraint dans beaucoup d'usages. En effet, Nao est un robot humanoïde de 57 cm et environ 6 kilos. Son usage est surtout la recherche: modélisation de la marche humaine, modélisation de l'équilibre, accompagnement d'enfants et personnes âgées, etc...

Dans la partie technique, Nao possède 25 DoF dont 2 sont réellement des actionneurs (les deux mains seulement ont deux positions: fermé/ouvert), multiples capteurs répartis dans tout le robot, quatre microphones situés dans la tête, deux haut-parleurs situés dans les côtés de la tête, etc... Toute l'information technique de Nao est sous libre accès dans [19]. Dans l'étude du hardware robotique nous intéresse surtout:

- Le CPU (dans la tête) quad-core, 2GHz et 4GB de RAM. (Le même que Pepper)
- L'information des articulations (voir Figure 2.15 et Table 2.3) [6] [7].

Nao, utilise aussi une base de coordonnées *right handed Z-up* (comme celui de la Figure 2.6) avec x vers l'avant, y vers la gauche et z vers le haut du robot. En plus, dans [18], sont expliquées toutes les articulation (convention de signes pour les *roll, pitch, yaw*) et ses butées articulaires. Par exemple, pour les articulations *RShoulderRoll* et *RElbowRoll* on a (voir Figure 2.16) :

6. Noter que *RHipYawPitch* est mécaniquement liée à *LHipYawPitch* et donc elles composent un seul DoF. Pour cette raison il y a 26 articulations et 25 DoF.
7. Noter que les articulations des bras et jambes sont symétriques.

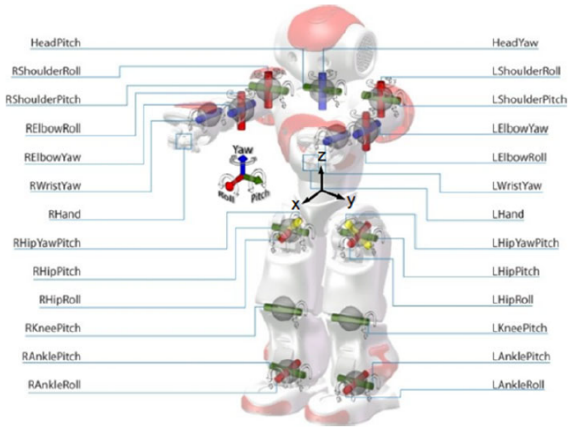


FIG. 2.15: Articulations de Nao. SOURCE: [18]

Joints	ID	Joints	ID
HeadYaw	0	LAnkleRoll	13
HeadPitch	1	RHipYawPitch	14
LShoulderPitch	2	RHipRoll	15
LShoulderRoll	3	RHipPitch	16
LElbowYaw	4	RKneePitch	17
LElbowRoll	5	RAnklePitch	18
LWristYaw	6	RAnkleRoll	19
LHand	7	RShoulderPitch	20
LHipYawPitch	8	RShoulderRoll	21
LHipRoll	9	RElbowYaw	22
LHipPitch	10	RElbowRoll	23
LKneePitch	11	RWristYaw	24
LAnklePitch	12	RHand	25

TAB. 2.3: *Liste d'articulations de Nao*

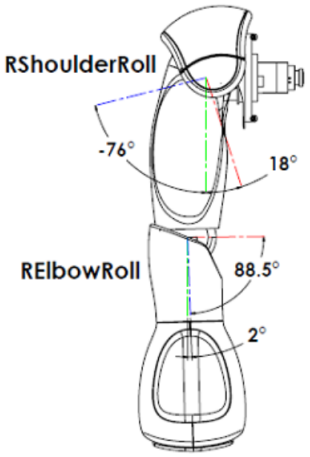


FIG. 2.16: Nao *RArm rolls*. SOURCE: [18]

C'est important de remarquer que Nao peut être divisé dans cinq chaînes d'articulations dont le torse est le point commun:

- Head → HeadYaw, HeadPitch.
- LArm → LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll, LWristYaw.
- LLeg → LHipYawPitch, LHipRoll, LHipPitch, LKneePitch, LAnklePitch, LAnkleRoll.

– RLeg → RHipYawPitch, RHipRoll, RHipPitch, RKneePitch, RAnklePitch, RAnkleRoll.
– RArm → RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll, RWristYaw.

Ceci est une approche que je vais utiliser pour modéliser le robot et qui va être approfondie dans la section 4.1.

Finalement, le CPU est l'endroit où toutes les fonctionnalités de Pepper et Nao vont être enregistrées, en particulier, *NAOqi* qui es le SDK[8] créé par *SoftBank Robotics* pour programmer et manipuler leur robots. En plus, les librairies concernant la partie d'immersion du téléopérateur pendant l'imitation, i.e., le son et le vidéo, doivent être aussi placées sur le CPU du robot. Les librairies choisies pour réaliser ce travail ont été créées par *GStreamer*.

2.1.3 Quadricoptère

Le drone acquis de la part de *Pilgrim Technologies* et ses spécificités sont présentés ici. La figure suivante (2.17) montre l'aspect général du quadricoptère:

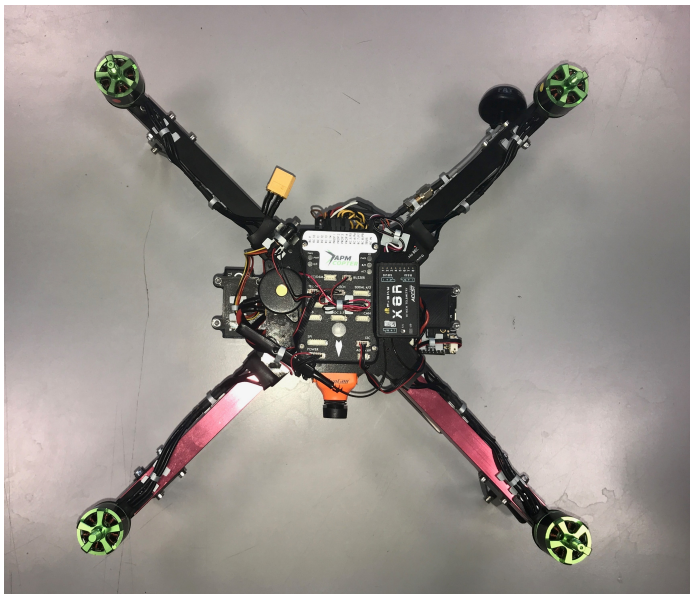


FIG. 2.17: Quadricoptère (vue du dessus)

Les composants présents sont:

– Quatre moteurs (et quatre hélices).
– Une caméra (voir Figure B.3).

8. Software Development Kit

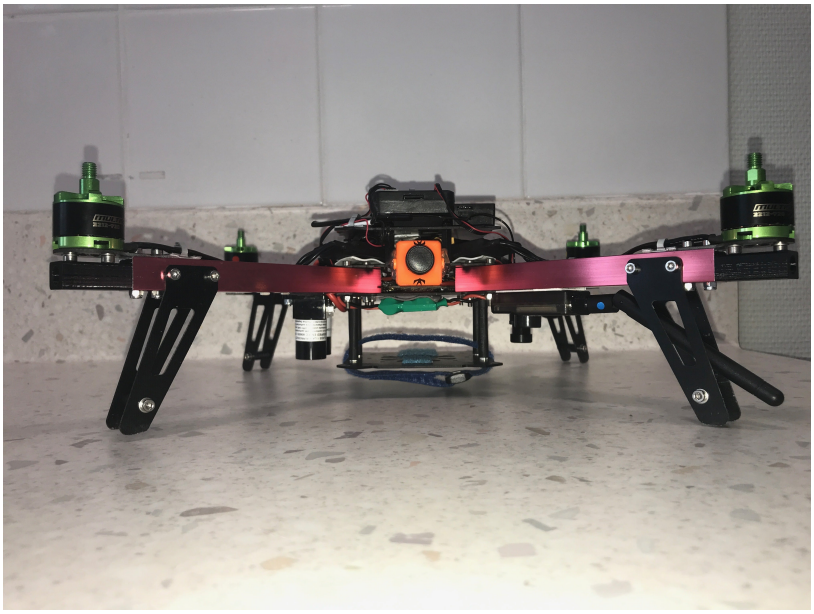


FIG. 2.18: Quadricoptère (vue de face).

- Un contrôleur de vol (voir Figure B.1).
- Deux antennes de radio.
- Un capteur de distance LIDAR-Lite V3 (voir Figure B.2).
- Une PX4FLOW smart caméra (voir Figure B.2).

Pour alimenter le contrôleur de vol et le reste des composants, on utilise une batterie LiPo[9] 11.1V (3S). En addition, le pilotage manuel du drone se réalise grâce a une manette comme celle de la Figure 2.19:

FIG. 2.19: Manette Taranis-Q-X7.

9. Lithium-Polymère

L'aspect le plus important du drone est le contrôleur de vol *APM Copter* qui contient tout le contrôle pour maintenir la stabilité, appliquer les modes de vol et contrôler les quatre DoF du drone: le *roll, pitch, yaw et throttle*. Pour cela, le software intégré dans celui-ci est *ArduPilot*, un projet totalement *Open Source* qui vise au développement de systèmes de contrôle pour des *copter, rover, submarine, etc...* [6].

2.2 Software

La software implémenté dans le robot qu'on utilise pour appliquer la technique du *Danseur d'Ombres* est très important puisqu'il doit être *Open Source* pour le bon développement et doit apporter les suffisantes fonctionnalités pour manipuler les moteurs du robots. Dans ce sens, *NAOqi* est un software convenable et dans cette section vont être exposées les fonctions fondamentales pour l'imitation.

2.2.1 NAOqi

Le software de *SoftBank Robotics* est *Open Source* et toute l'information nécessaire pour développer avec est dans [20]. Leur SDK est écrite sous C++ et Python et elle est supportée par les systèmes d'exploitation Windows, Linux et Mac. Avec *NAOqi* on commande Pepper et Nao, on récupère les informations de leurs capteurs et les positions de leurs chaînes d'articulations.

NAOqi est le nom du software principal qui s'exécute sur le robot et le contrôle. Le *NAOqi Framework* est le cadre de programmation utilisé pour programmer les robots de *SoftBank Robotics*. Il répond aux besoins courants de la robotique comme : parallélisme, ressources, synchronisation, création d'événements. Il permet aussi « une communication homogène entre les différents **modules** (mouvement, audio, vidéo), une programmation homogène et un partage homogène des informations. » [20]

NAOqi framework est ***Cross platform*** et ***Cross language*** qui permet de créer des applications sous Windows, Linux et MacOS en Python ou C++:

- Utilisation de Python : facilement exécutable directement sur le robot. Non utilisé dans cet étude.
- Utilisation de C++ : il s'agit d'un langage compilé, on utilise un outil de compilation croisée afin de générer un code capable de fonctionner sur le système d'exploitation

du robot : NAOqi OS. C'est le ***Cross platform***.

L'exécutable NAOqi qui est dans le robot s'appelle: ***broker***. Lorsqu'on se connecte à ce *broker*, il définit les librairies nécessaires pour faire fonctionner le code. Chaque librairie possède un ou plusieurs modules qui permettent l'accès aux **méthodes**: fonctions qui agissent directement sur le robot. La Figure 2.20 montre de façon simplifié la communication intrinsèque de *NAOqi*.

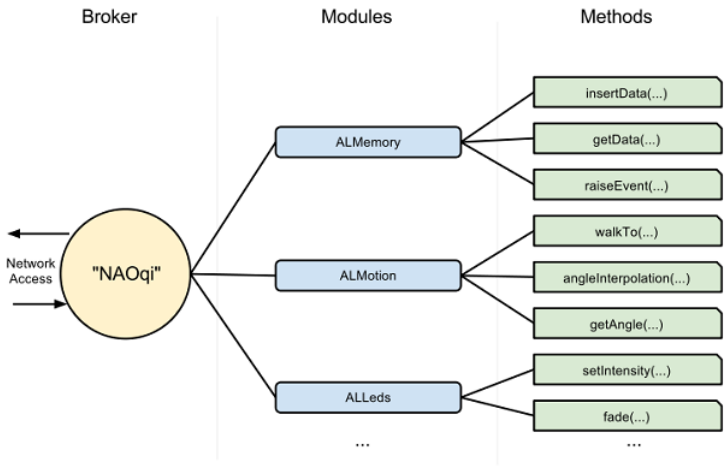


FIG. 2.20: Schéma de *NAOqi*. SOURCE: [20]

Après avoir installé *NAOqi framework* dans le PC (voir la guide sur [21]), on est capable de développer le code pour manipuler les robots. Les méthodes plus importantes sont:

- **ALMotionProxy::getAngles**(*robot_joint_names*, *bool*) → renvoie un vecteur avec les valeur articulaire du vecteur *robot_joint_names* de la posture en cours.
- **ALMotionProxy::setAngles**(*robot_joint_names*, *robot_joint_values*, *motors_speed*) → fixe les articulations du vecteur *robot_joint_names* aux valeurs du vecteur *robot_joint_values* à une vitesse définie dans la variable *motors_speed*.
- **ALMotionProxy::angleInterpolationWithSpeed**(*robot_joint_names*, *robot_joint_values*, *motors_speed*) → fixe les articulations du vecteur *robot_joint_names* aux valeurs du vecteur *robot_joint_values* à une vitesse définie dans la variable *motors_speed* suivant une courbe d'interpolation type *spline* pour des trajectoires non abruptes.

La grande différence entre *setAngles* et *angleInterpolationWithSpeed* est que la première n'est pas bloquante, i.e., pendant que le robot atteint la position, les autres fonctions peuvent démarrer (la boucle continue); la deuxième est bloquante et elle est utilisé pour assurer la stabilité. En l'occurrence, la méthode *angleInterpolationWithSpeed* est utilisée lorsqu'on calcule et impose des positions à la limite de la stabilité et le facteur inertiel

doit être très petit (en plus qu'on assure que la boucle soit arrêtée jusqu'à que le robot atteigne la position).

La grande problématique rencontrée avec le software de NAOqi est que même s'il est *Open Source*, les fonctions utilisées sont des **boîtes noires**. On ne peut pas y accéder dans le code de chaque méthode et, par conséquence, le développement reste limité. Par exemple, la seule commande que *NAOqi* offre pour manipuler le robot est une commande angulaire (il existe aussi une commande par position mais qui ne permet pas de développer avec) et donc il n'y a pas moyen d'agir ou commander directement les moteurs (avec une commande *torque-based* par exemple). Ceci est extrapolable à toutes les axes de recherche (e.g. la modélisation de l'équilibre de Nao) et reste un petit frein pour la recherche dans le domaine de la robotique humanoïde.

2.2.2 GStreamer

GStreamer es une librairie dédié à la création de composants multimédia et est un *framework* conçu pour gérer les flux multimédia. Les applications possibles sont très larges, depuis le streaming d'un audio/vidéo jusqu'à le mélange complexe d'audio et la transformation (non linéaire) de vidéo [11].

En particulier, l'utilisation des librairies *GStreamer* dans la technique du *Danseur d'Ombres* est extrêmement importante puisqu'elles s'occupent de la correcte immersion du téléopérateur: envoie du son, retour du son et retour de l'image.

GStreamer permet de transférer des données en passant par des *pipelines*. Ceux-ci sont les inter-connecteurs entre l'émetteur et le récepteur. Pour simplifier le schéma conçu par *GStreamer*, la connexion la plus basique est composée de deux éléments: source (src) et consommateur ou récepteur (sink), et leur connexion: le *pipeline*. La figure (2.21) suivante représente ce *pipeline* complet:

FIG. 2.21: Construction d'un *pipeline* basic. SOURCE: [11]

Pour construire un *pipeline* plus complet, on intègre plus d'éléments (e.g., vidéo + audio)

puis en les rassemble avec les fonctions de la librairie et on obtient un *pipeline* comme celui de la Figure 2.22:

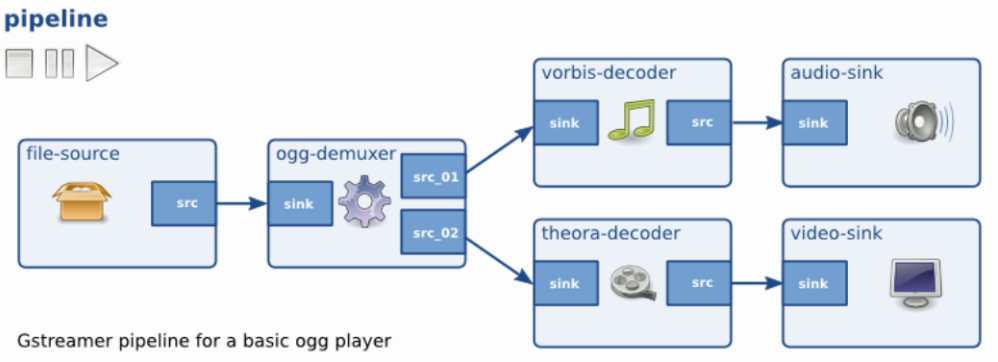


FIG. 2.22: Construction d'un *pipeline* plus complet. SOURCE: [11]

En l'occurence, les *pipelines* qui sont intéressants de créer pour apporter l'immersion à l'opérateur sont:

- **Émission du son**: port micro de l'ordinateur - **Réception du son**: haut-parleurs du robot.
- **Réception du son**: haut-parleurs de l'ordinateur - **Émission du son**: micro du robot.
- **Émission du vidéo**: caméra/s du robot - **Réception du vidéo**: lunettes d'immersion[10].

Lorsque ces *pipelines* sont activés ou lancés en même temps (sous un thread[11]) ou pas, c'est important de bien les fermer sinon ils rouleront en arrière plan. Un pipeline ne s'arrête que lorsqu'on lui demande avec notamment l'envoie d'un signal « *End Of Stream* » (EOS).

Voici un exemple de la construction d'un *pipeline* de façon manuelle pour connecter la caméra supérieure de Pepper avec le PC en stream:

- Pour envoyer la vidéo, le *pipeline* est :
  v4l2src device=/dev/video0 ! video/x-raw-yuv, width=320, height=240, framerate=30/1 ! jpegenc ! rtpjpegpay ! udpsink host=192.168.1.237 port=3000 sync=false
- Pour recevoir la vidéo, le *pipeline* est :
  udpsrc port=3000 ! application/x-rtp, encoding-name=JPEG, payload=96 ! rtpjpegdepay ! jpegdec ! autovideosink sync=false

10. Actuellement, la vision est basé sur un téléphone avec une application qui lance le *pipeline* de réception de vidéo et celui-ci intégré dans des lunettes type réalité virtuelle.
11. Envoi audio, retour audio et retour vidéo en même temps.

2.2.3 Ardupilot

ArduPilot est un software, un projet totalement *Open Source* qui vise au développement de systèmes de contrôle pour des *copter, rover, submarine, etc...* [6]. En particulier, le quadricoptère utilisé est un *copter* de quatre moteurs, avec une structure (ou *frame*) en *X* et avec le contrôleur de vol *APM Copter*.

Pour faciliter l'installation du software *ArduPilot* dans le drone, on utilise l'interface et logiciel *Mission Planner* qui nous sert pour réaliser la configuration totale du drone (faite dans ce cas précis par *Pilgrim Technologies*) avec l'aide de l'installation du *Copter firmware*. Toute l'information correspondante est offerte dans sa page web [7] et inclut des guides pour la construction, développement [5] et utilisation de n'importe quel UAV [12] (bien-sûr avec les contrôleur de vol indiqués).

L'accès au code du contrôleur de vol étant complet, sa structure se base sur :

- Le code du véhicule
- Librairies partagées.
- *Hardware abstraction layer* (AP_HAL)
- *Tools directories*
- *External support code* (i.e. mavlink, dronekit)

L'environnement du code interne du contrôleur de vol fourni par *ArduPilot* peut être sous Windows, Linux et MacOS et est écrit sous C++.

Dans l'expérimentation actuelle, on vise à tester une technique sur ce robot et on peut se baser au 99% du project *ArduPilot* pour les quadricoptères et c'est pour cette raison que uniquement les aspect du **code du véhicule** et des **librairies partagées** ont été approfondies. La cible doit donc être chercher où le contrôleur de vol reçoit les commandes de la manette et changer cette ligne par une fonction pour accéder à l'information de Xsens.

12. Unmanned Aerial Vehicle

2.3 L'équilibre

L'équilibrage statique d'un solide s'accomplie lorsque son CoM est aligné avec son CoS [13]. En particulier, lorsque le CoS correspond à la projection du CoM dans le sol.
D'après les similitudes de morphologie entre un robot humanoïde et un humain, la base de l'équilibrage d'un robot est analogique à celui chez l'humain [16].

L'équilibre du corps humain

Le CoM d'un humain est situé plus ou moins derrière le ventre. Le CoS d'un humain est situé entre ses deux pieds lorsqu'il est débout mais qui change s'il a une main par terre ou s'il est sur une jambe. L'équilibre statique se retrouve lorsque le CoS représente une projection dans le sol du CoM comm on peut observer dans la Figure 2.23:

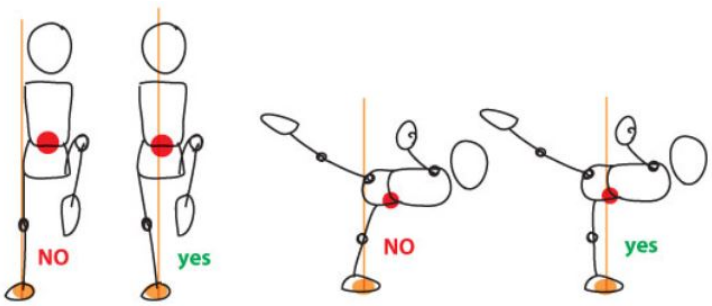


FIG. 2.23: Exemples de positions d'équilibre statique. SOURCE: [13]

L'humain a la possibilité de se stabiliser avec des *mouvements dynamiques* ou de maintenir une position de instabilité statique pendant quelques secondes de façon plus ou moins intuitive. Dans ce sens, le robot va être incapable de le faire puisque on va se baser pour l'équilibrer dans une **stabilité quasi-statique**. C'est-à-dire, les termes dynamiques ne vont pas être pris en compte même s'ils existent des études sur l'équilibrage dynamique (e.g.: ZMP [24], *angular momentum CoM control* [3], etc...) leur application en temps réel reste un domaine très complexe pour l'aborder dans 6 mois.

L'équilibre du robot humanoïde

L'équilibre quasi-statique se base sur assurer l'équilibre dans chaque position (positions statiques) lors d'un mouvement d'un robot. Ceci est possible lorsqu'on impose une tâche au robot qui est: *maintenir le CoM dans le polygone de stabilité*. Ce polygone est usuellement un rectangle entre les deux pied (en support double (DS)) ou la surface du pied

13. Le milieu entre tous les point qui sont en contact avec le sol.

(en support simple (SS)).

Pour atteindre cette tâche, on doit établir le DGM massique du robot et obtenir:

$$X_{CoM} = f(q) \tag{2.1}$$

Avec l'étude de l'évolution de CoM du robot et un polygone de stabilité que dépend de la position du robot et son support actuel, la stabilité est assurer. Dans la section 4.1 on va approfondir sur la modélisation du DGM du robot et comment implémenter l'équilibre.

On vient de poser la première tâche ou restriction du robot: on impose que son $X_{CoM} = (x_{CoM}, y_{CoM})$ reste à l'intérieur du *polygone de stabilité*. Cette tâche a deux DoF et pour un robot redondant, ils existent une infinité de possibilités pour atteindre cette tâche, c'est pour cela qu'on introduit le concept de **hiérarchisation des tâche ou *task prioritization***[25]. En effet, la configuration articulaire qui satisfait la tâche de l'équilibre est (IKM):

$$\Delta q = J_{CoM}^{-1} \Delta X_{CoM} \tag{2.2}$$

avec q le vecteur de positions articulaires de taille: le numéro de DoF, et X_{CoM} de taille 2. Si le robot est très redondant, comme par exemple, le robot Nao avec 25 DoF, l'équation antérieur montre l'infinité de solutions posibles.

2.4 Hiérarchisation des tâches

Si une tâche n'utilise pas tous les DoF du robot, les restant peuvent être utilisés pour résoudre autres tâches. La hiérarchisation des tâches ordonne ces tâches par priorité d'application, de tel façon que les tâches de priorité base ne dérangent pas celles qui ont une priorité supérieure d'exécution [25].

La problématique se base sur la faisabilité de plusieurs tâches dans un même espace articulaire, celui de robot, composé d'un nombre élevé de DoF. On commence par poser la définition mathématique de l'évolution du robot dans l'espace cartésien par rapport à ses variables articulaires (ou DKM) est:

$$\dot{e} = J\dot{q} \tag{2.3}$$

La définition ci-dessus est en effet, le *modèle cinématique directe* d'un robot avec J la matrice Jacobienne, e un vecteur de positions définit dans l'espace cartésien (ou *task space*) et q un vecteur de positions articulaires définit dans l'espace articulaire (ou *joint space*). Dans cette section, on ne va pas approfondir sur la définition du *DKM* (voir section 4.1) mais sur comment on peut inclure plusieurs tâches dans un problème de type quadratique.

On peut donc définir une tâche désirée exprimée dans l'espace cartésien $\dot{e}^*$ et résoudre l'équation (2.1) pour obtenir la configuration articulaire qui atteigne cette tâche. Ceci est appelé *Inverse Kinematics* et correspond à résoudre la suivante équation quadratique :

$$\text{Find } \dot{q}^* \in \underset{\dot{q}}{\text{Arg min}} \; ||J\dot{q} - \dot{e}|| \tag{2.4}$$

Ou plus généralement:

$$\text{Find } x^* \in \underset{x}{\text{Arg min}} \; ||Ax - b|| \tag{2.5}$$

L'équation (2.5) peut être résolue par la méthodologie des moindres carrées (LS [14]) en utilisant ce qu'on appelle la pseudo-inverse de Moore-Penrose A^+:

$$x^* = A^+ b \tag{2.6}$$

2.4.1 Equality Hierarchical Quadratic Program

Maintenant on va introduire la hiérarchie des tâches dans (2.5). L'explication est approfondie en [2] et explique que deux (ou plusieures) tâches peuvent être réalisées en même temps si la deuxième tâche est projeté sur l'espace nul de la matrice Jacobienne de la première (et plus prioritaire) tâche:

$$x^* = A^+ b + P\tilde{x}_2 \tag{2.7}$$

avec $P = I - A^+ A$ une projection dans l'espace nul de A (c'est-à-dire, $AP = 0$ et $PP = P$) et $\tilde{x}_2$ un vecteur arbitraire qui peut être utilisé comme une entrée pour satisfaire une deuxième tâche. Lorsqu'on introduit une deuxième tâche $A_2 x = b_2$, la solution complète qui résout (A_1, b_1) au mieux et (A_2, b_2) si possible est:

$$x_2^* = A_1^+ b_1 + (A_2 P_1)^+ (b_2 - A_2 A_1^+ b_1) + P_2 \tilde{x}_3 \tag{2.8}$$

14. Least-squares problem

avec x_2^* la solution pour deux hiérarchies différentes (premiers deux niveaux de priorité) et P_2 la projection de $\underline{A}_2 = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$.
Ceci est généralisé pour p niveaux de hiérarchie avec $k \in \{1 \dots p\}$ (travail [9]):

$$x_p^* = \sum_{k=1}^{p} (A_k P_{k-1})^+ (b_k - A_k x_{k-1}^*) + P_p \tilde{x}_{p+1} \tag{2.9}$$

avec $P_0 = I$, $x_0^* = 0$ et $P_k = P_{k-1}\tilde{P}_k$ la projection de $\underline{A}_k = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix}$ (travail [15]).

Toute cette recherche et lecture présentée ci-dessus permet d'établir les bases pour l'implémentation d'algorithmes pour le calcul de l'espace articulaire qui résout plusieur tâches classées dans une hiérarchie. Toute l'implémentation sur le robot Nao sera expliquée dans le chapitre 4.

Une autre ligne de recherche a été sur les restriction ou tâches qui suivent une inéquation et pas une équation comme tout ce qui a été déjà expliqué. Ceci **n'a pas été implémenté mais reste une ligne de recherche très intéressante pour augmenter la performance du *task prioritization***.

2.4.2 Inequality Hierarchical Quadratic Program

En particulier, on va aborder l'iQP [15] [1] qui est sous la forme:

$$\text{Find } x^* \in \{x,\ s.t. Ax \leq b\} \tag{2.10}$$

Si l'ensemble de contraintes est non faisable, alors x^* va être résolu comme la minimum valeur dans le sens des moindres carrées.

L'équation (2.5) peut être généralisé aux inéquations avec l'addition de la variable w, la *slack variable*:

15. Inequality quadratic program

$$\begin{aligned} &\min_{x,w} ||w|| \\ &s.t.\ Ax \leq b + w \end{aligned} \tag{2.11}$$

La *slack variable* peut relaxer la contrainte dans le cas de non faisabilité (voir plus précisément dans [3]). Cette forme (voir équation (2.11)), permet de définir aussi bien des eQP [16] que des iQP. Par exemple, la double inégalité $b^l \leq Ax \leq b^u$ est définie par:

$$\begin{bmatrix} -A \\ A \end{bmatrix} x \leq \begin{bmatrix} -b^l \\ b^u \end{bmatrix} \tag{2.12}$$

et l'égalité $Ax = b$ par:

$$\begin{bmatrix} -A \\ A \end{bmatrix} x \leq \begin{bmatrix} -b \\ b \end{bmatrix} \tag{2.13}$$

C'est très intéressant de lire l'article [14] et voir l'ampliation des eHQP [17] aux iHQP [18] en utilisant la *slack variable*. En effet, c'est [3] qui introduit le terme *slack variable* et à partir de lui, les articles [1], [2], [14] et [12] se basent sur cette variable. En particulier, ce dernier propose une méthode très intéressante de résolution de iHQP qui es computationnellement moins coûteuse que celle de [14].

Le premier niveau de hiérarchie est estimé atteint (ce qui es raisonnable avec une haute redondance) et donc $w_1^* = 0$ par hypothèse. Après, chaque niveau ($k > 2$) est résolue dans les espaces nuls des niveaux $k = 2$ jusqu'à $k - 1$:

$$\min_{z_k, w_k} ||w_k|| \tag{2.14}$$

$$s.t.\ A_1(x_{k-1}^* + Z_{k-1} z_k) \leq b_1 \tag{2.15}$$

$$A_k(x_{k-1}^* + Z_{k-1} z_k) \leq b_k + w_k \tag{2.16}$$

avec x_{k-1}^* la solution optimal pour les $k - 1$ premiers niveaux et Z_{k-1} la projection dans l'espace nul des niveaux 2 jusqu'à $k - 1$ (voir la section 2.5 pour mieux comprendre la matrice Z). Ceci peut être résolu avec un algorithme du type *Active Search Algorithm*. La

16. Equality quadratic program
17. Equality Hierarchical Quadratic Program
18. Inequality Hierarchical Quadratic Program

méthode de [12] est moins coûteuse computationnellement puisque le QP qu'on résout est plus petit que l'antérieur (la dimension de z_k diminue avec k).

2.5 Optimisation computationnelle.

Le temps réel est une restriction très importante et le calcul avec des matrices si grandes (numéro de DoF très grands) ne facilite pas le travail. C'est pour cette raison qu'une partie de ma recherche s'est basée sur l'optimisation computationnelle est plus concrétement sur le calcul matriciel.

2.5.1 Décomposition Matricielle

L'équation (2.6) est computationnellement très lente lorsque on applique cette résolution à des problèmes *inverse kinematics* avec des robots très redondants (> 20 DoF). En effet, c'est la computation de la pseudo inverse de matrices très grandes qui pose un problème et dans ce document vont être présentées deux types de décompositions matricielles:

- **SVD: *Singular Value Decomposition***
- **COD: *Complete Orthogonal Decomposition***

La méthode SVD décompose une matrice A tel que:

$$A = \begin{bmatrix} U_k & V_k \end{bmatrix} \begin{bmatrix} S_k & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Y_k^T \\ Z_k^T \end{bmatrix} = U_k S_k Y_k^T \tag{2.17}$$

où $\begin{bmatrix} U_k & V_k \end{bmatrix}$ et $\begin{bmatrix} Y_k & Z_k \end{bmatrix}$ représentent des matrices orthogonales carrées et S_k c'est une matrice diagonale avec les valeurs singulières. Grâce à cette décomposition, la pseudo-inverse A^+ correspond à:

$$A^+ = \begin{bmatrix} Y_k & Z_k \end{bmatrix} \begin{bmatrix} S_k^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_k^T \\ V_k^T \end{bmatrix} = Y_k S_k^{-1} U_k^T \tag{2.18}$$

C'est la méthode la plus précise numériquement pour obtenir une pseudo-inverse [10] et la plus robuste pour les mauvais conditionnements de A^+ quand le robot est proche d'une singularité. Par contre, elle est encore assez coûteuse à calculer.

De l'autre côté, la méthode COD décompose la matrice A tel que:

$$A = \begin{bmatrix} U_k & V_k \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L_k & 0 \end{bmatrix} \begin{bmatrix} Y_k^T \\ Z_k^T \end{bmatrix} = V_k L_k Y_k^T \tag{2.19}$$

où $\begin{bmatrix} U_k & V_k \end{bmatrix}$ et $\begin{bmatrix} Y_k & Z_k \end{bmatrix}$ représentent des matrices orthogonales carrées et L_k c'est une matrice inversible triangulaire vers le bas. Grâce à cette décomposition, la pseudo-inverse A^+ correspond à:

$$A^+ = \begin{bmatrix} Y_k & Z_k \end{bmatrix} \begin{bmatrix} 0 & 0 \\ L_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} U_k^T \\ V_k^T \end{bmatrix} = Y_k L_k^{-1} V_k^T \tag{2.20}$$

La COD est moins coûteuse d'implémenter que la SVD. Les algorithmes utilisés pour calculer la COD appliquent des transformations basiques (Givens ou Householder rotations) et sont aussi robustes que les algorithmes qui calculent la SVD (et plus faciles d'implémenter).

Maintenant qu'on a étudié des méthodes plus efficientes en calcul que la pseudo-inverse de Moore-Penrose, on va ajouter cette nouvelle écriture à (2.5). En plus, si on ajoute une hiérarchie dans les tâches, cela équivaux à résoudre (2.5) grâce à une **HCOD**[19].

Avant d'expliquer le HCOD, te travail [1] nous explique une réécriture de la matrice de projection en base matricielle pour optimiser le coût de calcul. On pose Z_1 une base de l'espace nul de A_1 (tel que $Z_1 A_1 = 0$ et $Z_1^T Z_1 = I$) et donc la projection dans l'espace nul de A_1 peut se réécrire :

$$P_1 = Z_1 Z_1^T \tag{2.21}$$

et donc (2.9) peut s'écrire:

$$x_p^* = \sum_{k=1}^{p} Z_{k-1} (A_k Z_{k-1})^+ (b_k - A_k x_{k-1}^*) + Z_p z_{p+1} \tag{2.22}$$

avec Z_k une base de l'espace nul de A_k et z_{p+1} un vecteur de la dimension de l'espace nul

19. Hierarchized Complete Orthogonal Decomposition

de $\underline{A}_p = \begin{bmatrix} A_1 \\ \vdots \\ A_p \end{bmatrix}$

Cette écriture est moins coûteuse computationnellement que (2.9) à cause de la différence dans la taille matricielle.

2.5.2 Hierarchical Complete Orthogonal Decomposition

Si on reprend l'équation (2.3) cinématique du début puis sa résolution en (2.6) et on applique les méthodologies étudiés ci-dessus (notamment on applique sur (2.6) la COD) pour obtenir:

$$x_1^* = A_1^+ b_1 = Y_1 L_1^{-1} U_1^T b_1 \tag{2.23}$$

Pour un deuxième niveau de hiérarchie, c'est un peu plus complexe. Dans ce document on va faire une présentation que des équations et pas des algorithmes de resolution.

Avec toutes les simplifications et modifications de l'écriture pour mettre en valeur la hiérarchie des tâches avec un coût computationnel plus bas, on retrouve que:

$$x_2^* = x_1^* + Z_1 (A_2 Z_1)^+ (b_2 - A_2 x_1^*) + Z_2 z_3 \tag{2.24}$$

$$x_2^* = \begin{bmatrix} Y_1 & Y_2 & Y_3 \end{bmatrix} \begin{bmatrix} Y_1^T x_1^* \\ Y_2^T \tilde{x}_2^* \\ z_3 \end{bmatrix} \tag{2.25}$$

avec $\tilde{x}_2^* = Y_2 L_2^{-1} U_2^T (b_2 - A_2 x_1^*)$ qui correspond a la contribution du deuxième niveau à l'optimal.

3

Abstraction de la technologie.

3.1 Rectification et traitement des données Xsens

La table 2.1 est reproduite ci-dessous pour faciliter la recherche et la référence des information.

Segments	**Joints**	**ID**	**Segments**	**Joints**	**ID**
Pelvis ●	-	-	LShoulder ●	LShoulder	10
L5	L5S1	0	LUpperArm ●	LUpperArm	11
L3	L4L3	1	LForeArm ●	LElbow	12
T12	L1T12	2	LHand ●	LWrist	13
T8 ●	T9T8	3	RUpperLeg ●	RHip	14
Neck	T1C7	4	RLowerLeg ●	RKnee	15
Head ●	C1Head	5	RFoot ●	RAnkle	16
RShoulder ●	RShoulder	6	RToe	RToe	17
RUpperArm ●	RUpperArm	7	LUpperLeg ●	LHip	18
RForeArm ●	RElbow	8	LLowerLeg ●	LKnee ;	19
RHand ●	RWrist	9	LFoot ●	LAnkle	20
-	-	-	LToe	LToe	21

TAB. 3.1: *Noms des articulations et des segments du squelette humain selon Xsens*

Ces articulations et surtout, leur identification (ID), vont être la base de l'extraction des valeurs articulaires. Chacune de ces articulations a associée un repère (dont son centre est le centre de la liaison) **MAIS** Xsens utilisera une base de coordonnées anatomiques pour exprimer ses rotations, i.e., pas un système de coordonnées XYZ conventionnel (voir

Figure 2.6). Les figures suivantes (voir Figure 3.1 jusqu'à 3.5) ont pour but de décrire l'anatomie du mouvement [4], d'expliquer pourquoi la figure 2.4 est si importante et quel est le système de coordonnées utilisé par Xsens pour calculer les angles des articulations (ou rotation des liaisons) :

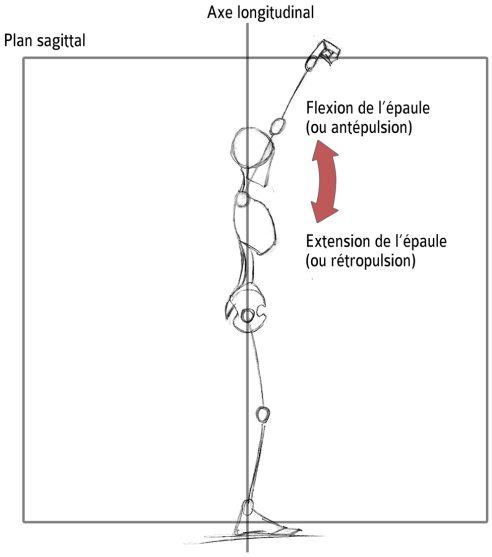


FIG. 3.1: Flexion et extension de l'épaule. [4]

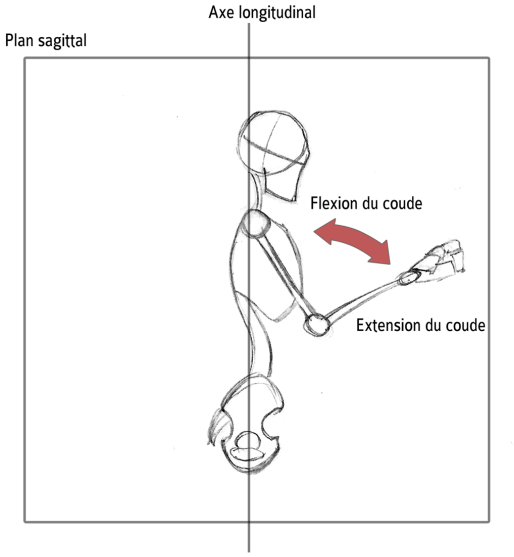


FIG. 3.2: Flexion et extension du coude. [4]

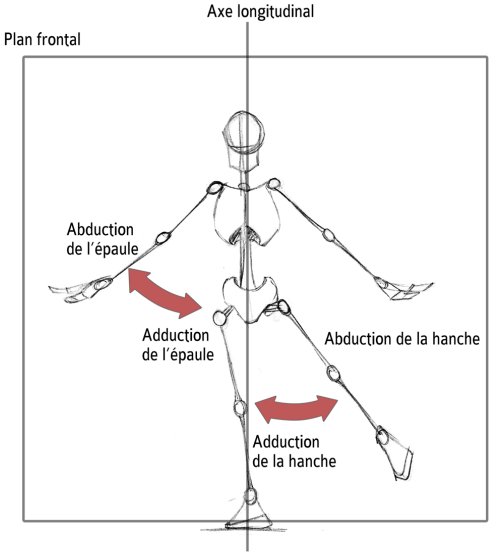


FIG. 3.3: Abduction and adduction. [4]

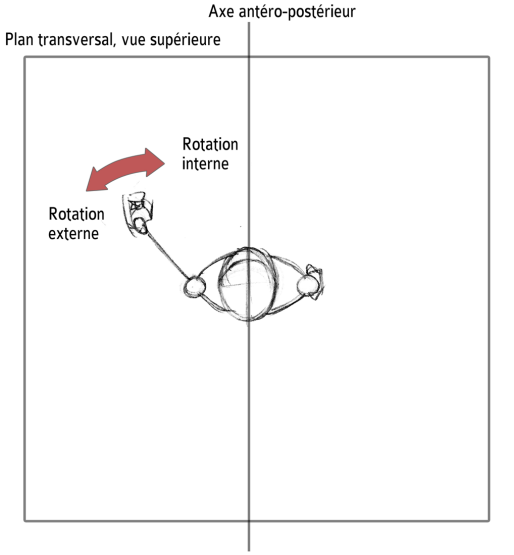


FIG. 3.4: Rotation externe et interne. [4]

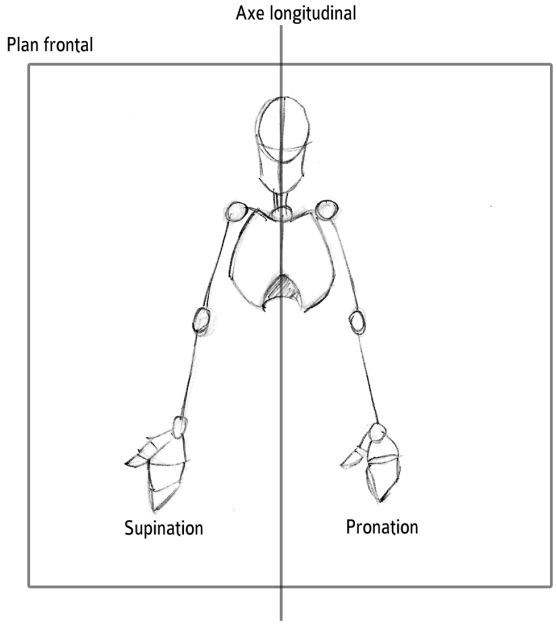


FIG. 3.5: Supination et pronation. SOURCE: [4]

Chaque articulation de la Table 3.1 va être exprimée dans un système de coordonnées anatomiques comme indiqué ci-dessus. Ainsi, toutes les articulations n'ont pas la même orientation parce que lorsque on réalise une abduction des deux coudes (+) dans un système de coordonnées conventionnel, ils rotent dans des directions opposées. Cette problématique sera résolue par une normalisation du système de coordonnées anatomiques Xsens, i.e., chaque articulation aura un système de coordonnées *right handed Z-up* avec l'analogie entre les plans du corps (voir figure 2.4) et les plans XYZ. Par exemple, l'abduction (+) normalisée de l'épaule droite est un *roll* négatif et son adduction normalisée (-) un *roll* positif. Le but est donc de récupérer la base décrite dans la Figure 2.6 et ce calcul a lieu lors du traitement des données envoyées par le logiciel *MVN Analyse*.

Pour comprendre les systèmes de coordonnées articulaires, on analyse le logiciel *MVN Analyse* et sa fonctionnalité graphique (voir Figure 3.6).
Après l'étude de la convention des signes des rotations anatomiques de Xsens, voici la synthèse des différentes articulations de la table 3.1 et leur système de coordonnées anatomiques:

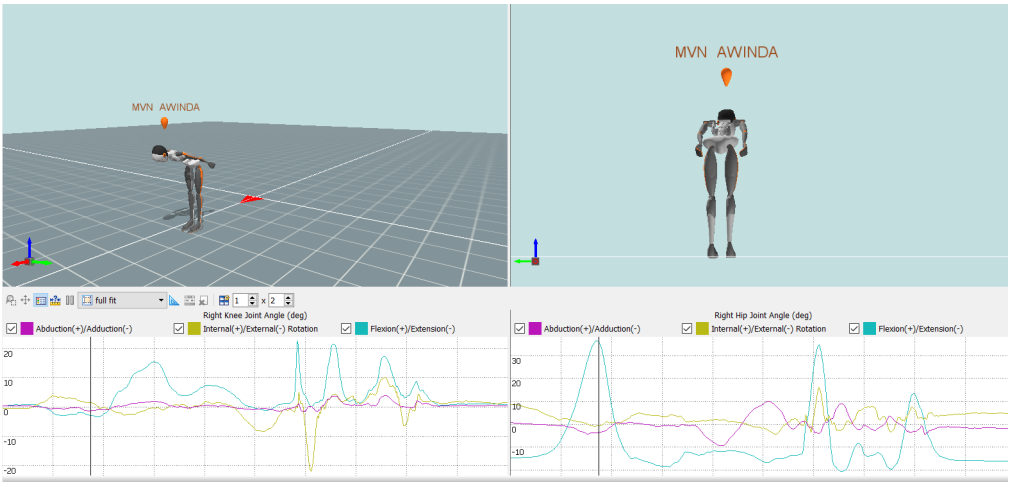


FIG. 3.6: Interface graphique de la capture des mouvements.

- **L5S1, L4L3, L1T12, T9T8, T1C7, C1Head** :
  - Plan frontal / coronal : Pli latéral droit (+) - Pli latéral gauche (-)
  - Plan sagittal : Flexion (+) - Extension (-)
  - Plan transversal : Rotation axiale (sens antihoraire (+) - sens horaire (-))
- **RShoulder, LShoulder, RUpperArm, LUpperArm, RHip, LHip, RKnee, LKnee, RToe, LToe** :
  - Plan frontal / coronal : Abduction (+) - Adduction (-)
  - Plan sagittal : Flexion (+) - Extension (-)
  - Plan transversal : Rotation interne (+)/externe (-)
- **RAnkle, LAnkle** :
  - Plan frontal / coronal : Abduction (+) - Adduction (-)
  - Plan sagittal : Dorsiflexion (+) - Plantarflexion (-)
  - Plan transversal : Rotation interne (+)/externe (-)
- **RElbow, LElbow, RWrist, LWrist** :
  - Plan frontal / coronal : Déviation radiale (+) - Déviation cubitale (-)
  - Plan sagittal : Flexion (+) - Extension (-)
  - Plan transversal : Pronation (+) - Supination (-)

En faisant l'analogie avec les informations exposées ci-dessus et une base *right handed Z-up* (voir Figure 2.6), le pré-traitement des données articulaires Xsens consiste en transformer le signe et le plan de appartenance de chacune des rotation vers la base normalisée.

3.1.1 Pré-traitement

Tout d'abord, comme on l'a remarqué précédemment, le protocole *MXTP20* envoi les données selon un système de coordonnées *right handed Y-up* (comme expliqué dans les Figures 2.5 et 2.10), puis une système de coordonnées anatomiques expliqué précédemment. Le repère situé dans l'articulation est:

- Origine du repère : centre de la liaison rotule (approximative)
- X vers l'avant.
- Y vers le haut.
- Z pointant vers la droite.

Poursuivant l'analogie entre le modèle anatomique et le système de coordonnées XYZ, nous déduisons :

- le plan Frontal / coronal correspond au plan YZ,
- le plan Sagittal au plan XZ,
- et le plan transversal au plan XY.

Ainsi, nous déduisons que l'information transmise dans le protocole *MXTP20* correspond à des rotations anatomiques sous forme d'un repère *right handed Y-up*. Le pré-traitement consiste à réaliser la rotation du repère du protocole au repère de notre base normalisée (*right handed Z-up*)(voir Figure 3.7) puis vérifier que les signes anatomiques exposés ci-dessus correspondent bien à la convention des signes la nouvelle base.

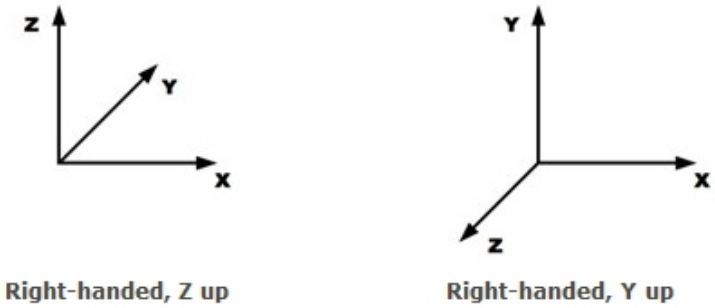


FIG. 3.7: Système de coordonnées right handed Z-up et Y-up

Par exemple, une flexion du coude droit (plan sagittal) est une rotation positive (l'extension est négative) et donc le protocole *MXTP20* enverra une rotation positive selon l'axe Z pour l'articulation **ID:8**. Dans notre base normalisée, ce mouvement de cette articulation correspond à un *pitch* (rotation suivant l'axe Y) négatif.

En conclusion, pour normaliser l'environnement Xsens, nous devons comprendre comment l'information est envoyée. La transformation n'est pas uniquement une rotation du repère, mais celle d'un cadre de coordonnées anatomiques basé sur un repère *right handed Y-up* à une base conventionnelle *right handed Z-up*. Ainsi, les changements apportés sont :

- Changer les *pitch* en *yaw* (et vice versa).
- Examiner chaque articulation et son propre cadre anatomique (signe) et les traduire à un système de coordonnées *right handed Z-up*. Il n'est pas correct d'effectuer la transformation directe de *pitch* à *yaw* parce que chaque articulation a un cadre anatomique, donc, e.g., l'abduction sera toujours une rotation positive indépendamment de la direction de l'axe X, donc chaque articulation doit être étudiée une par une.

3.1.2 Implémentation

Tout d'abord, la façon la plus simple de récupérer le datagramme est d'utiliser une classe qui contienne tous les éléments. Dans la figure suivante (3.8) nous pouvons voir les éléments du paquet *MXTP20* (Figure 2.10) définis dans la classe **joint_buffer**.

```
class joint_buffer
{
public:
    int id_parent;
    int id_child;
    float rotation_x;
    float rotation_y;
    float rotation_z;
    std::vector<float> rotation;
    void define(char buffer[],int buffer_count);
};
```

FIG. 3.8: Définition de la classe

Dans la fonction principale nous connectons l'ordinateur au port Xsens via un socket (UDP) pour recevoir tous les paquets de données et les stocker dans une variable de la classe *joint_buffer*. Les données étant de type octet, on doit la traiter avant de l'utiliser grâce a la fonction ***define*** (voir Figure A.3 et A.2).

Maintenant que nous avons le vecteur d'angles d'articulation humaine normalisés (type **joint_buffer**), c'est exactement une configuration d'articulation $q = \{q_1,q_2,....q_i\}$, avec i = numéro d'articulations, et chaque q_i inclut son *roll, pitch et yaw*. C'est le vecteur q qui sera envoyé à un robot.

3.1.3 Rectification

La complète normalisation des valeurs du vecteur articulaire q nécessite d'une rectification.

Cette **rectification**, ou **calibration**, se base sur ce qu'on va appeler la ***N-pose*** (voir Figure 3.9). Ceci es une posture, pré-définie en Xsens, humaine dont l'individu se pose débout et relaxé.

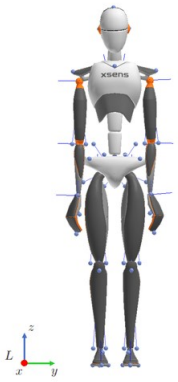


FIG. 3.9: N-pose. SOURCE [30]

Avant de commencer l'imitation, le robot se met en tension ou ***wakes up*** et la configuration articulaire dans cet instant s'enregistrera comme *robot_wakeup_joint_angle* et servira pour calibrer le robot. De l'autre côté, à cet même instant $t = 0$, on enregistre les valeur articulaires de l'opérateur comme *human_Npose_joint_angle* que servira pour calibrer le téléopérateur. Les deux vecteurs définis ici constituent les positions de base et donc d'une *zero-pose* ou *N-pose*.

Lorsque l'humain ou le robot sont dans une posture déterminé, les valeurs articulaires dépendent de la base à laquelle ils ont été conçus, e.g., la valeur articulaire des épaules dans la N-pose chez Xsens vaut environ 0 rad et chez Pepper (ou généralement, NAOqi) vaut environ 1.55 rad[1]. C'est précisément à cause de ce décalage, que **c'est nécessaire une calibration ou rectification, pour que les valeurs articulaires de l'opérateur puissent être associées à celles du robot sans importer la base dans laquelle elles ont été définies**. On établit donc que la posture ***N-pose*** est la posture de base, où toutes les valeurs articulaires valent 0 et tout changement à partir de cette position sera un changement relatif à cette posture. On s'assure pas seulement qu'un δq de l'opérateur équivaut à un δq du robot mais aussi qu'un q de l'opérateur correspond au même q du

1. 1,55 $rad \approx 90°$

robot (s'il est dans les butées articulaires du robot).

Les valeurs articulaires rectifiées correspondent à:

$$robot\_rectified\_joint\_angle = robot\_current\_joint\_angle - robot\_wakeup\_joint\_angle \tag{3.1}$$

$$human\_rectified\_joint\_angle = human\_current\_joint\_angle - human\_Npose\_joint\_angle \tag{3.2}$$

Et finalement, la valeur articulaire rectifiée envoyée au robot correspond à:

$$robot\_current\_joint\_angle = human\_rectified\_joint\_angle + robot\_wakeup\_joint\_angle \tag{3.3}$$

On réussi à abstraire la base articulaire du robot de la technique ou méthode de téléopération.

3.2 Validation

Pour envoyer le vecteur des articulations au robot, on doit connaître quels sont les spécificités du robot, e.g., les DoF du robot en question. Pendant ce stage de fin d'études, j'ai rencontré trois robots:

- Pepper: 17 DoF.
- Nao: 27 DoF.
- Un quadricopter: 4 DoF.

La logique utilisée, comme elle a été décrite auparavant, associe les DoF de l'humain à celui du robot, i.e., associe le roll de l'articulation *RWrist* à un DoF du robot, e.g., à l'actionneur de la main droite de Pepper *(RHand actuator)*. Dans ce sens, tant que on dispose de DoF de l'opérateur, on peux associer sa valeur à un DoF d'un robot.

En plus, les DoF de l'opérateur correspondent à des rotations des liaisons rotules du modèle du squelette humain de Xsens. Dans ce sens, lorsque un robot humanoïde est téléopéré, l'association des DoF est directe: la valeur du roll de l'articulation *RShoulder* de l'opérateur est envoyée au DoF: *RShoulderRoll* de Pepper ou Nao.

Par rapport à l'utilisation précédente du *Danseur d'Ombres*, les résultats de téléopération sont identiques. La différence est que le code n'est plus spécifique à Pepper, où les rec-

tifications et normalisations des valeurs articulaires étaient faites que pour ce robot en particulier. Avec ces modifications, l'implémentation de la téléopération d'autres robots par la technique du *Danseur d'Ombres* sera directe, avec l'ajout du vecteur des DoF du robot, son rectification et son association à celui du téléopérateur.

Le code est donc divisé en trois parties inter-connectées mais avec des buts différents:

- Connexion avec Xsens et création de l'objet *robot*. Partie **danseur d'ombres**.
- Pré-traitement des données Xsens et normalisation de la base. Partie **xsens**.
- Envoyer les données au robot, l'initialiser (n DoF, butées articulaires, etc...) et créer les fonctions de téléopération. Partie **robot**.

L'objet robot créé, doit porter un nom (e.g., Pepper), qui permettra d'accéder aux fonction de téléopération mais avec les spécificités du robot créé (17 DoF). Chaque robot à un vecteur d'articulation différents qui doit être associés au vecteur d'articulations de l'opérateur. Si un nouveau robot est inclut, un nouveau nom doit être assigné avec ces spécificités correspondantes.
Cette logique permet donc d'abstraire la technologie du *Danseur d'Ombres* à n'importe quel robot ayant des DoF qui puissent être utilisés comme une entrée de commande. Par contre, la vrai validation arrive dans les chapitres suivants où cette logique est appliquée et où les vrais résultats sont visibles.

3.3 Limites et usages

La limite la plus importante de cette amélioration est de connaître bien la différence entre **téléopération** et **imitation**. Lorsque le robot manipulé par la technique du *Danseur d'Ombres* est un robot de type **humanoïde**, la téléopération du robot devient une imitation puisque l'association de toutes les articulations de l'opérateur à celles du robot humanoïde est directe à cause de la définition d'un robot humanoïde (voir section 2.1.2): il a donc la forme d'un humain avec la plupart des mêmes articulations. i.e., si l'humain et le robot ont, tout les deux, une articulation en commun, la valeur (rectifiée) de cette articulation récupérée par Xsens est **équivalente** à celle qui doit être envoyée au robot: **on obtient une imitation exacte**. Dans cette situation, la logique permet d'une abstraction de la technique avec n'importe quel type de robot humanoïde du marché, sans problème.

Par contre, si cette logique s'applique à « *l'imitation* » d'un humain par un quadricoptère, on trouve plusieurs problématiques. Déjà, un drone ne peut pas imiter à un téléopérateur puisque sa morphologie n'est pas la même que celle de l'humain. C'est dans ce type de cas où uniquement on peut parler de téléopération et pas d'imitation. Lorsqu'on suit la logique expliquée, on associe la valeur du *pitch* de l'articulation *RElbow* au DoF: *throttle* du quadricoptère. Cette association n'est plus une équivalence comme dans le cas de l'imitation. Cette contrainte n'est pas très importante puisque la solution est claire: établir les variables *throttle min* et *throttle max*, les associer à des valeurs articulaires (u_{min} et u_{max}) du téléopérateur et générer une fonction linéaire (par exemple) pour la transition du type:

$$throttle = throttle_{min} + \frac{(u - u_{min})}{(u_{max} - u_{min})} * (throttle_{max} - throttle_{min}) \tag{3.4}$$

L'association des DoF avec ces types de robots n'est pas intuitive et donc, un aspect à réfléchir (en plus de décrire les spécificités du robot, comme, e.g., le vecteur des valeurs articulaires ou DoF) est l'ergonomie du téléopérateur pour manipuler le robot de façon confortable pendant une durée raisonnable.

Une autre limite de l'amélioration présentée est la limite des *66 DoF* maximum que la technologie de Xsens permet d'extraire à l'humain. On rappelle que grâce aux capteurs inertiels *MVN AWINDA*, Xsens permet de modéliser l'opérateur vers un squelette de 23 segments et 22 liaisons rotules ou articulations. Chacune de ces liaisons ont 3 DoF (*roll, pitch, yaw*) et donc le total de *66 DoF*. En reprenant la définition ci-dessus: « *Cette logique permet donc d'abstraire la technologie du Danseur d'Ombres à n'importe quel robot ayant des DoF qui puissent être utilisés comme une entrée de commande* » contient la limite que le robot doit avoir au maximum 66 dégrées de liberté.

La dernière limite rencontrée est sur le contrôle effectué. En effet, toute la technique du *Danseur d'Ombres* se base sur un contrôle de position angulaire, ce qui en robotique est très peu fréquent à cause de **son usage**. Un robot industriel va vouloir que sont *end-effector* atteigne un position (e.g., pour prendre un objet) et donc, le contrôle fait sera par position (ou par vitesse). Comme il a été expliqué dans la section 4.1, un robot très redondant peut avoir une infinité de solutions pour une tâche de position (x,y,z) de son *end effector* et donc, avec une hiérarchisation des tâches (ref. section 2.4), on est capables d'atteindre la tâche principale tout en accomplissant d'autres tâches secondaires.

Le contrôle de position angulaire impose une valeur de position à tous les articulations. En effet, il n'y a pas plus de redondance. Ce contrôle empêche la réalisation de tâches secondaires (e.g. l'équilibre) mais permet une imitation très vraisemblable du téléopérateur. **Par rapport à l'usage de cette technique, le contrôle par position angulaire est parfait.** L'étude de l'usage de la technique est fondamentale pour comprendre et établir la direction du développement et aujourd'hui, l'*Association Robots!* emploi la technique du *Danseur d'Ombres* pour la divulgation scientifique sous forme de conférences télé-opérées par le robot Pepper. Dans ce sens, cet étude aide à comprendre les limites de la technique et les nouveaux usages possibles.

En addition, une autre limitation rencontrée avec ce type de contrôle est celui de la morphologie du robot humanoïde. Il est, en effet, très similaire à l'humain mais ses articulations sont des rotules parfaites quoi que les humains on a des articulation plus complexes, comme, e.g., les épaules sont des liaisons rotule avec la possibilité de se déplacer grâce à la clavicule. Dans ce sens, l'association équivalente entre les DoF d'un humain et un robot humanoïde est limité et n'est pas exacte. **Certaines articulations ne sont pas équivalentes à cause de la complexité du corps humain.**

4

Application sur le robot humanoïde NAO.

Ce chapitre présente le premier test pour valider la logique exposée dans ce document. Ce test est l'implémentation du *Danseur d'Ombres* dans le robot Nao présenté dans la section 2.1.2.

Au delà d'implémenter la technique comme elle est sur Pepper, Nao presente plusieurs spécificités qui demandent un travail approfondie:

- Numéro de DoF supérieur.
- L'équilibre.

La logique présentée dans le chapitre 3 a permis une implémentation directe du *Danseur d'Ombres* sur le corps supérieur de Nao: l'imitation est parfaite et en temps réel. Par contre, pour une téléopération complète du robot, il nécessite d'une fonction qui assure son équilibre pour sa sécurité et celle de son environnement. Pour accomplir cette mission, je me suis basé beaucoup sur la logique menée par Louise Poubel dans son travail [17]. Les modèles du robot (*DGM* et *DKM*) présentés dans [17] se basent sur une unique chaîne qui repose sur le pied droit en DS et en RS puis sur le pied gauche en LS. Certaines modifications on été appliquées concernant la modélisation du robot Nao:

- Il n'y a plus de une seule chaîne géométrique ni cinématique sinon 5: Head, LArm, LLeg, RLeg et RArm. Chaque modèle sera résolu pour chaque chaîne.
- Le plus important: le point de repère n'est plus le pied sinon le torse. Cela va permettre une diminution de complexité des modèles énormes et une diminution du temps de calcul puisque tous les calculs seront faits relatives au torse et donc les chaînes deviennent indépendantes entre elles (et fortement dépendantes au torse).

Le modèle du robot repose donc en 4 parties:

- Modèles géométriques des *end-effectors*: Head, LHand, LFoot, RFoot et RHand.
- Modèle géométrique du robot par rapport au CoM pour pouvoir calculer sa position en fonction de la configuration articulaire (moyenne pondérée des 5 chaînes + torse).
- Modèles cinématiques des *end-effectors*: Head, LHand, LFoot, RFoot et RHand.
- Modèle cinématique du robot par rapport au CoM.

4.1 Modèle géométrique et modèle cinématique.

4.1.1 DGM

Les modèles nécessaires pour définir le robot sont:

- Modèle géométrique direct des *end-effectors*.
- Modèle géométrique direct par rapport au CoM du corps entier.
- Modèle cinématique direct des *end-effectors*.
- Modèle cinématique direct par rapport au CoM du corps entier.

Mathématiquement, ces quatre modèles s'expriment respectivement:

$$X_{eff} = f(q) \tag{4.1}$$

$$X_{CoM} = f(q) \tag{4.2}$$

$$\dot{X}_{eff} = J(q)\dot{q} \tag{4.3}$$

$$\dot{X}_{CoM} = J_{CoM}(q)\dot{q} \tag{4.4}$$

Les équations (4.3) et (4.4) peuvent se discrétiser et donner:

$$\Delta X_{eff} = J\Delta q \tag{4.5}$$

$$\Delta X_{CoM} = J_{CoM}\Delta q \tag{4.6}$$

avec:

$$\Delta X_{eff} = X_{eff}^{desired} - X_{eff}^{current} \tag{4.7}$$

$$\Delta X_{CoM} = X_{CoM}^{d} - X_{CoM}^{c} \tag{4.8}$$

$$\Delta q = q^{d} - q^{c} \tag{4.9}$$

où X_{eff} représente la position XYZ du *end effector* et X_{CoM} représente la position XY du CoM total du robot.

Comme on peut l'observer ci-dessus, si on réussi à exprimer matriciellement/mathématiquement l'équation (4.1), on pourra calculer la position de chaque *end effector* pour n'importe quelle configuration articulaire.

Pour cela, on va définir les 5 chaînes cinématiques du robot NAO avec une base en commun: le torse. Les chaînes sont (voir Figure 4.1) avec les différentes articulations associées:

- Head → HeadYaw, HeadPitch.
- LArm → LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll, LWristYaw.
- LLeg → LHipYawPitch, LHipRoll, LHipPitch, LKneePitch, LAnklePitch, LAnkleRoll.
- RLeg → RHipYawPitch, RHipRoll, RHipPitch, RKneePitch, RAnklePitch, RAnkleRoll.
- RArm → RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll, RWristYaw.

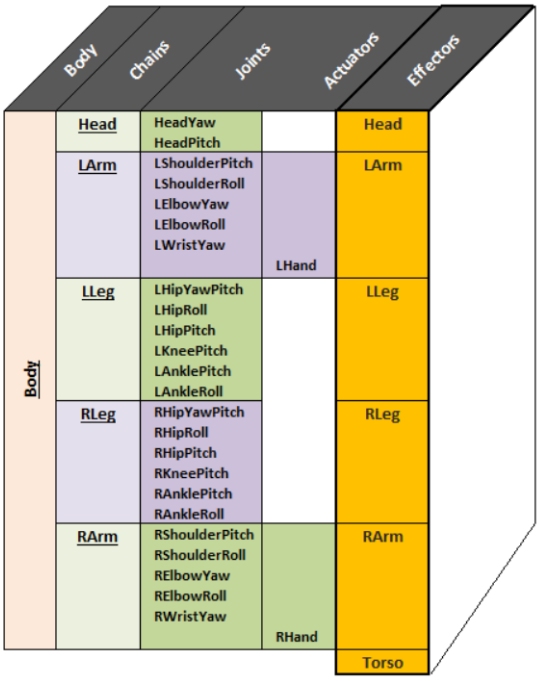


FIG. 4.1: Différentes chaînes et end effectors de NAO. SOURCE: [26]

Et les *end effectors* de chaque chaîne sont:

- Head → Head.
- LArm → LHand.
- LLeg → LFoot.
- RLeg → RFoot.
- RArm → RHand.

Le but du DGM est de pouvoir déterminer la position de ces 5 *end effectors* pour n'importe quelle configuration articulaire (selon l'équation (4.1)). Le torse va être la base commune à toutes les chaînes et les modèles seront calculées dans le repère XYZ du torse montré dans la Figure 4.2.

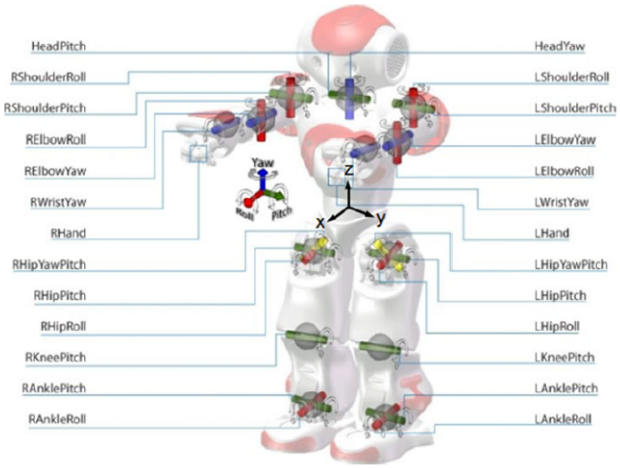


FIG. 4.2: Nao et ses différentes articulations.

Dans la Figure 4.2 on voir toutes les articulations selon un repère *right-handed Z-up* (avec *Yaw-Pitch-Roll* qui correspond aux rotation selon les axes positifs Z, Y et X respectivement). On veut donc obtenir une transformation qui permette de représenter la position des *end effectors* en fonction des articulations de la chaîne cinématique dans le repère du torse. C'est-à-dire, trouver une matrice de transformation T tel que:

$$Pos_{TORSO} = T^{eff}_{TORSO} Pos_{eff} \tag{4.10}$$

Avec la géométrie du robot [27], l'étude de chaque chaîne cinématique et les différents changements de repère pour exprimer toutes les articulations, on obtient les matrices de transformations suivantes:

$$\mathbf{T^{Head}_{Torso}} = \mathbf{A^{Neck}_{Torso}} \mathbf{T^{1}_{0}} \mathbf{T^{2}_{1}}; \tag{4.11}$$

$$\mathbf{T}^{\mathbf{LHand}}_{\mathbf{Torso}} = \mathbf{A}^{\mathbf{LShoulder}}_{\mathbf{Torso}}\mathbf{R_y}(\frac{-\boldsymbol{\pi}}{\mathbf{2}})\mathbf{T}^{\mathbf{1}}_{\mathbf{0}}\mathbf{T}^{\mathbf{2}}_{\mathbf{1}}\mathbf{A}^{\mathbf{LElbow}}_{\mathbf{LShoulder}}\mathbf{T}^{\mathbf{3}}_{\mathbf{2}}\mathbf{T}^{\mathbf{4}}_{\mathbf{3}}\mathbf{A}^{\mathbf{LWrist}}_{\mathbf{LElbow}}\mathbf{T}^{\mathbf{5}}_{\mathbf{4}}\mathbf{R_y}(\frac{\boldsymbol{\pi}}{\mathbf{2}})\mathbf{A}^{\mathbf{LHand}}_{\mathbf{LWrist}}; \quad (4.12)$$

$$\mathbf{T}^{\mathbf{LFoot}}_{\mathbf{Torso}} = \mathbf{A}^{\mathbf{LHip}}_{\mathbf{Torso}}\mathbf{R_z}(\frac{-\boldsymbol{\pi}}{\mathbf{2}})\mathbf{R_y}(\frac{-\mathbf{3}\boldsymbol{\pi}}{\mathbf{4}})\mathbf{T}^{\mathbf{1}}_{\mathbf{0}}\mathbf{R_y}(\frac{\mathbf{3}\boldsymbol{\pi}}{\mathbf{4}})\mathbf{R_z}(\frac{\boldsymbol{\pi}}{\mathbf{2}})\mathbf{T}^{\mathbf{2}}_{\mathbf{1}}\mathbf{T}^{\mathbf{3}}_{\mathbf{2}}\mathbf{A}^{\mathbf{LKnee}}_{\mathbf{LHip}}\mathbf{T}^{\mathbf{4}}_{\mathbf{3}}\mathbf{A}^{\mathbf{LAnkle}}_{\mathbf{LKnee}}\mathbf{T}^{\mathbf{5}}_{\mathbf{4}}\mathbf{T}^{\mathbf{6}}_{\mathbf{5}}\mathbf{A}^{\mathbf{LFoot}}_{\mathbf{LAnkle}}; \quad (4.13)$$

$$\mathbf{T}^{\mathbf{RFoot}}_{\mathbf{Torso}} = \mathbf{A}^{\mathbf{RHip}}_{\mathbf{Torso}}\mathbf{R_z}(\frac{-\boldsymbol{\pi}}{\mathbf{2}})\mathbf{R_y}(\frac{-\boldsymbol{\pi}}{\mathbf{4}})\mathbf{T}^{\mathbf{1}}_{\mathbf{0}}\mathbf{R_y}(\frac{\boldsymbol{\pi}}{\mathbf{4}})\mathbf{R_z}(\frac{\boldsymbol{\pi}}{\mathbf{2}})\mathbf{T}^{\mathbf{2}}_{\mathbf{1}}\mathbf{T}^{\mathbf{3}}_{\mathbf{2}}\mathbf{A}^{\mathbf{RKnee}}_{\mathbf{RHip}}\mathbf{T}^{\mathbf{4}}_{\mathbf{3}}\mathbf{A}^{\mathbf{RAnkle}}_{\mathbf{RKnee}}\mathbf{T}^{\mathbf{5}}_{\mathbf{4}}\mathbf{T}^{\mathbf{6}}_{\mathbf{5}}\mathbf{A}^{\mathbf{RFoot}}_{\mathbf{RAnkle}}; \quad (4.14)$$

$$\mathbf{T}^{\mathbf{RHand}}_{\mathbf{Torso}} = \mathbf{A}^{\mathbf{RShoulder}}_{\mathbf{Torso}}\mathbf{R_y}(\frac{-\boldsymbol{\pi}}{\mathbf{2}})\mathbf{T}^{\mathbf{1}}_{\mathbf{0}}\mathbf{T}^{\mathbf{2}}_{\mathbf{1}}\mathbf{A}^{\mathbf{RElbow}}_{\mathbf{RShoulder}}\mathbf{T}^{\mathbf{3}}_{\mathbf{2}}\mathbf{T}^{\mathbf{4}}_{\mathbf{3}}\mathbf{A}^{\mathbf{RWrist}}_{\mathbf{RElbow}}\mathbf{T}^{\mathbf{5}}_{\mathbf{4}}\mathbf{R_y}(\frac{\boldsymbol{\pi}}{\mathbf{2}})\mathbf{A}^{\mathbf{RHand}}_{\mathbf{RWrist}}; \quad (4.15)$$

avec $A = \begin{bmatrix} 0 & 0 & 0 & x \\ 0 & 0 & 0 & y \\ 0 & 0 & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$ qui représente une translation pure (voir dimensions du robot [27])

et $T = \begin{bmatrix} s_x & n_x & a_x & 0 \\ s_y & n_y & a_y & 0 \\ s_z & n_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ qui représente une rotation pure.

Puisque on travaille selon un repère *Yaw-Pitch-Roll*, on sait que n'importe quelle rotation peut s'exprimer selon l'équation suivante:

$$T = R_z(\gamma)R_y(\beta)R_x(\alpha) \quad (4.16)$$

avec γ, β, α qui représentent le lacet, tangage et roulis respectivement [1] et R_z, R_y et R_x les rotation élémentaires selon les axes z, y et x respectivement. L'ordre des transformations T^i_j est l'odre définit lors de la définition des chaînes cinématiques auparavant. C'est-à-

1. Yaw, Pitch, Roll

dire, e.g., dans l'équation (4.14), T_1^2 représente *RHipPitch* et aurai pu être représenté sous forme de $R_y(RHipPitch)$.

Pour obtenir le DGM par rapport au CoM, la logique est la même mais avec CoM de chaque élément: articulation + segment (voir la documentation de la position du CoM de chaque segment et articulation de Nao [28]). Après, une moyenne pondérée des 5 chaînes + torse permet de définir l'équation (4.2).

Pour calculer ces modèles de façon symbolique (chaque chaîne contient x articulations et donc, x variables symboliques) on utilise le logiciel de calcul *MATLAB*. Les fonctions *MATLAB* représentent l'enchaînement des transformation successives des éléments de chaque chaîne pour mettre en relation le torse et le *end-effector* correspondant. Ces représentations symboliques des variables vont définir de façon matricielle les équations (4.1) et (4.2) (puis vont être vérifiées).

La nouvelle logique de vouloir représenter toutes les *end effectors* et chaînes par rapport à un même élément (le torse) permet de travailler avec des matrices beaucoup plus petites et travailler encore plus en relative puisque les positions des *end effectors* sont maintenant indépendantes entre elles.

Par contre, le torse n'est pas un élément fixe et ne doit pas être un repère de base pour le contrôle du robot. Il nous permet de simplifier les matrices et la génération du modèle et d'augmenter la vitesse computationnelle. Dans notre approche, on va tout décaler au pied droit grâce à une transformation déjà calculée:

$$T_{RFoot}^{Torso} = (T_{Torso}^{RFoot})^{-1} \tag{4.17}$$

Maintenant, avec une simple opération matricielle, on obtient la position de tous les *end-effectors* (et le CoM) par rapport au *RFoot*, e.g.:

$$T_{RFoot}^{LHand} = T_{RFoot}^{Torso} * T_{Torso}^{LHand} \tag{4.18}$$

Les contraintes avec le pied fixe vont être expliquées dans la section 4.2.

4.1.2 DKM

A partir des équation (4.3) et (4.4) on obtient le DKM. Le but est de trouver la matrice Jacobienne J qui établi la relation entre $\dot{X}$ et $\dot{q}$. J(q) désigne la matrice jacobienne de dimension (m x n)[2] du mécanisme, égale à :

$$J(q) = \frac{\delta X}{\delta q} \tag{4.19}$$

Le calcul de la matrice Jacobienne peut se réaliser en dérivant le DGM, $X = f(q)$, tel que:

$$J_{ij} = \frac{\delta f_i(q)}{\delta q_j} \tag{4.20}$$

avec $i = 1, \cdots m$, $j = 1, \cdots n$ et J_{ij} est l'élément (i,j) de la matrice Jacobienne J.

Grâce à *MATLAB*, ses fonctionnalités et le calcul symbolique, on retrouve $J(q)$ et $J_{CoM}(q)$ des équations (4.3) et (4.4) à partir du *DGM* expliqué dans la section antérieure.

4.2 Implémentation

Comme il a été introduit au début du chapitre, l'implémentation de la technique du *Danseur d'Ombres* dans le robot humanoïde Nao avec la nouvelle logique a été directe. Il suffit de connaître le vecteur articulaire du robot et associer à chaque articulation un DoF de l'opérateur déjà extrait (grâce à la technologie Xsens), pré-traité et rectifié et l'imitation est implémenté.

L'immersion reste un peu plus complexe. En faite, la communication entre pipelines grâce à *GStreamer* (voir section 2.2.2) nécessite d'un émetteur qui créé le pipeline d'« émission » et que le récepteur fasse respectivement. Pour cela, on doit installer les librairies *GStreamer* sur le Nao pour que lorsqu'un élément soit créé dans le pipeline chez Nao, il en retrouve tous les composants nécessaires. En particulier, **le pipeline d'émission de vidéo, son et celui de réception de son**.

La partie la plus complexe de l'implémentation à été l'**équilibre**. La grande partie de la recherche est dans ce point. Grâce à toutes les explications exposées dans les sections 4.1,

2. avec m = 23

2.3, 2.4 et 2.5, une fonction ***équilibre*** a été testée.

Premièrement, on va énumérer les tâches (avec l'ordre respectif de priorité) qu'on va établir:

- Limites articulaires: jamais dépasser les butées articulaires (ni max ni min).
- Maintenir le pied droit fixe et parallèle au sol.
- Maintenir une position d'équilibre.
- Imiter les positions articulaire du téléopérateur (contrôle par position angulaire).

Pour cela, on va computer les *inverse kinematics* et mettre toutes ces tâches sous cette forme:

$$\dot{q} = J^{+}(q)\dot{X} \tag{4.21}$$

En discrétisant:

$$\Delta q = J^{+}(q)\Delta X \tag{4.22}$$

avec $\Delta q = q^d - q^c$ et $\Delta X = X^d - X^c$.

Pour la tâche la plus importante, on va utiliser un algorithme développé dans [15] qui assure que toute solution retrouvée dans l'équation (2.22) soit entre les limites des butées articulaires.

Maintenir le *RFoot* fixe et dans un plan est plus prioritaire que l'équilibre car celui-ci est calculé selon un référentiel fixé dans le *RFoot* et s'il bouge, le calcul de l'équilibre sera incorrect. Du coup, la deuxième tâche se représente mathématiquement comme suit:

$$\Delta q = J^{+}(q)\Delta X_{RFoot} \tag{4.23}$$

avec $X^c_{RFoot} = X^{XSENS}_{RFoot}$ et $X^d_{RFoot} = [0,0,0]$.

La logique qu'on va utiliser maintenant est que le IK va être calculée avant d'envoyer la commande d'imiter, i.e., la position *current* n'est pas la position actuelle du robot mais la position du téléopérateur (ou celle calculée par Xsens). Par conséquence, la position *desired* est la contrainte ou restriction qu'on veut imposer, i.e., rester fixe pour le *RFoot* et rester dans le polygone de l'équilibre pour le CoM.

L'avantage principale d'utiliser cette logique est que la dernière tâche va être assurée implicitement, car la méthode de résolution de l'équation (2.22) utilisée (rappel: *least-squares method*) minimise la solution, i.e., minimise Δq. Si q^c correspond à la position envoyée par Xsens, q^d va être calculée en fonction de minimiser Δq. En observe qu'on a maintenant que deux tâches ou contraintes à implémenter.

Pour résumer le paragraphe antérieur, il existe la position actuelle du robot (*current current*), puis la position envoyée par Xsens (*current*) et la position désiré (*desired*), que dans la plupart des cas sera q^c mais lorsque la configuration articulaire que Xsens envoie ne détermine pas un équilibre chez Nao, q^d représentera une configuration équilibré la plus proche possible de la configuration demandée par Xsens.

Pour déterminer si la prochaine configuration articulaire de Nao est équilibré, on utilise le *DGM* par rapport au CoM exposée dans l'équation (4.2) avec $q = q^{Xsens}$:

$$X_{CoM}^{future} = f(q^{Xsens}) \tag{4.24}$$

Si X_{CoM}^{future} est dans le polygone de sécurité, l'imitation se déroule sans rentrer dans l'algorithme d'équilibrage: **on suppose qu'en quasi-statique, lorsque le CoM se situe dans le polygone de stabilité, l'équilibre est assuré**.

Si X_{CoM}^{future} n'est pas dans les limites du polygone de sécurité, $X_{CoM}^{desired}$ prend la valeur de la limite du CoM établit et $X_{CoM}^{current}$ prend la valeur de X_{CoM}^{future}. En rentre donc dans le l'algorithme de l'équilibre et l'ajout de la deuxième tâche.

Si on reprend l'équation (2.7) et on fait l'analogie avec notre *IK*, on a:

$$\Delta q = J^+(q)\Delta X + P\Delta q_2 \tag{4.25}$$

i.e.

$$\Delta q = J^+_{RFoot}(q)\Delta X_{RFoot} + P_{RFoot}J^+_{CoM}(q)\Delta X_{CoM} \tag{4.26}$$

avec $P_j = I - J_j^+ J_j$ la projection dans l'espace nul de la Jacobienne de la première tâche ou tâche plus prioritaire.

Finalement, on applique une *Complete Orthogonal Decomposition* des deux matrices

Jacobienne J_{RFoot} et J_{CoM} pour que leur inverse ne soit pas si coûteuse computationnellement (voir section 2.5 équation (**??**)).

Ce différentiel Δq peut maintenant être additionné à la position actuelle et vérifier sa position de CoM, si les limites articulaires sont respectées, etc... Si tout est positif, on envoie $q^{new} = q^c + \Delta q$ au robot, sinon, on refait une autre itération pour calculer la configuration articulaire stable mais maintenant en partant de q^{new} qui est logiquement plus proche que q^c d'être satisfaisant.

4.3 Validation, limites et usages

La validation a été un 'succès'. Les guillemets sont très bien posés puisque les résultats obtenus ont permis une analyse en profondeur sur les usages et limites de la téléopération de Nao par la technique du *Danseur d'Ombres*.

En premier temps, j'ai observé que lorsque l'opérateur atteint une position très déséquilibré, l'algorithme trouve une configuration articulaire non satisfaisante (non équilibré) à cause que le *RFoot* bouge. Une possible amélioration est revoir le système de fixation du Nao, i.e., le repère externe qui permet d'obtenir la position du *RFoot* sans utiliser son *DGM*.

En deuxième, l'algorithme d'équilibrage, qui n'a pas été évalué en terme de performance de temps (à cause d'une manque de temps), est encore lent et par conséquence, ne permet pas une inclusion de l'immersion du téléopérateur. Si le temps réel n'est pas parfait, une immersion serait très déstabilisant et perturbant pour l'opérateur. C'est-à-dire, si le robot utilise l'équilibre, le *Danseur d'Ombres* doit avoir accés que aux articulations et ne pas à toute la partie vidéo et son. En addition, il ne peux pas non plus accéder aux fonctions de marche ni de tourner car se sont des fonctions de NAOqi (boîtes noires) et elles apportent beaucoup de mouvements dynamiques qui résultent en une déstabilisation lorsque le robot finisse la fonction.

Pour ces raisons, deux modes de fonctionnement pour Nao ont été implémentés:

- **Mode Artistique**: le robot imite tous les mouvements du téléopérateur en quasi temps réel mais il reste fixe. Le *Danseur d'Ombres* n'a pas accès au vidéo ni au son du robot, ainsi qu'aux fonctions de marche et tourner.
- **Mode Imitation**: le corps supérieur du robot imite tous les mouvement du corps

supérieur du téléopérateur. Le *Danseur d'Ombres* a accès à toutes les fonctionnalités qui décrivent la technique: l'immersion, le déplacement, etc...

La dernière limite rencontré a été le support pendant l'équilibrage de Nao. Pour l'instant, l'algorithme implémenté ne soutient que le double support et l'objectif, dans le mode artistique, est que Nao puisse se poser sur seulement le pied droit (RS) ou le pied gauche (LS).

5

Application sur un drone.

Dans cette section va être présentée le dernier test pour valider la logique exposée dans ce document. Ce test est l'implémentation du *Danseur d'Ombres* dans le quadricoptère présenté dans la section 2.1.3.

Le drone est un robot aérien, non-humanoïde, composé de quatre DoF: *roll, pitch, yaw* et *throttle* selon un repère de coordonées *right-handed Z-up*.

- Le *roll* est une rotation autour de l'axe X. Selon la morphologie des quadricoptères (une structure qui ne va pas être expliquée dans ce document), un rotation autour d'un axe X entraîne un mouvement rectiligne dans la **direction de l'axe $\boldsymbol{Y}$**.
- Le *pitch* est une rotation autour de l'axe Y qui va entraîner un déplacement dans **la direction de $\boldsymbol{X}$**.
- Le *yaw* est **une rotation autour de l'axe $\boldsymbol{Z}$** qui entraîne une rotation autour de lui même.
- Le *throttle* est la commande qui détermine la puissance envoyé aux quatre moteurs et donc qui effectue un changement de **hauteur**.

Ces quatre mouvements sont les seuls permis par un drone et donc ils composent les quatre DoF. Pour les téléopérer de façon manuelle, c'est nécessaire une manette à distance, qui se communique avec le contrôleur de vol et qui est l'interface opérateur-drone. **La cible doit donc être changer cette interface en excluant la manette et en intégrant la technologie Xsens.** Pour cela, on doit bien comprendre le software interne du contrôleur de vol, qui est *ArduPilot* et qui es totalement *Open Source*. **Le pré-requis de pouvoir développer sous un software externe est donc possible.**

Le contrôleur de vol fourni par *ArduPilot* présente multiples librairies pour harmoniser le code et le code particulier de chaque véhicule. En particulier, on va approfondir sur celui du *Copter* et ses modes de fonctionnement. En faite, le drone fonctionne par modes de fonctionnement ou vol (environ 20 différents) et c'est très intéressant cette structure car elle nous permet uniquement de créer un nouveau mode de vol qui actualise les valeurs de *roll, pitch, yaw, throttle* selon une fonction qui collecte les données envoyées par Xsens pour atteindre notre objectif. Chaque mode (ou en général chaque fonction) sous *ArduPilot* est composée d'une fonction *setup()* et *loop()*:

- *setup()* → initialisation de toutes les valeurs par défauts, elle est exécutée une seule fois lorsque le mode de vol est appelé.
- *loop()* → elle est appelée après le *setup()* et s'effectue sans arrêt. C'est dans cette fonction où toutes les opérations sont réalisées.

Dans la fonction *loop()* il y a une fonction appelée *run()* qui interprète les entrées de commande (par manette) et fixe les valeurs de *roll, pitch, yaw, throttle*. En faite, pendant la configuration du contrôleur de vol et la manette, on établit la communication entre les deux sous forme de pulsation ***PWM***: c'est la *Radio Calibration* [8].

Tandis que notre objectif est d'exclure la manette comme intermédiaire entre le téléopérateur et le robot, on se rend compte que le code est basée sur ce type de communication: **la longueur d'onde des pulsations PWM.**

Cette problématique permet d'analyser l'ensemble de l'expérimentation plus clairement. En effet, le drone a été conçu pour recevoir une communication particulière et c'est l'aspect le plus important. Pour pourvoir téléopérer un robot selon la technique du *Danseur d'Ombres*, **c'est nécessaire une communication PC ↔ robot** pour envoyer les données enregistrées par Xsens.

C'est donc la communication l'aspect qui va déterminer la faisabilité ou pas de l'expérimentation et en conséquence, c'est l'aspect qu'on doit garantir ou mettre en place pour appliquer la technique du *Danseur d'Ombres*.

5.1 Validation

La validation de cette expérimentation se base sur **l'analyse faite sur la problématique** exposée ci-dessus.

L'*Association Robots!* a acquis un quadricoptère fait à mesure qui intègre, comme il a été expliqué dans la section 2.1.3, une antenne radio pour communiquer avec la manette. C'est le seul moyen qui est à notre disposition pour émettre des signaux au contrôleur de vol est par le biais d'un émetteur radio. Cela suppose l'ajout d'autres éléments à la chaîne de communication déjà existante ce qui pose un danger pour le **temps réel**:

$$\text{Capteurs Xsens} \overset{WiFi}{\longleftrightarrow} \text{Recepteur Xsens} \overset{filaire}{\longleftrightarrow} \text{PC} \overset{WiFi}{\longleftrightarrow} \text{robot.}$$

Deviens:

$$\text{Capteurs Xsens} \overset{WiFi}{\longleftrightarrow} \text{Recepteur Xsens} \overset{filaire}{\longleftrightarrow} \text{PC} \overset{filaire}{\longleftrightarrow} \text{générateur PWM} \overset{filaire}{\longleftrightarrow} \text{émetteur radio} \overset{radio}{\longleftrightarrow} \text{robot.}$$

Le drone actuel n'est donc pas convenable pour l'application de la technique du *Danseur d'Ombres*. Par contre, l'analyse réalisé nous fixe la cible: **établir une communication PC $\leftrightarrow$ quadricoptère** et dans la section suivante, plusieurs solutions possibles sont exposées.

5.2 Limites et usages

J'ai rencontré deux grandes limitation:

- **La communication avec le quadricoptère**: le PC ne peux pas émettre des pulsation PWM à moins qu'il n'intègre un disposition externe, e.g., un *arduino* ou *PCTx* connecté avec un émetteur radio.
  En plus, il a été testé que les signaux radio n'étaient pas efficaces avec le vol indoor: on a observé beaucoup de pertes de signal ce qui n'est pas acceptable pour la sécurité du robot et l'entourage.
- **La taille du drone**: le vol indoor implique beaucoup de perturbations à cause des hélices. Par conséquence, cela provoque une instabilité extra à prendre en compte lors de la manipulation et lorsque l'opérateur est en immersion, c'est **très difficile** de se déplacer dans des zones fermés. En faite, le drone actuel pèse environ 1kg et fait $>$ de 30 cm de diamètre.

Ces limitations renforcent l'argumentation de la non-convenabilité du quadricoptère et pour cela, un analyse en profondeur a été réalisé sur les différents drones plus adaptés qui pourraient être testés dans cette expérimentation. En effet, trois solutions ont été proposées:

Solution 1: Adapter le drone.

Comme il a été expliqué dans la section précédente, le problème principal pour téléopérer le drone avec la logique du *Danseur d'Ombres* est la communication PC-DRONE. L'ordinateur n'a pas les moyens d'émettre une signal radio à la fréquence désirée ni le drone de recevoir des signaux WiFi. En effet, le récepteur de signal du drone attend une pulsation PWM à une fréquence de 2,4GHz. Pour cela, on peut utiliser deux technologies:

- **Arduino UNO**: il reçoit l'information de l'ordinateur et le transmet à la manette puis au drone. Ceci se fait en ouvrant la manette et la connectant électroniquement à l'arduino. (Voir [1])
- **PCTx**: Permet de connecter le PC et la manette pour contrôler le drone avec le PC. (Voir [2])

`Problèmes`: Le drone devrait être adapté à ce besoin et ne pas acheter plus de composants pour le contrôler. En plus, la communication radio n'est pas désirable et ceci ne change pas.

`Avantages`: On réussi à tester la possibilité de l'implémentation du *Danseur d'Ombres* sur le drone actuel.

Solution 2: Acheter un nouveau drone.

Acheter un nouveau drone plus adapté à nos besoins. Ceci repose la question: *qu'est-ce qu'on a besoin?*:

- Drone équipé d'un contrôleur de vol Open Source.
- Communication à ce contrôleur de vol via WiFi pour envoyer les signaux aux servo-moteurs.
- Drone d'une taille acceptable pour le pilotage indoor.

Après une recherche sur les drones petits du marché avec la possibilité d'être connectés par WiFi au Smartphone (l'application la plus répandue), la conclusion est mauvaise: il n'existe pas l'Open Source dans cette gamme du marché. Le contrôle est intégré dans une app (boîte noir) et le téléphone remplace la manette.

`Avantages`: L'ordinateur peut aussi se connecter au drone et à partir d'un code à partir du scrath: propre contrôle de vol, propre manipulation des 4 DoF (roll, pitch, yaw et throttle).

1. https://www.instructables.com/id/Computer-Controlled-Quadrotor-the-Easiest-Way/
2. http://www.endurance-rc.com/pctx.php

`Problèmes`: On n'a aucune piste sur quel contrôleur de vol l'entreprise utilise et donc on doit créer un propre (difficulté très haute) ou utiliser une librairie déjà existante (aucune confirmation qu'on puisse l'implémenter sur le drone).

Solution 3: Construire notre propre drone.

La section antérieur montre que le fait de construire son propre drone reste une option très bonne et en plus si on veut développer une technologie avec lui. Les composants essentiels d'un drone sont:

- Frame type Owl (voir Figure 5.1 (structure du drone) + Hélices.
- Moteurs + régulateurs de fréquence (ESC[3]).
- Caméra + émetteur de vidéo (VTx[4]).
- Contrôleur de vol + module WiFi ESP8622 (mode réseau qui attend information).
- PDB[5] (permet d'alimenter toutes les composants du drone).
- Batterie 3S ou 4S.

FIG. 5.1: Frame de type Owl.

Sans oublier les accessoires pour la construction du drone, tel que:

- Cables électroniques.
- Matériel pour souder.
- Ruban isolant électrique.

`Problèmes`: Les travaux de construction de drones déjà réalisés n'ont pas été faits avec le module WiFi. Il n'y a aucune certitude que le module WiFi puisse se connecter dans la branche RC-IN (Remote Control Input) du contrôleur mais il existe la possibilité d'embarquer un arduino NANO dans le drone pour envoyer les signaux PWM au contrôleur. Conclusion: Beaucoup d'incognitos.

3. Electronic speed control
4. Video Transmitter
5. Power Distribution Board

`Avantages`: Le montage d'un drone est extrêmement facile et pas cher. En achetant les composants 1 par 1 on peut créer quelque chose très adapté et les résultats peuvent réellement ouvrir beaucoup de voies dans l'implémentation du *Danseur d'Ombres* dans les différents robots: **si on réussi a transmettre les information XSENS via le module WiFi ESP8622 vers un autre robot**.[6].

Finalement, j'ai préparé un devis complet pour chacune des trois options qui a été présenté à l'*Association Robots!* et qui est présenté (que une partie) dans l'annexe B Figure B.6.

6. Il existe un tutoriel (pas très précis) sur comment construire un drone commandable par WiFi: https://www.instructables.com/id/Build-a-WiFi-Enabled-Micro-quadrotor/

6

Conclusions et perspectives

Dans la conclusion de ce rapport, le travail de **Sylvain Baud** doit être mentionné. Grâce à former équipe avec lui, on a réussi à rendre la technique du *Danseur d'Ombres* une application Windows, transportable à d'autres ordinateurs et de facile démarrage. En faite, c'est lui qui s'est occupé de toute la partie de la documentation sur Windows et de réaliser le GUI (voir Figure 6.1[1]) pour pouvoir téléopérer un robot par imitation de façon intuitive.

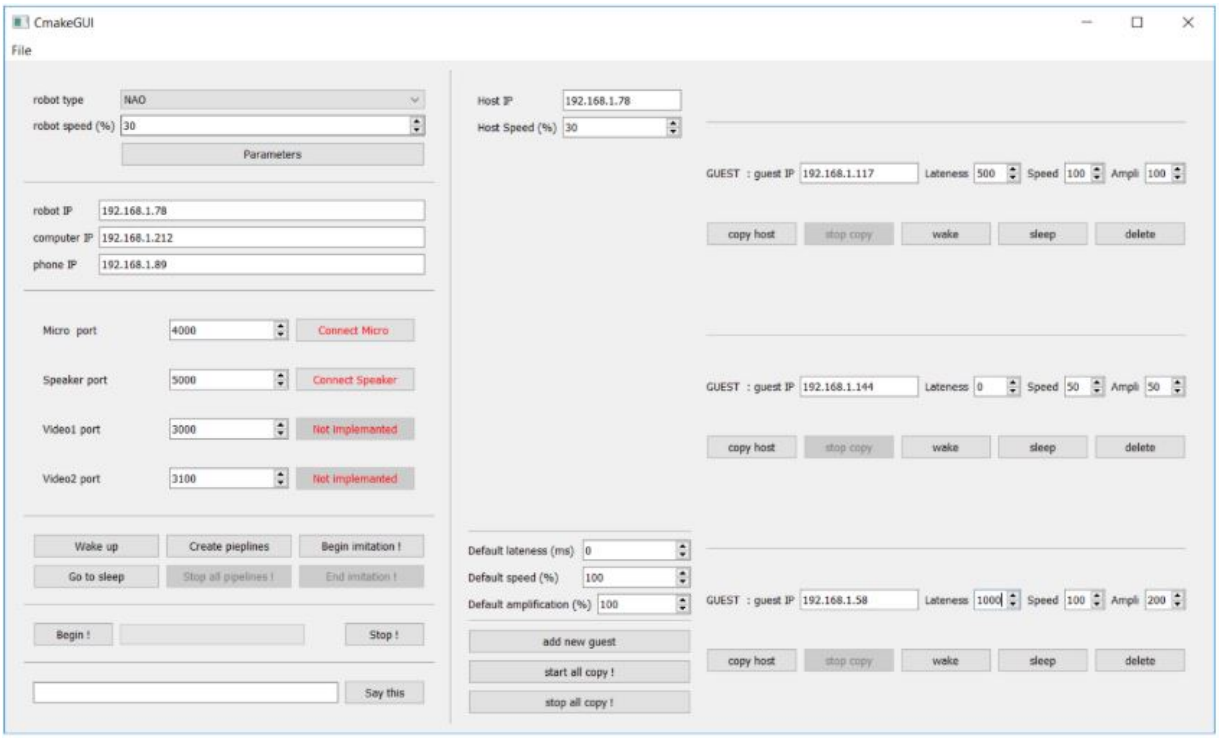


Fig. 6.1: Graphical User Interface.

1. Ce n'est pas la version définitive. La version définitive a été modifié par moi pour implémenter les modes opérationnels de Nao et une fonction de *démarrer avec opérateur seul*.

L'abstraction de la technologie du *Danseur d'Ombres* à différents robots est réalisable. Elle a été testé sur Nao avec succès mais pas sur le drone. L'analyse réalisé sur la faisabilité de contrôler un drone par la technologie de Xsens nous permet de comprendre qu'elle se base sur la communication PC $\leftrightarrow$ drone. Par conséquence, le drone doit être équipé et préparé pour recevoir des commandes via une communication WiFi.

L'équilibre sur Nao est quelque chose qui peut être encore optimisé, je suis satisfait des résultats mais pas entièrement. La complexité de l'équilibre dans le domaine de la robotique m'a surpris et tout le travail fait en quasi-statique ne va jamais être totalement satisfaisant. La recherche doit continuer vers le contrôle dynamique, basé sur *ZMP* ou un *torque-based control*. J'ai eu l'impression que beaucoup d'études, article, travails ont été faits à propos du contrôle dynamique d'un robot humanoïde mais en simulation, jamais avec les robots physiques. Ils entraînent beaucoup de contraintes: usage des moteurs, capteurs peu précis, etc...

L'analyse des différentes expérimentations réalisés est très importante et elle permet de comprendre l'importance de se poser la question de l'usage de toute technologie. Avant de tester la faisabilité, c'est importante de voir si **on en a besoin et pourquoi**.

D'un point de vu personnel, le travail en recherche sur l'innovation m'a apporté beaucoup de connaissances, j'ai compris la dureté de travailler sur la recherche et j'ai découvert un monde entier de possibilités.

D'autre part, et encore plus important, j'ai découvert aussi le travail social. En l'occurrence, j'ai eu la chance de participer dans le travail de Virginie ROUPENEL en tant que aidant mais aussi en tant qu'ingénieur. Son travail était de faire réaliser un test de personnalité à un groupe de participants anonymes pour une femme, un homme et un robot. Cependant, le robot (Pepper) était téléopéré par Virginie et par conséquence, moi je devait m'assurer que la téléopération se déroulait correctement et c'était aussi un moment pour tester fâce au public les innovation et la nouvelle logique appliqué. Le but de cette expérimentation était de commencer à comprendre (ou se poser la question) comment des personnes (qui ne savent pas que derrière le robot avait un personne) interagissent avec un robot avec des *traces d'intelligence* (je n'ose pas encore donner un qualificatif humain à un robot).

Le travail dans une association m'a aussi aidé à découvrir beaucoup d'initiatives, de

travailler avec des très bons professionnels et de connaître beaucoup de personnes qui travaillent pour le bien commun. **Travailler chez l'*Association Robots!* a été en définitive une excellente expérience**.

Appendices

Annexe A

Traitement des données Xsens

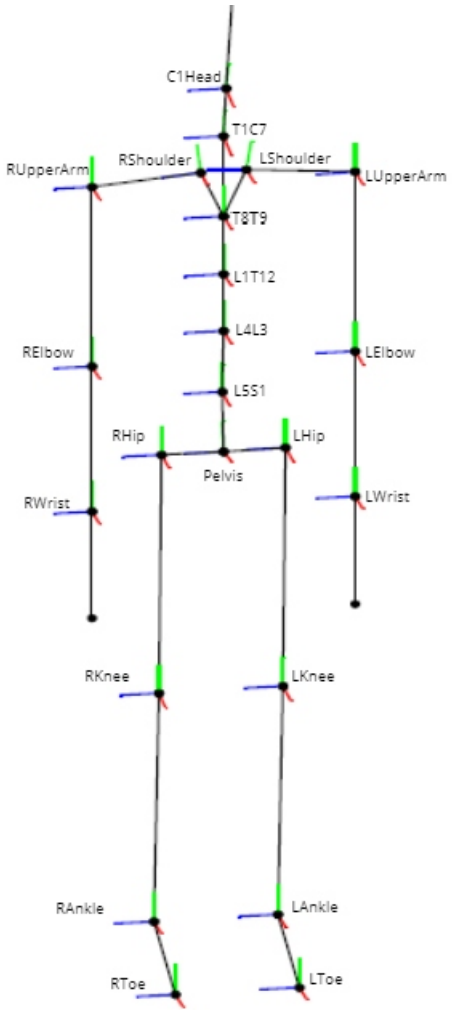


FIG. A.1: Nom et position des articulation du squelette modèle XSENS.

```
int bufferToInt(char buffer0, char buffer1, char buffer2, char buffer3)
{
    int a = int((unsigned char)(buffer0) << 24 |
                (unsigned char)(buffer1) << 16 |
                (unsigned char)(buffer2) << 8 |
                (unsigned char)(buffer3));
    return a;
}

float bufferToFloat(char buffer0, char buffer1, char buffer2, char buffer3)
{
    int myFourBytes = bufferToInt(buffer0, buffer1, buffer2, buffer3);
    float*  myFloat = (float*) &myFourBytes;
    return *myFloat;
}
```

FIG. A.2: Transformer les octets à des valeurs numériques.

```
void joint_buffer::define(char buffer[],int buffer_count)
{
    float xsens_rotation_x, xsens_rotation_y, xsens_rotation_z;

    id_parent = bufferToInt(buffer[buffer_count],buffer[buffer_count+1],buffer[buffer_count+2],buffer[buffer_count+3]);
    id_child = bufferToInt(buffer[buffer_count+4],buffer[buffer_count+5],buffer[buffer_count+6],buffer[buffer_count+7]);
    xsens_rotation_x = bufferToFloat(buffer[buffer_count+8],buffer[buffer_count+9],buffer[buffer_count+10],buffer[buffer_count+11]);
    xsens_rotation_y = bufferToFloat(buffer[buffer_count+12],buffer[buffer_count+13],buffer[buffer_count+14],buffer[buffer_count+15]);
    xsens_rotation_z = bufferToFloat(buffer[buffer_count+16],buffer[buffer_count+17],buffer[buffer_count+18],buffer[buffer_count+19]);

    rotation_x = xsens_rotation_x;  ///*************************************************************************///
    rotation_y = xsens_rotation_z;  ///************     XSENS JOINT FRAME WORK -- Y-UP right handed     ************///
    rotation_z = xsens_rotation_y;  ///*************************************************************************///

    if(buffer_count == 164 | buffer_count == 204) //Right Arm
    {
    else if(buffer_count == 184) //Right Elbow
    {

    else if(buffer_count == 244 | buffer_count == 284) //Left Arm
    {
    else if(buffer_count == 264) //Left Elbow
    {


    rotation = {rotation_x,
        rotation_y,
        rotation_z};

}
```

FIG. A.3: Stockage des angles du téléopérateur (*float*) et normalisation de la base.

Annexe B

Quadricoptère testé

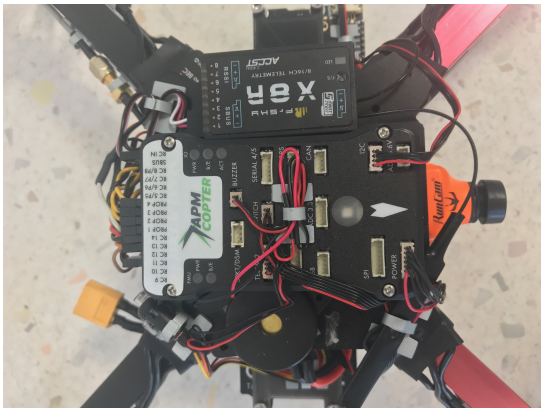


FIG. B.1: Contrôleur de vol *APM Copter*

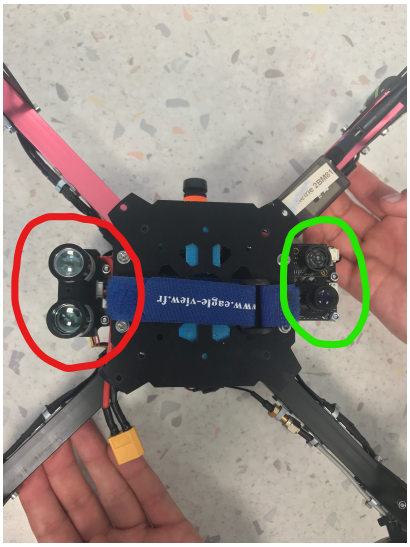


FIG. B.2: Cercle rouge: capteur de distance LIDAR-Lite V3; Cercle vert: PX4FLOW smart caméra.

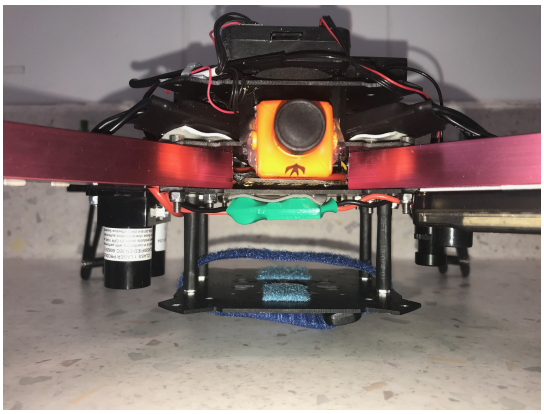


FIG. B.3: Caméra (vue de face).

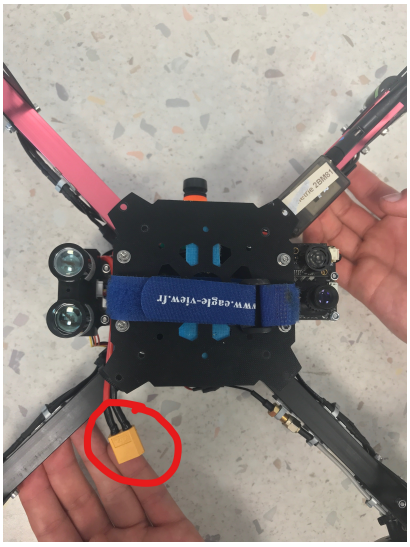


FIG. B.4: Connexion de la batterie du drone.

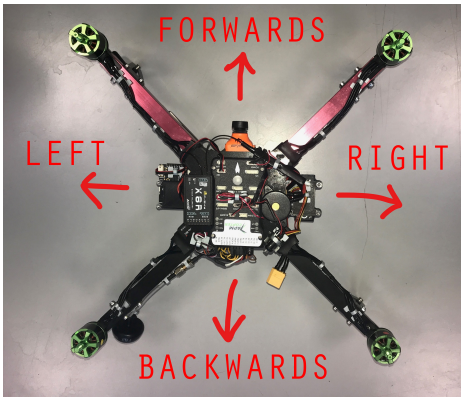


FIG. B.5: Les orientations du drone.

B.1 Solution proposée.

OPTION 3/ Construire un nouveau drone :

- BESOIN : Instructions précises avec liste des éléments nécessaires pour construire drone téléopérable :
Contrôleur de vol avec possibilité de connecter un module Wi-fi + Open Source

Panne de rechange 0,5mm	2,50 €	http://www.e44.com/outillage/soudage-dessoudage/accessoires/
Fer à souder céramique 30w	9,90 €	http://www.e44.com/outillage/soudage-dessoudage/fers-a-souder/fe
Support pour fer à souder	2,90 €	http://www.e44.com/outillage/soudage-dessoudage/supports-fer/sup
Solder wire – 1mm – 100g	6,50 €	http://www.e44.com/outillage/soudage-dessoudage/soudure/soudure
SUBTOTAL	**21,80 €**	
Ruban isolant électrique 15*10mm	1,00 €	http://www.e44.com/outillage/quincaillerie/produits-adhesifs/rubans-a
Pixracer + ESP8622 + cables	78,11 €	https://www.banggood.com/fr/Pixracer-R14-F4-Flight-Controller-Witl
Frame	8,15 €	https://www.banggood.com/Emax-TinyhawkS-Spare-Part-75mm-Pol
2 LiPo battery 2S 7,4V 1400mAh	18,12 €	https://www.banggood.com/Emax-TinyhawkS-Spare-Part-2S-7_4V-3
15500KV 1-2S Brushless Motor	36,24 €	https://www.banggood.com/Emax-TinyhawkS-Spare-Part-0802-155C
Power Distribution Board	3,62 €	https://www.banggood.com/Matek-Mini-Power-Hub-Power-Distributic
4 Propellers 40mm	2,71 €	https://www.banggood.com/2-Pairs-Emax-Tinyhawk-Indoor-FPV-Ra
Electronic Speed Controler 5A	14,85$	https://www.cesdeals.com/fr/product/4-in-1-5a-brushless-esc-progra
Camera	20,24 €	https://www.banggood.com/URUAV-UR65-FPV-Racing-Drone-Spare
SUBTOTAL	**167,19 €**	
TOTAL	**189,99 €**	

FIG. B.6: Devis proposé pour la solution numéro 3.

Bibliographie

Sources d'articles

[1] P-B. Wieber. A. Escande N. Mansard. "*Fast Resolution of Hierarchized Inverse Kinematics with Inequality Constraints*". In: (2010).

[2] P-B. Wieber. A. Escande N. Mansard. "*Hierarchical quadratic programming: Fast online humanoid-robot motion generations*". In: (2014).

[3] H. Herr A. Hofmann M. Popovic. "*Exploiting angular momentum to enhance bipedal center-of-mass control*". In: (2009).

[9] J.-J. Slotine B. Siciliano. "*A general framework for managing multiple tasks in highly redundant robotic systems*". In: (1991).

[10] G. Golub and C. Van Loan. "*Matrix computations*". In: (1996).

[12] A. Hertzmann M. De Lasa I. Mordatch. "*Feature-based locomotion controllers*". In: (2010).

[13] Joumana Medlej. "*Human Anatomy Fundamentals: Balance and Movement*". In: (2014).

[14] P.-B Wieber O. Kanoun F. Lamiraux. "*Kinematic control of redundant manipulators: generalizing the task priority framework to inequality tasks*". In: (2011).

[15] R. Boulic P. Baerlocher. "*An inverse kinematic architecture enforcing an arbitrary number of strict priority levels*". In: (2004).

[17] Louise Poubel. "*Whole-body online human motion imitation by a humanoid robot using task specification*". In: (2012).

[24] P. Kulvanit S. Amano M. Sasaki. "*A ZMP Feedback Control for Biped Balance and its Application to In-Place Lateral Stepping Motion*". In: (2008).

[25] D. Cehajic. S. Sakka L. Penna Poubel. "*Tasks prioritization for whole-body realtime imitation of human motion by humanoid robots*". In: (2014).

Autres sources

[4] P. Debraux A. Manolova. *Biomécanique du Sport et de l'Exercice. Chapitre 2: Description anatomique du mouvement.*
`https://www.sci-sport.com/theorie/biomecanique-du-sport-et-de-l-exercice.php`.
Accédé: 2019-05-10. Écrit: 2012-04-24.

[5] ArduPilot. *ArduPilot site de développement.*
`http://ardupilot.org/dev/index.html`.
Accédé: 2019-07-21.

[6] ArduPilot. *ArduPilot site web.*
`http://ardupilot.org/ardupilot/`.
Accédé: 2019-07-21.

[7] ArduPilot. *Installation de ArduPilot dans un drone.*
`http://ardupilot.org/copter/index.html`.
Accédé: 2019-07-21.

[8] ArduPilot. *Radio Calibration.*
`http://ardupilot.org/planner/docs/common-rc-transmitter-flight-mode-configuration.html`.
Accédé: 2019-07-21.

[11] GStreamer. *GStreamer web site.*
`https://gstreamer.freedesktop.org/`.
Accédé: 2019-07-01.

[16] Thierry Paillard. *Posture et équilibration humaines*. 2016.

[18] SoftBank Robotics. *Nao joints.*
`https://developer.softbankrobotics.com/nao6/nao-documentation/nao-developer-guide/kinematics-data/joints`.
Accédé: 2019-06-12.

[19] SoftBank Robotics. *Nao technical overview.*
`https://developer.softbankrobotics.com/nao6/nao-documentation/nao-developer-guide/technical-overview`.
Accédé: 2019-06-12.

[20] SoftBank Robotics. *NAOqi Developer guide.*
`https://developer.softbankrobotics.com/nao6/naoqi-developer-guide`.
Accédé: 2019-06-12.

[21] SoftBank Robotics. *NAOqi Developer guide installation.*
https://developer.softbankrobotics.com/nao6/naoqi-developer-guide/getting-started.
Accédé: 2019-06-12.

[22] SoftBank Robotics. *Pepper joints.*
https://developer.softbankrobotics.com/pepper-naoqi-25/pepper-documentation/pepper-developer-guide/kinematics-data/joints.
Accédé: 2019-06-12.

[23] SoftBank Robotics. *Pepper technical overview.*
https://developer.softbankrobotics.com/pepper-naoqi-25/pepper-documentation/pepper-developer-guide/technical-overview.
Accédé: 2019-06-12.

[26] *SoftBank description of NAO's chain joints.*
https://developer.softbankrobotics.com/nao6/nao-documentation/nao-developer-guide/kinematics-data/effector-chain-definitions.
Accédé: 2019-06-17.

[27] *SoftBank description of NAO's links.*
https://developer.softbankrobotics.com/nao6/nao-documentation/nao-developer-guide/kinematics-data/links.
Accédé: 2019-06-17.

[28] *SoftBank description of NAO's links and joints mass.*
http://doc.aldebaran.com/2-1/family/robots/masses_robot.html.
Accédé: 2019-06-17.

[29] XSENS. *MVN real-time network streaming - Protocol Specification.* Septembre 2018.

[30] XSENS. *User Guide Xsens MVN, MVN Link, MVN Awinda.* Octobre 2018.