Ecole des Mines de Douai, Option Productique

GUILBERT Matthieu

pour les Moyens Robotiques
Inria Rhône-Alpes

Mémoire de Projet de fin d'études

pour obtenir le titre

d'Ingénieur des Mines de Douai

Intégration logicielle en vue d'une expérimentation en réalité virtuelle autour de la marche humaine

INRIA Rhône-Alpes
Équipe des Moyens Robotiques
ZIRST - 655 avenue de l'Europe
38330 Montbonnot

Tuteurs de stage :
Laurence Boissieux
Pierre-Brice Wieber

Remerciements

Je tiens vivement à remercier le service des Moyens Robotiques de l'INRIA Rhône-Alpes pour m'avoir fait confiance et accepté au sein de leur service. Je tiens tout particulièrement à remercier **Laurence BOISSIEUX**, qui a su me guider, me conseiller tout au long de mon stage. Je tiens à remercier **Pierre-Brice WIEBER**, du projet BIP, pour m'avoir conseillé, mais aussi pour son encadrement scientifique, sa patience, et ses conseils tout au long de ce stage.

Enfin, je tiens à remercier, pour leur gentillesse et leurs nombreux conseils tout le personnel des Moyens Robotiques, en particulier Gérard BAILLE, Soraya ARIAS, Jean-François CUNIBERTO. Mais aussi les doctorants de l'INRIA, comme Christine AZEVEDO, ou Raphaël GRASSET et les stagiaires qui ont travaillé avec moi.

Table des matières

Table des figures

Introduction

0.1 Un challenge

De nos jours, l'animation par ordinateur se retrouve dans tous les domaines. Le besoin de visualiser des informations va maintenant au delà de la simple image en deux dimensions. Les tendances actuelles montrent un engouement certain pour la visualisation 3D mais aussi pour l'animation de ces images. Ces besoins d'animation/simulation se retrouvent dans des domaines aussi divers que :

- l'audiovisuel et, plus particulièrement, la création de films et d'effets spéciaux,
- les jeux vidéos qui cherchent à améliorer, à faible coût de calcul, la qualité des scènes animées,
- l'éducation par l'utilisation de simulateurs mettant les apprentis dans des situations complexes à produire,
- la médecine par l'apport d'outils de simulations d'actes chirurgicaux ou d'aide à la ré-éducation (aide à la marche des paraplégiques),
- la robotique, pour visualiser les mouvements d'un robot (bras manipulateur, robotique mobile, marcheuse) et vérifier la faisabilité d'une commande,
- le milieu sportif pour simuler le mouvement à la recherche de performances ou pour diagnostiquer des baisses de niveau.

Dans toutes ces applications, il est souvent nécessaire d'animer des personnages synthétiques (êtres humains, personnages imaginaires, robots humanoïdes, ...) en cherchant à reproduire de manière réaliste le mouvement humain. Ce réalisme peut prendre différents sens suivant le domaine d'application.

Intrinsèquement, le mouvement humain est très complexe puisqu'il résulte de la mise en oeuvre d'un grand nombre d'élements dont tous les maillons ne sont que partiellement connus.

Dans ce projet, nous proposons de capturer la démarche d'un être humain et de la reproduire sur un simulateur 3D du robot anthropomorphe BIP de l'INRIA Rhône-Alpes. Ce projet s'insère dans le cadre du projet BIP, et du service MRV2 (Moyens robotiques, Vision et réalité Virtuelle), et a comme objectif, à long terme, l'analyse de la marche des paraplégiques, et l'aide à la génération de commandes pour le robot BIP.

0.2 Les moyens robotiques, vision et réalité virtuelle

Les missions de ce service sont de trois types :

- Activité de service :
  - maintenance des systèmes robotiques,
  - installation et maintenance de logiciels spécialisés,
  - interface entre les utilisateurs et le service informatique,
  - assistance aux utilisateurs,
- Activité de développement :
  - mise en place d'expérimentations,
  - développement de logiciels dédiés à la robotique,
- Activité de recherche
  - conception de systèmes robotiques,
  - confrontation entre théories et expérimentations,

Le but du service robotique est de fédérer l'effort expérimental en favorisant :

- les expérimentations inter-projets,
- la mise en commun des moyens expérimentaux,
- les outils réutilisables (environnement de développement, machine de vision.

Ce service met à la disposition des projets (SHARP, MOVI, BIP, iMAGIS, PRIMA) de l'INRIA Rhône-Alpes un certain nombre de moyens techniques [1]

- dans la halle robotique :
  - un bras manipulateur,
  - 3 véhicules électriques Cycab,
  - un robot portique dédié à la vision,
  - un robot bipède,
- un espace aménagé pour des expérimentations en réalité virtuelle,
- des systèmes d'acquisition d'images.

0.3 Le projet BIP

Depuis 1994 et sous l'impulsion de Bernard Espiau, Directeur de Recherche à l'INRIA et à l'origine du projet BIP, l'INRIA Rhônes-Alpes et le Laboratoire de Mécanique

[1] Une description complète se trouve dans le site internet : http ://www.inrialpes.fr/iramr/.

des solides (LMS) de l'Université de Poitiers ont développé dans le cadre d'un projet commun, un robot anthropomorphe à quinze degrés de liberté. Le LMS a pris en charge la conception mécanique du prototype, le service des Moyens Robotiques a réalisé pour sa part, la partie électronique et informatique, enfin l'équipe de recherche BIP a eu la responsabilité de la partie contrôle-commande du système. Ce travail commun a permis d'aboutir à une plate-forme expérimentale robuste pour l'étude de la marche bipède robotisée.

L'objectif, ici, n'est pas de reproduire la morphologie des jambes humaines qui est une machine extraordinaire comprenant quelques 55 os, 90 muscles et 16 nerfs dont une réalisation mécanique précise peut largement dépasser la technologie courante. Le but est plutôt d'appliquer le comportement anthropormorphe à une version simplifiée des jambes.

Axes de recherche du projet :

- Modélisation de la marche humaine dans diverses configurations (en collaboration avec des biomécaniciens) : mesures de paramètres sur des ensembles de sujets, recherche d'invariants posturaux ou de mouvement, détermination des schémas de contrôle sous-jacents.

- Etude de méthodes de commande basées sur les régimes passifs quasi-périodiques, la stabilisation d'ensembles, les tâches référencées capteurs (force, proximité ou vision).

- Mise au point d'outils de conception, programmation et vérification pour l'ensemble du contrôle/commande à partir d'une approche synchrone, études de tolérance aux fautes.

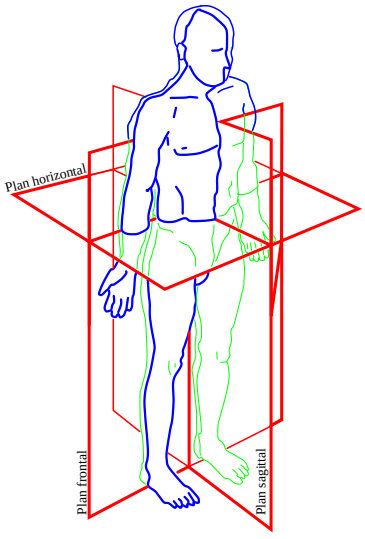


FIG. 1 – Les plans principaux servant à l'étude de la marche

0.4 Introduction à la marche humaine

Si on observe un mouvement de marche (décomposable en cycles) dans trois plans différents de l'espace (figure 1), le déplacement du sujet a lieu principalement dans le plan sagittal, mais de nombreux mouvements apparaissent dans chacun des autres plans : déhanchement, déplacement latéral et rotation du bassin qui améliorent considérablement la fluidité du mouvement.

La fluidité est très importante dans la marche puisqu'elle révèle une certaine forme d'économie de mouvements, caractéristique qui est attribuée assez généralement à tout mouvement humain. C'est pourquoi l'étude de la marche humaine permet une meilleure compréhension de cette fluidité.

0.5 L'animation basée sur des mouvements capturés

Les techniques de capture de mouvements ont été largement utilisées pour animer des squelettes 3D. En utilisant des technologies magnétiques ou optiques, il est possible de stocker les positions et les orientations de points situés sur le corps humain. Un calcul numérique fournit une correspondance entre le personnage synthétique et la personne, afin d'adapter les données à la nouvelle morphologie. Plusieurs approches [TMT96] [MFD] ont introduit des techniques pour adapter la capture de trajectoires à un personnage différent. De tels systèmes traitent aussi les erreurs de capture en approximant dans leurs calculs numériques les erreurs de calibration, le bruit électronique, etc. La méthode consiste à retrouver la trajectoire angulaire de chaque articulation durant le mouvement et à l'appliquer au personnage synthétique. A partir de la position et de l'orientation des différents marqueurs, un algorithme d'optimisation calcule la cinématique inverse pour produire la trajectoire des articulations. Le personnage synthétique exécute exactement le même mouvement que l'acteur réel. Les outils de capture de mouvements [SM] les plus utilisés sont de deux types, les systèmes à marqueurs actifs, et ceux à marqueurs passifs. Les premiers utilisent des marqueurs qui émettent un signal (électro-magnétique ou lumineux) et les seconds utilisent des marqueurs qui réflechissent un signal (la plus part du temps lumineux). Les systèmes à marqueurs passifs sont munis de stromboscopes qui émettent une lumière (infra-rouge) et les marqueurs la réfléchissent. Des caméras sensibles à cette lumière enregistrent ces signaux, il faut alors indexer chaque marqueur et trouver ses coordonnées. Ce système est difficilement utilisable en temps réel, c'est pourquoi nous n'avons pas retenu cette solution. Nous devons utiliser un système à marqueurs actifs, nous pouvons soit utiliser des marqueurs électromagnétiques, mais ils sont très sensibles à l'environnement (les matériaux métalliques influent énormément sur les données, elles deviennent peu fiables), soit des marqueurs infra-rouge, dont lémission est captée par des caméras CCD. Ce principe est celui de l'OPTOTRAK (Northern Digital), nous utiliserons ce

matériel qui est le plus adapté à notre cas.

Dans notre projet, le principe utilisé sera celui cité ci-dessus. Ainsi, il nous faut capturer les mouvements de la personne à l'aide de l'OPTOTRAK, puis retrouver les coordonnées articulaires et les paramètres intrinsèques de la personne à l'aide d'un algorithme d'optimisation (une simple inversion cinématique serait plus coûteuse en marqueurs, il en faudrait 3 par solides. De plus, l'optimisation [MFD] est une méthode plus robuste tant numériquement que méthodologiquement) et enfin reproduire le mouvement sur le simulateur du robot BIP.

0.6 Déroulement du projet

Plusieurs points sont abordés dans la capture de mouvements utilisée pour l'animation d'un personnage. Il faut tout d'abord créer le personnage et pouvoir l'animer dans un simulateur, il faut ensuite traiter les données capturées. Le projet se décompose naturellement en trois parties :

- adaptation du simulateur 3D de BIP et la capture de mouvements,
- aspect numérique de l'inversion cinématique,
- intégration logicielle et matérielle

Pour aborder tous ces domaines, il m'a fallu me familiariser avec openGL et Open Inventor (surcouche d'openGL), avec les algorithmes d'optimisation, avec l'optotrak et plus généralement avec la capture de mouvements. C'est pourquoi chaque partie de ce projet a commencé par une phase de lecture et d'apprentissage de ces différents domaines.

Première partie

Le simulateur 3D et la capture de mouvement

Chapitre 1

Modèle géométrique du robot

1.1 Présentation générale [Wie00]

L'objectif initial du projet BIP est la réalisation d'un robot bipède à 15 degrés de liberté (ddl), aux dimensions et masses des membres inférieurs voisines de celles de l'homme [Sar00]. Ce robot a été conçu pour reproduire une marche dynamique 3D stable sur sol plat et assurer la montée et la descente d'escaliers.

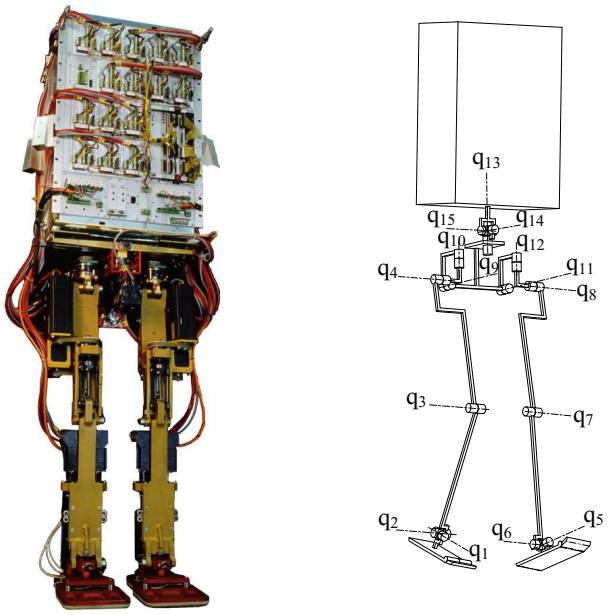


FIG. 1.1 – Le robot bipède BIP et ses degrés de liberté

Dans sa première version (juillet 1999) le système ne comportait que deux jambes (8 ddl) associées à un chariot à quatre roues (6 ddl plan). Le prototype suivant (janvier 2000) vit l'apparition d'un tronc intégrant l'électronique de commande. Cette dernière version totalise ainsi 15 ddl dont une rotule complète (3 ddl) au niveau du tronc (cf figure 1.1). Il mesure 180 cm pour 105 kg.

1.2 Modélisation du système, formalisation de Khalil-Kleinfinger [Wie00]

Pour étudier les mouvements d'un robot constitué d'un ensemble de solides, il est indispensable de pouvoir décrire la position dans laquelle il se trouve à chaque instant. La position et l'orientation de chacun de ses éléments définissent la posture du robot, à laquelle il faut ajouter sa position et son orientation dans l'espace de travail. La posture décrit ainsi la disposition des solides les uns par rapport aux autres, tandis que la position et l'orientation permettent de les situer correctement par rapport à l'environnement.

Les articulations du robot BIP ne permettent que des rotations autour de certains axes (figure 1.1). Ces articulations conditionnent donc la disposition des solides les uns par rapport aux autres, et il suffit de répertorier les positions dans lesquelles elles se trouvent pour pouvoir décrire complètement la posture du robot.

Une fois la posture du robot déterminée, il suffit de connaître la position et l'orientation de l'un de ses solides pour pouvoir en déduire la position et l'orientation du robot au complet. On peut alors réunir l'ensemble des positions articulaires ainsi que la position et l'orientation de ce solide pour constituer un vecteur $q \in \mathbb{R}^n$ qui décrit entièrement la position du robot.

Le robot BIP compte actuellement 15 degrés de liberté. Sa posture est donc déterminée par un vecteur $\vec{q}$ de dimension 15, auquel il faut ajouter 6 variables supplémentaires pour exprimer la position et l'orientation du robot dans l'espace. On obtient un système totalisant 21 degrés de liberté et pour lequel un vecteur $\vec{q}$ de dimension 21 décrit totalement sa position.

La formalisation dite de Khalil-Kleinfinger [KK86] est une modification de la représentation de Denavit-Hartenberg afin de faciliter son application aux chaînes cinématiques fermées. Ce paramétrage décrit le passage du repère $i-1$ au repère i avec les conventions de la figure 1.2.

A chaque solide i, on associe un repère (O_i, X_i, Y_i, Z_i). La représentation de Khalil-Kleinfinger classique pour une chaîne de n segments successifs est donnée par une liste de vecteurs de la forme $(r_i, \lambda_i, \alpha_i, q_i)_{i=1..n}$. Pour définir la position et l'orientation du solide i par rapport au solide $i-1$, ces paramètres sont définis de la manière suivante :

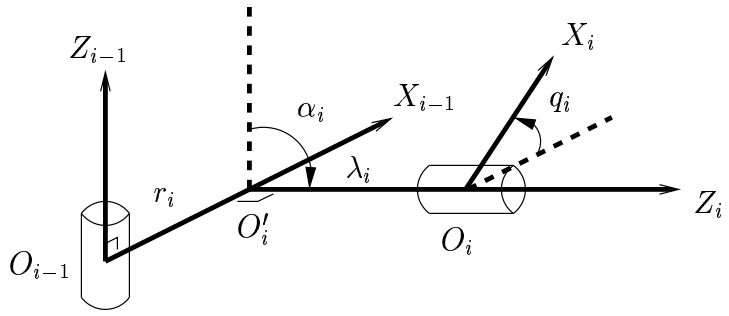


FIG. 1.2 – Description du paramétrage de Khalil-Kleinfinger

$$\begin{cases} r_i = \overrightarrow{O_{i-1}O'_i} \cdot \overrightarrow{X_{i-1}} \\ \lambda_i = \overrightarrow{O'_iO_i} \cdot \overrightarrow{Z_i} \\ \alpha_i = (\overrightarrow{Z_{i-1}, Z_i}) \cdot \overrightarrow{X_{i-1}} \\ q_i = (\overrightarrow{X_{i-1}, X_i}) \cdot \overrightarrow{Z_i} \end{cases} \tag{1.1}$$

Ainsi, la matrice qui permet de passer d'un repère à l'autre prend la forme (en coordonnées homogènes) :

$$\begin{bmatrix} \cos q_i & -\sin q_i & 0 & r_i \\ \cos\alpha_i \sin q_i & \cos\alpha_i \cos q_i & -\sin\alpha_i & -\lambda_i \sin\alpha_i \\ \sin\alpha_i \sin q_i & \sin\alpha_i \cos q_i & \cos\alpha_i & \lambda_i \cos\alpha_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Le détail de la modélisation géométrique du bipède est disponible dans le rapport technique de P. Sardain [Sar00].

1.3 Les différents modèles de BIP [Sar00]

BIP a été décrit par deux modèles géométriques distincts, le premier décrit les positions du robot et de chaque solide avec un référentiel initial lié au pied droit. Un second paramètrage à partir du pelvis, considéré comme plateforme mobile, a été établi afin d'équilibrer, d'alléger la description géométrique de BIP.

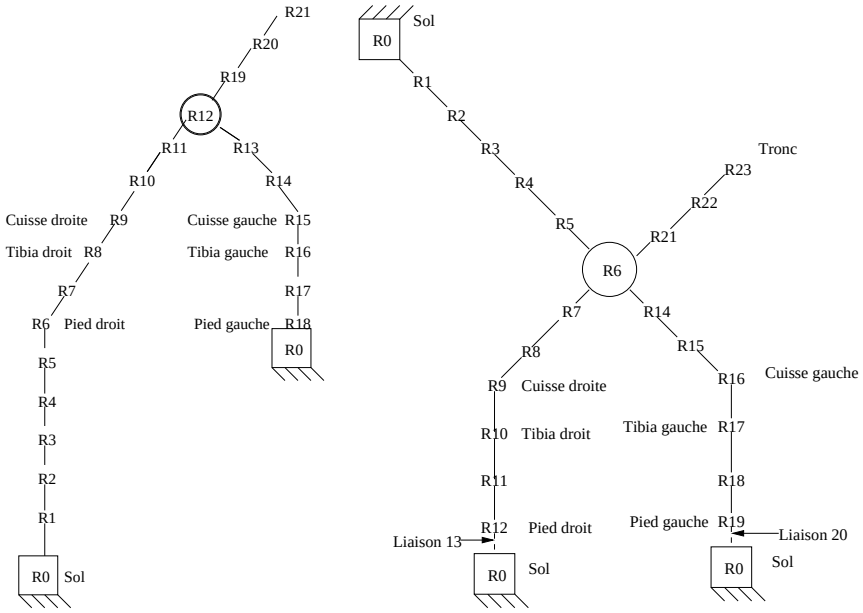


FIG. 1.3 – Ancienne et nouvelle description des repères du paramètrage de Khalil-Kleinfinger

On distingue :

	q	**type**
Les 6 ddl du pelvis	q_{16} à q_{18} q_{19} à q_{21}	glissières pivots
les 15 axes motorisés		
Jambe droite	$q_1..q_4$, q_9 et q_{10}	pivots
Jambe gauche	$q_5..q_8$, q_{11} et q_{12}	pivots
Tronc	$q_{13}..q_{15}$	pivots

Prenons quelques solides caractéristiques comme les pieds droit et gauche, le pelvis et le tronc :

Solides caractéristiques	**Nombre de changements de repère**	
	modèle 1	**modèle 2**
Pied droit	3	9
Pied gauche	15	9
Pelvis	9	3
Tronc	12	6

Ce tableau comparatif montre que le deuxième modèle est plus adapté au robot BIP que le premier. En effet, le premier nécessite plus de changements de repère que le second, c'est pourquoi un changement de modèle sur le visualisateur openInventor est indispensable

Chapitre 2

Modèle Open Inventor

2.1 Présentation d'Open Inventor

Open Inventor [Wer94] est une librairie d'objets et de méthodes pour créer des applications 3D interactives. C'est un ensemble de blocs qui permet à l'utilisateur de concevoir des applications 3D de manière simple et rapide. Basé sur OpenGL, il offre l'efficacité d'un système totalement orienté objet.

Son principe de fonctionnement est basé sur les *nodes*.Une node constitue la brique de base pour créer des scènes. Chaque *node* apporte une information nécessaire à la scène comme le type de surface d'une pièce, une transformation géométrique...

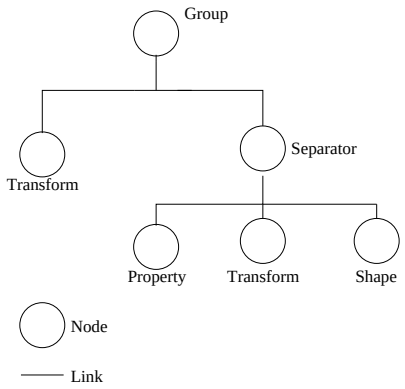


FIG. 2.1 – Exemple de *Scene Graph*

Une collection (ordonnée) de *nodes* est regroupée en *Scene Graph* (cf figure 2.1), qui est elle-même stockée dans la base de données Inventor (Cette dernière peut en contenir plusieurs). L'ordre des *nodes* est très important étant donné que le graphe est lu de gauche à droite et que chaque node hérite des transformations des nodes en amont et influe sur celles en aval (le *node SoSeparator* permet de stopper cet héritage).

2.2 Modèle du robot BIP

Un modèle openInventor a été développé par l'INRIA (Pierre-Brice WIEBER du projet BIP), il est basé sur le premier modèle géométrique du robot. Par souci d'homogénéité, nous avons modifié le code C pour le transformer en C++ (création d'une classe modeleBIP).

Pour "assembler" le robot, il faut positionner un repère sur tous les solides de BIP (nous avons 15 pièces, toutes stockées dans des fichiers .iv comme dans la figure 2.2). Puis, grâce à la formulation de Khalil-Kleinfinger, nous positionnons tous les repères (donc les différents solides) du modèle les uns par rapport aux autres.

Les repères du nouveau modèle sont différents de l'ancien, il nous a donc fallu repositionner les repères sur les 15 pièces.

FIG. 2.2 – Exemple de fichier .iv importé pour construire le robot en open Inventor

Ainsi, le modèle Inventor comporte 21 repères et donc :

Nb	**Transformations**	**Param. K-K**
21	Translations / X	$r_{1..21}$
21	Rotations / X	$\alpha_{1..21}$
21	Translations / Z	$\lambda_{1..21}$
21	**Articulations** rotations /Z	$q_{1..21}$

2.3 Le simulateur 3D

Pour mettre en mouvement le robot, il suffit de modifier les valeurs des différentes articulations (les $q_{1..21}$). Nous obtenons alors le résultat de la figure 2.3.

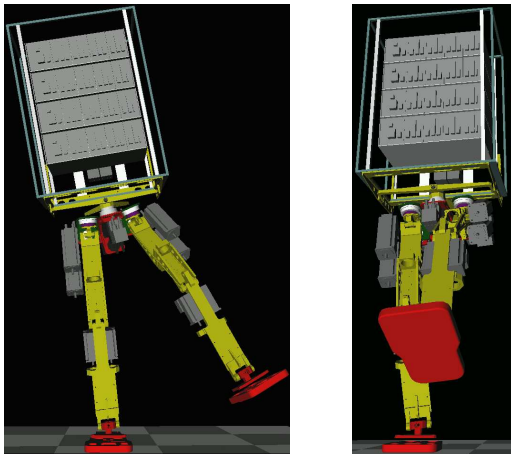


FIG. 2.3 – Simulateur 3D du robot BIP : modèle Open Inventor

Afin de visualiser BIP, nous avons décidé de ne pas créer notre propre *Viewer* mais d'utiliser celui développé par Gilles Debunne du projet iMAGIS : QGLViewer. C'est une librairie *Open Source* qui permet de commencer rapidement le développement d'application 3D. Il a été conçu pour des développeurs C++ 3D, il peut facilement afficher et mouvoir des scènes 3D en fournissant simplement les instructions OpenGL qui définissent leur géométrie 3D. Le *Viewer* a été conçu pour être aussi général que possible, et non pas pour une application 3D spécifique. Il est destiné à être un point de départ pour tout programme 3D graphique.

Ce *Viewer* utilise Qt (Qt est une plateforme croisée C++ et GUI). Elle fournit toutes les fonctionalités dont ont besoin les développeurs d'applications pour concevoir des interfaces utilisateurs graphiques. Qt est totalement orienté objet, c'est un composant standard de la majorité des distributions des systèmes UNIX.

2.4 Implémentation du simulateur

D'un point de vue programmation, nous avons écrit 2 classes :

- Robot (Description en *Annexe B*) : Construction du modèle Inventor du robot.
- BIPViewer (Description en *Annexe C*) : Sous Classe de QGLViewer permet tant d'afficher que de mouvoir le robot.

Chapitre 3

Description et Principe de fonctionnement de l'OPTOTRAK

3.1 Présentation matérielle générale

3.1.1 Principe de fonctionnement

L'OPTOTRAK [Dig] consiste en trois caméras CCD [1] couplées avec 3 objectifs, montés sur une barre stabilisatrice de 1.1m. A l'intérieur des trois objectifs, la lumière d'une source infra-rouge (un marqueur OPTOTRAK) est dirigée vers une caméra et mesurée. Les trois mesures se complètent pour fournir en temps réel la position 3D d'un marqueur (diodes Infra-Rouge) situé sur la personne qu'il faut suivre.

FIG. 3.1 – Les 3 objectifs CCD de l'OPTOTRAK

Le système OPTOTRAK, construit par Northern Digital, mesure des mouvements et des positions 3D en suivant la course de marqueurs solidaires d'un sujet. Lorsque

[1] Charge Coupled Device. dispositif par transfert de charges. C'est une matrice composée de plusieurs pixels(photosites). Ces pixels sont sensibles à la lumière, ils gardent en mémoire par accumulation sous la forme de charges électriques proportionnelles à la quantité de lumière reçue.

le sujet bouge, l'OPTOTRAK détecte les positions des marqueurs et calcule simultanément de manière précise les positions 3D de chaque marqueur.

L'OPTOTRAK active séquentiellement chaque diode IR (toutes branchées sur des strobers [2]). Ainsi le système peut automatiquement identifier chacune d'entre elles pendant la prise de données : les marqueurs sont donc indexés.

Ce système de capture est un instrument très flexible, il est utilisé pour de nombreuses applications comme les études sur le mouvement humain, la robotique, la chirurgie, et l'aéronautique.

3.1.2 Quelques spécifications techniques

Quelques propriétés de l'OPTOTRAK sont trés intéressantes pour notre problème :

- flux de données rafraîchi à 600 Hz

- précision de 0.01 mm à 2.50 m sur les trois axes (l'OPTOTRAK peut ainsi distinguer deux marqueurs si leur distance l'un de l'autre est plus grande que 0.01 mm),

- exactitude à 2.25 m de 0.1 mm pour les x et y, 0.15 mm pour les z (l'erreur absolue de position d'un marqueur).

Le champ de vision de l'OPTOTRAK est cependant assez restreint(*Annexe D*), et ne permet pas de faire de grands mouvements ou de parcourir de grandes distances en marchant. Par contre, il est possible d'en coupler plusieurs, ce qui permet de faire une meilleure analyse de la démarche humaine, car le volume d'observation est plus grand et le risque d'occultations des marqueurs plus faible.

Les positions des marqueurs sont relatives à un repère, deux possibilités s'offrent à nous :

- le repère absolu lié à l'OPTOTRAK,

- utiliser un *rigid body*.

Un *rigid body* est composé d'au moins 3 leds, elles permettent de créer un repère que l'on positionne comme on veut par rapport au sujet. Ainsi les positions des marqueurs sont données par rapport à ce nouveau repère.

[2] collecteurs de Leds

FIG. 3.2 – RigidBody

3.2 Partie logicielle de la capture de mouvements

3.2.1 Utilisation de librairies C/C++ prédeveloppées

Afin de pouvoir utiliser l'OPTOTRAK de façon correcte et rapide, j'ai utilisé une librairie développée par le service des moyens robotiques de l'INRIA (Laurence BOISSIEUX) qui est une surcouche objet de la librairie C fournie par le constructeur de l'OPTOTRAK Northern Digital.

Cette librairie permet (description en *Annexe E*) :
- d'utiliser les capteurs IR de l'OPTOTRAK
- d'initialiser le système
- de récupérer les positions 3D en temps-réel de manière bloquante ou non bloquante
- d'arrêter le système

3.2.2 Sur-couche logicielle pour adapter la librairie à notre projet

Afin d'adapter la librairie de l'OPTOTRAK à la capture de mouvement, j'ai développé une surcouche logicielle qui permet :
- d'initialiser l'OPTOTAK
- d'avoir un tableau qui donne les positions de tous les marqueurs
- d'avoir un tableau avec tous les marqueurs occultés

- d'arrêter le système.

Deuxième partie

Calcul de la posture et de la position de la personne dans l'espace

Chapitre 1

Présentation du problème d'inversion cinématique

1.1 L'inversion cinématique

Les techniques [MFD] de capture de mouvements consistent à capturer les positions de certains points du sujet, le but est donc de retrouver sa posture, sa position et son orientation dans l'espace.

1.1.1 Inconnues du problème

Dans le paragraphe 1.2 de la partie précédente, nous avons défini le vecteur q qui représente la posture, la position et l'orientation de BIP dans l'espace. Nous définissons pour le sujet le même vecteur q tel que :

- les $q_{1..15}$ soient les coordonnées articulaire du sujet,
- les $q_{16..21}$ soient la position et l'orientation du sujet dans l'espace.

En définissant ce vecteur, qui décrit totalement le sujet dans l'espace, nous devons élaborer un modèle du sujet. Nous utilisons une formalisation de Khalil-Kleinfinger identique à celle du robot mis à part quelques paramètres qui sont intrinsèques à la morphologie du sujet :

- largeur de la hanche,
- hauteur du fémur,
- hauteur du tibia,
- hauteur sol-cheville.

En considérant que le sujet est symétrique, nous obtenons 4 paramètres. Pour décrire totalement le sujet, nous devons les ajouter au vecteur q, c'est pourquoi par la suite, q est de dimension 25 (q défini ci-dessus concaténé avec les 4 paramètres).

1.1.2 Description de l'inversion cinématique

Nous avons n points (appelés tags) disposés sur le sujet, dont nous pouvons connaître les positions, dans un repère lié à l'environnement, grâce à notre système de capture de mouvements. Ces données sont stockées dans un vecteur T^{obs} :

$T^{obs} = (x_1^{obs}, y_1^{obs}, z_1^{obs}, ..., x_k^{obs}, y_k^{obs}, z_k^{obs}, ..., x_n^{obs}, y_n^{obs}, z_n^{obs})^T$

Plus généralement nous définissons T la position des tags sur le sujet :

$T(q) = (x_1(q), y_1(q), z_1(q), ..., x_k(q), y_k(q), z_k(q), ..., x_n(q), y_n(q), z_n(q))^T$

Le modèle direct (qui est connu) nous permet de trouver la position de tous les tags à partir de q. Ainsi il nous faut trouver q tel que $T(q) = T^{obs}$. Notre problème se situe dans cette inversion de T : $q = T^{-1}(T^{obs})$.

1.2 Positionnement des tags sur le sujet

19 points caractéristiques du robot ont été définis dans [Wie00]. Nous prenons des points équivalents pour nos tags (cf figure 1.1). Chaque tag nous fournit 3 informations (ses coordonnées dans l'espace), nous devons trouver q, c'est à dire 25 inconnues. Tous les tags nous fournissent 57 informations, nous avons plus d'informations que d'inconnues, certaines données sont redondantes.

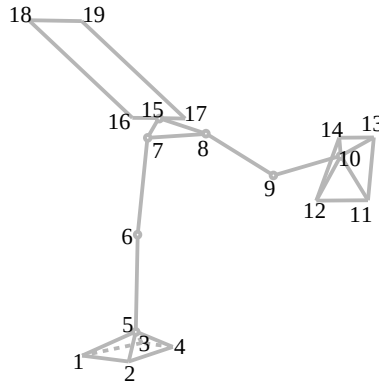


FIG. 1.1 – Quelques points caractéristiques

Intuitivement, les tags 3,4,15,17,13,14 sont inutiles, c'est vérifiable numériquement (chapitre 2). Connaissant les tags à disposer sur notre sujet, il suffit maintenant de retrouver q.

1.3 Le problème d'optimisation

Un problème d'optimisation [Stu] peut se formuler de manière générale :

Trouver $x \in \mathbb{R}^n$ qui minimise la fonction $f : \mathbb{R}^n \longmapsto \mathbb{R}$
sous les contraintes $c_i(x) = 0$
$c_i(x) \geq 0$

La philosophie de notre inversion cinématique est de faire varier q afin de minimiser la distance qui sépare la position des tags calculés en fonction de q et la position des tags observés. Il faut donc minimiser la distance pour chaque tag k :

$$f(q) = \frac{1}{2} \sum_k (x_k(q) - x_k^{obs})^2 + (y_k(q) - y_k^{obs})^2 + (z_k(q) - z_k^{obs})^2$$

$$\boxed{f(q) = \frac{1}{2} \|T(q) - T^{obs}\|^2}$$

Le but est de trouver un q qui minimise cette fonction. Pour ce type de problème, les algorithmes les plus efficaces sont ceux de type Newton [PEFT]. Ils se basent sur un développement de Taylor du premier ou second ordre, et permettent de converger vers une solution en un pas pour les fonctions quadratiques (cf Figure 1.2).

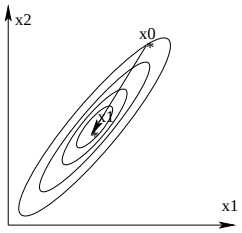


FIG. 1.2 – La méthode de Newton trouve le minimum d'une fonction quadratique en un pas

Soit f la fonction à minimiser, on a :

$$f(q + \delta q) = f(q) + \frac{\partial f}{\partial q}(q)\delta q + \frac{1}{2}\delta q^T \frac{\partial^2 f}{\partial q^2}\delta q + o(\|\delta q\|^2)$$

Si $\dfrac{\partial^2 f}{\partial q^2}$ est définie positive, alors f a un unique minimum en un point où le gradient est nul, c'est à dire lorsque :

$$\boxed{\frac{\partial f}{\partial q}(q) + \frac{\partial^2 f}{\partial q^2}(q)\delta q = 0}$$

Ce principe est utilisé dans beaucoup d'algorithmes d'optimisation, entre autre dans *Optim* et *FSQP* qui sont utilisés dans nos premières simulations.

Chapitre 2

Simulation numérique

Afin de simuler numériquement la reconstruction de la posture de la personne traquée, nous avons utilisé des positions de base fournies par le simulateur du robot développé par les membres du projet BIP. De même, nous avons utilisé la puissance de calcul formel du logiciel *MAPLE*, et le logiciel *SCILAB* pour le calcul numérique et sa capacité à générer une animation 3D rapidement.

2.1 Les algorithmes d'optimisation

Dans un premier temps, nous avons utilisé des librairies d'optimisation existantes :

- Fonction *Optim* de SCILAB : routine d'optimisation non linéaire. Nous utilisons un algorithme de Quasi-Newton (même principe que la méthode Newton mais la hessienne de f est approximée).

- Librairie *CFSQP* [CLT] : CFSQP est un ensemble de routines C pour la minimisation de fonctions continues soumises à des contraintes non-linéaires d'égalité ou d'inégalité.

Pour minimiser une fonction, ces algorithmes demandent les bornes supérieures et inférieures pour q et le gradient de la fonction. Il faut donc être capable de calculer $T(q)$ et donner le gradient de notre fonction à minimiser.

Avec $f(q) = \frac{1}{2}\sum_k (T_k(q) - T_k^{obs})^2$

On a aussi :

$$\frac{\partial f}{\partial q} = \sum_k \frac{\partial T_k(q)}{\partial q}(T_k(q) - T_k^{obs})$$

En définitive, on obtient :

$$\boxed{\frac{\partial f}{\partial q} = \left(\frac{\partial T(q)}{\partial q}\right)^T (T(q) - T^{obs})}$$

Le gradient de la fonction à minimiser est donc fonction de la jacobienne de T.

2.2 Mise en place du programme de simulation numérique

Dans un premier temps, on calcule T et sa jacobienne. Pour cela nous avons utilisé *MAPLE*. Ensuite nous générons des fonctions C, qui renvoient les vecteurs ou les matrices correspondantes.

Trois fichiers .maple ont été créés :

- TrackCine.maple : fichier déterminant la position du robot BIP en fonction de $\vec{q}$, par l'intermédiaire des paramètres KK (*Annexe H*)
- complement.maple : fichier contenant le référentiel de positionnement de chaque tag
- TrackRobot.maple : fichier contenant toutes les fonctions pour calculer la jacobienne et le vecteur des tags.

Les fonctions de TrackRobot.maple sont les suivantes (Les deux premières ont été préalablement développées au sein du simulateur du robot BIP) :

Fonctions	**Description**
matrice_passage(k)	calcule la matrice de passage du repère ref_k (*Annexe H*) au repère k
matrice_repere(k)	calcule la matrice de passage du repère du pelvis au repère k
coord_tag(tagVect,k)	calcule les coordonnées du tag k en fonction de sa position relative tagVect dans son repère de référence
matrice_tags()	réunit toutes les coordonnées des tags dans un vecteur sous la forme : (x_1 y_1 z_1 ... x_n y_n z_n ... x_{19} y_{19} z_{19})
jacobienne_tags()	calcule la jacobienne du vecteur tags en fonction de $q_{1..25}$
ecriture_modele()	génération du code C pour le vecteur tags et sa jacobienne

Dans un second temps, on lance la simulation numérique sous *SCILAB*.

Trois fichiers *SCILAB* ont été créés :

- invTags.sci : fichier définissant les algorithmes d'inversion cinématiques
- TrackFunctions.sci : fichier contenant les fonctions relatives à l'étude numérique
- Sim.sce : fichier contenant le programme principal d'éxécution.

Les fonctions de *invTags.sci* sont les suivantes :

Fonctions	**Description**
distance(j,q)	fonction à minimiser de notre problème d'optimisation
gradient_distance(j,q)	calcule le gradient de notre fonction à minimiser
invTags(tagsObserves,BT)	calcule l'inversion cinématique avec Optim ou FSQP tagsObserves : vecteur des valeurs des tags capturés BT : tags occultés

Les fonctions de *TrackFunctions.sci* sont les suivantes :

Fonctions	**Description**
BadTag(Complete_Matrix,BT)	enlève les lignes de *Complete_Matrix* fournies par le vecteur BT
compareTags(tags_found,tags_des) (cf Figure 2.1)	affiche en 3D filaire la reconstruction du robot superposé avec la position capturée

Les fonctions présentées ci-dessus sont les plus importantes, d'autre fonctions, plus spécifiques, ont été développées pour l'analyse numérique, comme l'influence des occultations de tag sur la stabilité de l'algorithme.

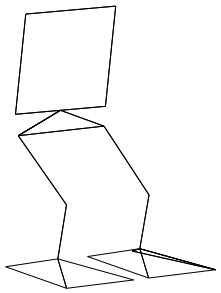


FIG. 2.1 – Résultat de la fonction compareTags

Chapitre 3

Résultats Numériques

Plusieurs tests ont été effectués :

- Stabilité des algorithmes avec occultation de certains tags
- Influence du bruit sur la reconstruction

La plupart des tests ont été effectués sur 20 postures différentes à reconstituer.

3.1 Premiers Résultats

Deux études ont été faites sur l'influence des tags :
- L'influence des tags pris individuellement
- L'influence des doublets.

3.1.1 Influence de chaque tag pris individuellement

Dans un premier temps, afin d'étudier quels sont les tags indispensables pour retrouver le meilleur q décrivant la personne, nous avons étudié l'influence de chaque tag sur toutes les coordonnées de q. En *annexe H*, on trouve les résultats pour les 6 premiers tags.

En observant tous ces résultats, on remarque que les erreurs sont beaucoup plus grandes pour certains tags.

On peut donc en conclure que certains tags sont indispensables à la bonne reconstruction :

- Tags 5 et 10 : les chevilles
- Tags 6 et 9 : les genoux
- Tags 7 et 8 : les hanches

Au final, et après étude des courbes, nous gardons les tags (cf figure 1.1) : 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 16, 18, 19.

Après cette constatation, nous avons créé de nouvelles fonctions sous Maple, qui n'utilisent que les tags cités ci-dessus.

3.1.2 Influence des doublets

Une première étude a été effectuée avec Optim, mais cette fonction faisait apparaître des erreurs de reconstruction incompréhensibles sur certaines positions, c'est pourquoi, nous avons conclu a un *bug* de cette dernière. Pour la suite de notre étude nous avons utilisé FSQP [CLT] comme algorithme d'optimisation. Cette étude a été effectuée sur 20 positions, sur chaque graphique apparaissent le minimum, la moyenne et le maximum de l'erreur causée par l'occultation de deux tags (cf *Annexe I*). En observant ces graphiques, nous obtenons des erreurs anormales sur certains doublets, on a remarqué que dans ces cas particuliers, les paramètres $q_{18..21}$ allaient en butées. Ces paramètres (ils représentent l'orientation de la personne dans l'espace) sont des fonctions de période 2π. Pour les $q_{18..21}$, il y a donc une infinité de solutions identiques (de minimums). Même si théoriquement il n'y a pas de butées pour ces valeurs de q, nous devons en donner à FSQP (nous donnons ± 1000). Pour des raisons qui nous sont inconnues, il arrive que FSQP diverge pour ces valeurs, dans ce cas là, il y a deux explications possibles (notre explication étant une simplification de la réalité mais représentative du problème) :

- ce point est un minimum local, FSQP renvoie ce minimum qui n'en est pas un (premier schéma de la figure 3.1)

- ce point est un maximum local, FSQP renvoie le minimum le plus proche, ce qui est correct dans notre cas (second schéma de la figure 3.1)

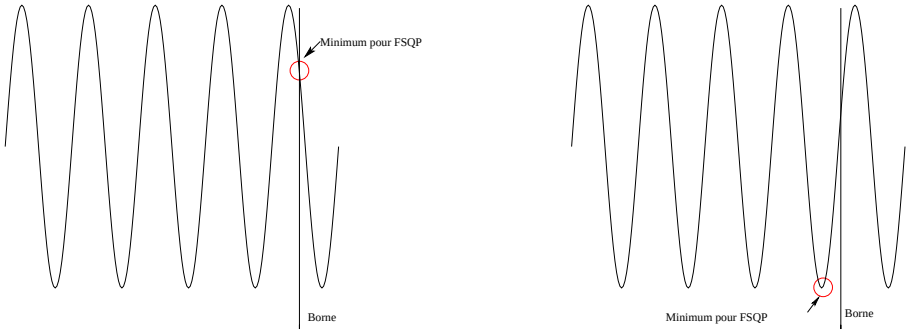


FIG. 3.1 – les deux cas de convergence de FSQP

A cause de ce problème, l'algorithme ne peut pas être utilisé, il nous faut alors développer notre propre algorithme.

3.2 La méthode de Newton

Avec les algorithmes précédents, l'hypothèse que chaque posture est très proche de la précédente (Notre fréquence de capture étant de 50 Hz) n'est pas prise en compte. Il nous faut développer un algorithme qui prenne en compte cette hypothèse.

Nous reprenons la fonction à minimiser définie dans le paragraphe 1.3 de cette partie, et donc son gradient calculé dans le paragraphe 2.1.

Ainsi nous avons :

$$f(q) = \tfrac{1}{2}\|T(q) - T^{obs}\|^2$$

$$\frac{\partial f}{\partial q} = \frac{\partial T}{\partial q}^T (T(q) - T^{obs})$$

Pour trouver un minimum de f, il faut que (cf paragraphe 1.3) :

$$\frac{\partial f}{\partial q}(q) + \frac{\partial^2 f}{\partial q^2}(q)\delta q = 0 \Leftrightarrow \delta q = -\left(\frac{\partial^2 f}{\partial q^2}(q)\right)^{-1} \frac{\partial f}{\partial q}(q)$$

Il faut donc calculer la hessienne $\left(\frac{\partial^2 f}{\partial q^2}(q)\right)$ de f (c'est à dire la jacobienne de son gradient). Posons $H = \frac{\partial^2 f}{\partial q^2}(q)$ la hessienne de f, donc $H_{i,j} = \frac{\partial^2 f}{\partial q_i \partial q_j}$.

Avec i et j respectivement les lignes et les colonnes de H, nous obtenons :

$$H_{i,j} = \sum_{k=1}^{n} \frac{\partial^2 T_k}{\partial q_i \partial q_j}(T_k(q) - T_k^{obs}) + \frac{\partial T_k}{\partial q_i}\frac{\partial T_k}{\partial q_j}$$

3.2.1 Résultats

Cette méthode ne donne pas les résultats escomptés, de gros problèmes apparaissent. On sait que, quels que soient les termes $\frac{\partial^2 T_k}{\partial q_i \partial q_j}$, H est définie positive au minimum de la fonction (on sait qu'il existe). Elle ne sera plus définie positive pour $(T(q) - T^{obs})$ plus grand qu'une certaine limite.

Sachant que :

$$\|T(q) - T^{obs}\| \geq \varepsilon \Leftrightarrow \|q - q_{opt}\| \geq \delta$$

Notre problème se situe à ce niveau. Après quelques tests, nous avons remarqué que la hessienne n'était plus définie positive pour $\|\delta q\| \geq 0.001$. La distance entre deux postures est trop petite, dans notre cas, nous traquons les mouvements toutes les 20 ms, et la plupart du temps nous avons $\|\delta q\| \geq 0.001$. L'algorithme ne pouvant converger correctement, une approche avec un algorithme de type Quasi-Newton est envisageable : on n'utilise alors une approximation de la hessienne de f.

3.2.2 L'algorithme quasi-Newton

On veut que la hessienne de f soit définie positive lorsque $T(q) = T^{obs}$, c'est à dire au minimum recherché.

On approxime ainsi la hessienne par :

$$\boxed{H_{i,j}^{approx} = \sum_k \frac{\partial T_k}{\partial q_i} \frac{\partial T_k}{\partial q_j}}$$

Jusqu'alors, nous n'avons pas introduit la notion de contrainte dans notre algorithme. Afin de pouvoir spécifier les bornes de q, nous allons utiliser l'algorithme *qld* qui permet de résoudre des problèmes d'optimisation quadratique. *qld* peut trouver :

$$min_x(\frac{1}{2}x^T Cx + d^T x)$$

Avec C : matrice NxN symétrique définie positive
d : Vecteur de dimension N.

Or résoudre l'équation $\frac{\partial f}{\partial q}(q) + \frac{\partial^2 f}{\partial q^2}(q)\delta q = 0$ revient à minimiser la fonction suivante :

$$\boxed{\left(\frac{\partial f}{\partial q}\right)^T (q)\delta q + \frac{1}{2}\delta q^T \frac{\partial^2 f}{\partial q^2}(q)\delta q}$$

Comme cette fonction est quadratique, il est possible d'utiliser *qld*.

Nous avons mis au point une fonction supplémentaire dans *invTags.sci* pour tester notre algorithme.

3.3 Résultats de notre algorithme

Toutes les données utilisées (Position des tags observée) pour les simulations ont été fournies par le simulateur du robot avec une période d'échantillonage de 10 ms. De même, les simulations sont effectuées de façon statique, pour chaque capture, l'optimisation démarre avec un vecteur q initial identique.

3.3.1 Robustesse aux occultations

Un des problèmes de notre système de capture est celui des occultations, c'est pourquoi il est indispensable de tester la robustesse de l'algorithme d'inversion.

Méthode pour tester la robustesse de l'algorithme

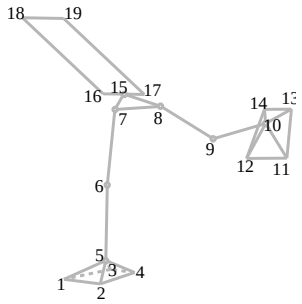
La hessienne de la fonction à minimiser est un point clé de l'étude numérique, en effet l'algorithme de minimisation que nous utilisons (*qld*) ne peut converger que si la hessienne est définie positive. Si son rang n'est pas plein, elle n'est plus définie positive, c'est pourquoi l'étude de son rang peut nous indiquer si l'algorithme peut converger vers une solution correcte ou non.

3.3.2 Robustesse à l'occulation d'un seul tag

Dans un premier temps, j'ai étudié l'impact de chaque tag occulté individuellement. Les résultats indiquent que le rang de la hessienne approximée est toujours plein. Cela signifie que si un seul tag est occulté, la reconstruction est numériquement possible. Dans un premier temps, ce résultat peut paraître étonnant, c'est pourquoi une étude plus complète de l'influence des occultations sur le rang est indispensable. Si ce résultat est conforté par la simulation numérique, il nous sera possible d'enlever des tags qui semblent inutiles (un tag sur chaque pied, et le tag 16 : la hanche droite).

Robustesse à l'occultation de deux tags

La première étude consiste à observer l'impact de l'occultation de deux tags sur le rang de la hessienne (figure 3.2). Si le rang de la hessienne n'est pas plein, cela signifie que la hessienne de la fonction à minimiser n'est plus définie positive, il sera difficile de trouver un minimum global.



Doublets	**Rang hessienne**	**Doublets**	**Rang hessienne**
(2 1)	23	(11 9)	24
(5 1)	24	(11 10)	24
(5 2)	24	(12 8)	24
(6 1)	24	(12 9)	24
(6 2)	24	(12 10)	24
(6 5)	24	(12 11)	23
(7 1)	24	(16 7)	24
(7 2)	24	(16 8)	24
(7 6)	23	(18 7)	24
(8 6)	24	(18 8)	24
(8 7)	23	(18 16)	23
(9 6)	23	(19 7)	24
(9 7)	24	(19 8)	24
(9 8)	23	(19 16)	23
(10 9)	24	(19 18)	23
(11 8)	24		

FIG. 3.2 – Chute de rang pour certains doublets

On remarque que l'influence des doublets est symétrique (par exemple une occultation des deux tags du pied gauche, comme du pied droit engendre une perte de rang de 2 de la hessienne, ...) ce qui tente à prouver que les résultats sont corrects (vu la symétrie du sujet). De même, un couplage étrange apparaît, il semble que certains tags soient couplés (comme (16 8), (16 7), (7 1), (7 2) ...) alors qu'il y a un découplage naturel de ces marqueurs de par la morphologie même du robot. Il faudrait étudier ce phénomène un peu plus en profondeur pour savoir pourquoi il apparaît.

Dans un second temps, nous avons voulu justifier l'utilisation du rang de la hessienne pour savoir si la reconstruction est possible ou non. Pour cela nous avons confronté les

erreurs de reconstruction (simulation numérique) et le rang de la hessienne (figure 3.3)

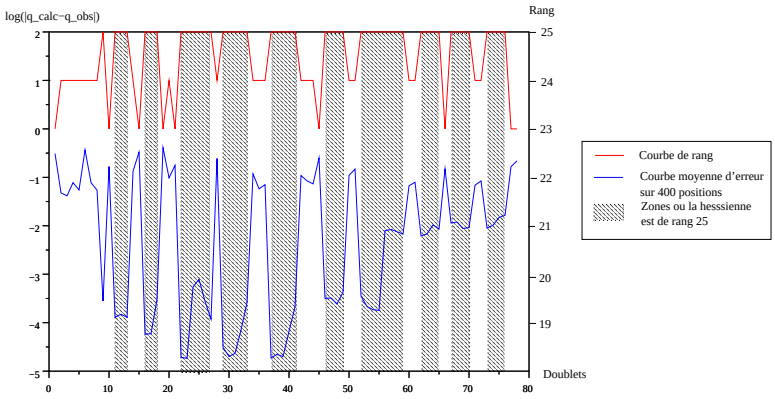


FIG. 3.3 – Confrontation erreurs - rang de la hessienne

Ce graphique montre assez bien le phénomène, mais il est plus intéressant de réorganiser les doublets (figure 3.4) pour regrouger tous ceux qui ne font pas chuter le rang de la hessienne, puis ceux qui le font chuter de 1, et ainsi de suite ...

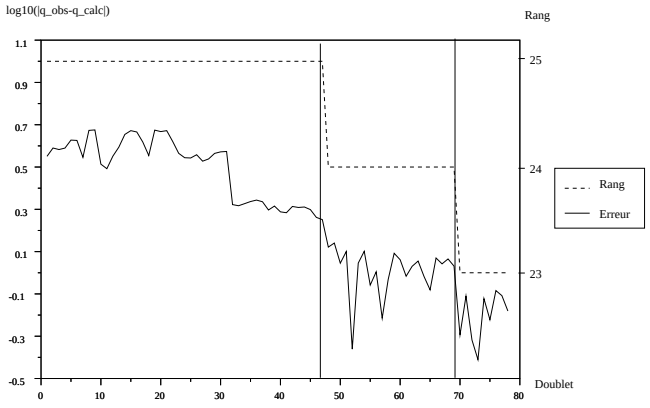


FIG. 3.4 – Nouvelle organisation des doublets

Ce graphique montre que l'erreur est très faible lorsque le rang est plein (de l'ordre de 10^{-3}) par contre lorsque le rang chute, il y a une augmentation non négligeable de l'erreur. Ceci prouve que le rang de la hessienne nous fournit une bonne information quant à la convergence de l'algorithme. On remarque une erreur plus importante (même

si le rang est plein) lorsque les tags 18 et 19 interviennent, c'est à dire pour une abscisse supérieure à 56 sur la figure 3.3. Une étude plus approfondie de ce phénomène est donc nécessaire.

On sait maintenant quand l'algorithme converge même lorsqu'il y a des occultations, il est intéressant de savoir en combien d'itérations il converge, vu que nous voulons que notre programme fonctionne en ligne.

Il est intéressant de regarder le comportement du rang de la hessienne, de l'algorithme lorsqu'on enlève quelques tags.

Robustesse à l'occultation lorsqu'on enlève 3 tags

Il est possible de faire plusieurs tests, enlevons le 16, 2, 12 puis les 16, 5 et 10. En fait on enlève soit un tag sur chaque pied ou on enlève les tags liés aux chevilles.

Tout d'abord, il est intéressant d'observer que le rang de la hessienne est plein lorsqu'on en lève ces trois tags ([16 2 12] et [16 5 10]). Maintenant regardons lorsqu'on occulte les tags pris individuellement (figure 3.5).

[16 2 12]		**[16 5 10]**	
Tag	**Rang hessienne**	**Tag**	**Rang hessienne**
1	23	1	24
5	24	2	24
6	24	6	24
7	23	7	24
8	23	8	24
9	24	9	24
10	24	12	24
11	23	11	24
18	23	18	23
19	23	19	23

FIG. 3.5 – Chute de rang pour certaines occultations

Avec seulement 10 tags, dès qu'il y a une occultation, il y a une chute de rang, cela signifie, que 10 est le nombre minimum de marqueurs pour pouvoir reconstruire correctement le vecteur de configuration q. Il est possible de remarquer que les deux triplets donnent quasiment les mêmes résultats, une chute de rang pour chaque occultation. Même si les chutes de rang sont moins importantes pour le deuxième, le résultat est le même, une mauvaise convergence de l'algorithme.

3.3.3 Temps de convergence

Le but de ce test est de regarger en combien d'itérations l'algorithme converge. Pour cela, les simulations sont toujours effectuées de façon statique. Ainsi, nous regardons en combien d'itérations l'algorithme se stabilise, nous pourrons alors définir un nombre d'itérations maximum pour notre algorithme. Nous allons observer la vitesse de convergence lorsqu'il n'y a pas d'occultation, puis lorsqu'il y un deux tags occultés.

Temps de convergence sans tag occulté

Nous savons qu'il n'y a pas de divergence lorsqu'il n'y a pas d'occultation, de plus le nombre maximum d'itérations pour converger est de 4, c'est pourquoi une valeur moyenne de la vitesse de convergence nous donnera une bonne information.

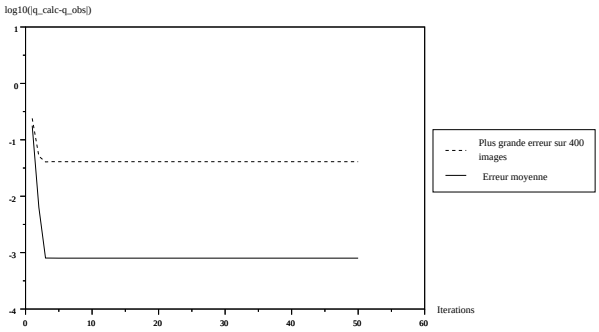


FIG. 3.6 – Convergence sur 400 images

Nous constatons qu'en moyenne (sur 400 images) il faut 3 itérations pour converger vers une solution (erreur de l'ordre de 10^{-3}), de même la plus mauvaise convergence donne une erreur de l'ordre de 10^{-1}, la meilleure convergence n'est pas représentée car l'erreur au bout de 3 itérations est nulle (elle n'est donc pas représentable dans notre graphique). S'il n'y a pas de tags occultés, la reconstruction se fait rapidement et avec une bonne précision. Il faut maintenant étudier la convergence lorsqu'il y a deux tags occultés pour voir si la vitesse de convergence est atteinte.

Temps de convergence avec deux tags occultés

Une première étude de la vitesse moyenne de convergence sur tous les doublets et sur les 400 images nous permet de voir en combien d'itérations l'algorithme converge, de la même manière nous pourrons voir l'erreur moyenne sur 400 images et sur les 78 doublets.

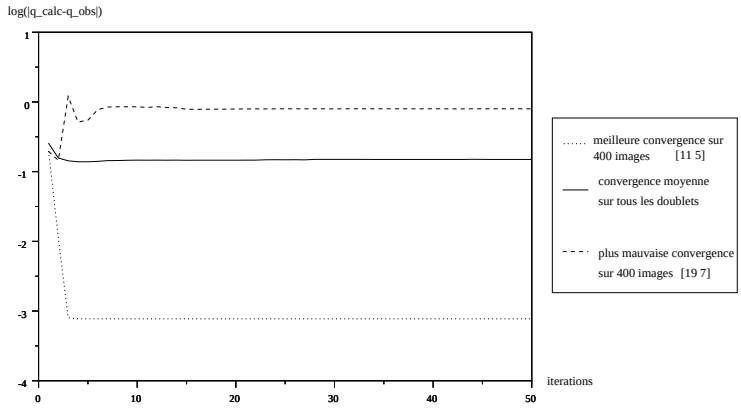


FIG. 3.7 – Convergence moyenne sur 400 images

Nous constatons que l'algorithme se stabilise assez rapidement (figure 3.7), que la convergence soit très bonne (précision de l'ordre de 10^{-3} m) ou qu'elle soit très mauvaise (de l'ordre de 1 m pour tous les paramètres de q soit 0.04 m pout chaque paramètre en moyenne), l'algorithme se stabilise très rapidement, en 10 itérations. Nous pourrons limiter le nombre maximum d'itérations à 10. De même, en moyenne sur les 78 doublets possibles et sur 400 images, la précision est de l'ordre de 0.15 m pour q soit 6mm pour chaque paramètre.

3.3.4 Influence du bruit sur la robustesse de l'algorithme

Un des problèmes de la capture de la marche humaine se trouve au niveau du bruit engendré par les glissements de peau et par la simplification de notre modèle de la personne (on modélise le genou par un simple pivot, alors qu'en réalité il y a 3 rotations et 1 translation). Une étude plus approfondie sur l'impact d'un bruit gaussien sur la reconstruction est donc intéressante, voire indispensable pour mieux évaluer la robustesse de notre algorithme. Après injection d'un bruit gaussien sur les tags observés, on obtient un résultat tout à fait intéressant(cf Figure 3.8), l'erreur sur q est proportionnelle à la variance du bruit.

Dans notre cas, pour les paramètres qui sont au nombre de 4, un bruit d'une variance 0.0125 m engendre une erreur cumulée (des 4 paramètres) de 0.025 m, ce qui est assez faible et prouve que notre algorithme est très stable au bruit. Cependant une étude plus approfondie des conséquences du bruit doit être entreprise pour expliquer quelques phénomènes indiqués sur la figure 3.8, comme l'ordonnée à l'origine de la deuxième courbe que nous ne pouvons expliquer pour l'instant.

Nous sommes capables de reconstituer la posture de la personne, sa position et son

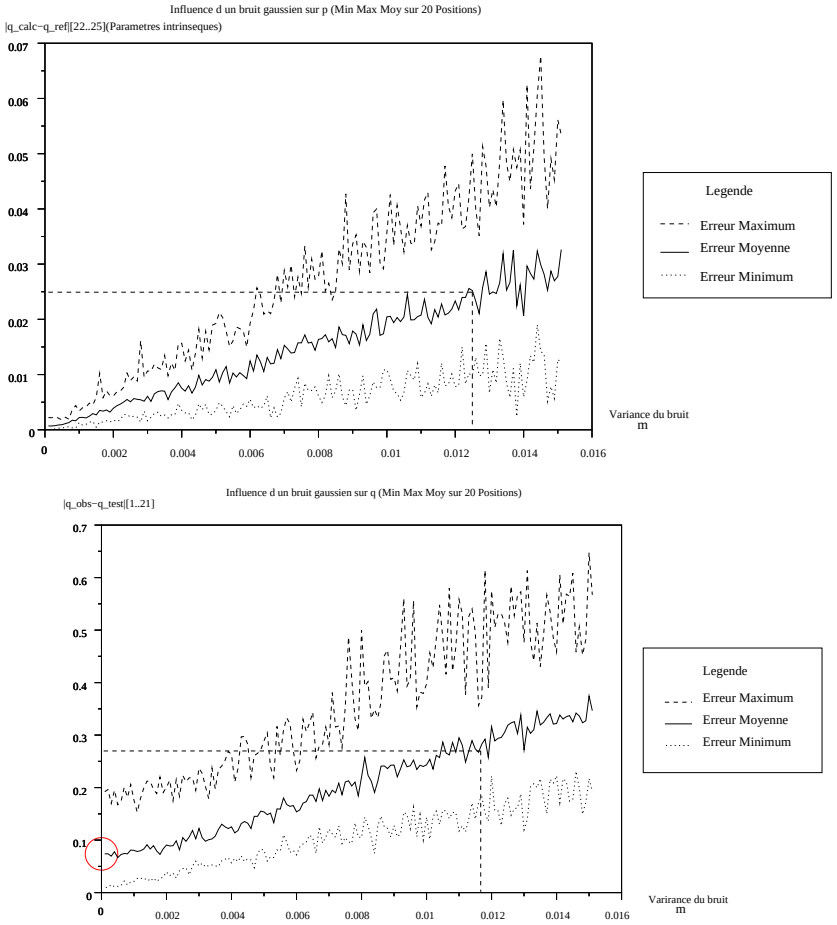


FIG. 3.8 – Influence du bruit sur la reconstruction de la personne

orientation dans l'espace. Ce calcul est l'interface entre la capture de mouvements avec l'OPTOTRAK et le simulateur du robot BIP, il faut intégrer ces calculs numériques dans le programme pour effectuer les premières expériences.

Troisième partie

Intégration logicielle et matérielle : Premières expériences

Chapitre 1

Intégration logicielle

1.1 Librairie de manipulations des matrices

Pour l'inversion numérique, nous avons eu besoin d'une librairie d'algèbre linéaire. Deux possibilités sont envisageables :

- Utiliser une librairie existante comme *LAPACK* ou *MTL*
- Créer sa propre librairie de manipulations de matrices

La *MTL* ou la *LAPACK* sont des librairies assez complexes à mettre en oeuvre, et à utiliser. C'est pourquoi, comme les manipulations de matrices de notre algorithme sont basiques, nous avons préféré développer notre propre librairie C++ : *Matrix* (*Annexe J* et tableau 1.1).

Classe Matrix	
Méthodes	**Description**
Matrix	Constructeur
display	Affichage de la matrice
getNRows	Renvoie le nombre de lignes
getNColumns	Renvoie le nombre de colonnes
setElement	Affecte un élément dans la matrice
getElement	Renvoie un élément de la matrice
copyMatrix	Copie une matrice
det	Renvoie le déterminant de la matrice
transpose	Renvoie la transposée de la matrice
sum	Renvoie la somme de deux matrices
mult	Renvoie la multiplication de deux matrices
scale	Multiplie la matrice par un scalaire
power	Renvoie une puissance de la matrice
getRows	Renvoie les lignes d'une matrice
getColums	Renvoie les colonnes d'une matrice
cat	Concatène deux matrices
delRow	Supprime une ligne
delCol	Supprime une colonne
coFact	Renvoie la matrice des coFacteurs

FIG. 1.1 – Description de la librairie Matrix

Cette librairie n'est certes pas optimisée comme le serait une librairie déjà existante, mais elle est amplement suffisante pour nos besoins.

1.2 Librairies Numériques

1.2.1 Sur-couche objet pour l'algorithme qld

A l'origine, la fonction *qld* est une fonction FORTRAN, nous sommes en possession d'une version C qui a été généré par f2C. Encore une fois, pour une question d'homogénéité, nous avons créé une classe *qld* qui utilise cette fonction C, et qui simplifie l'utilisation de cet algorithme. Une description plus complète de la classe est fournie en *Annexe K*).

1.2.2 Librairie numérique spécifique à notre projet

Toutes les fonctions utilisées lors de la simulation numérique sous *SCILAB* doivent être reproduites en C++ pour pouvoir être utilisées avec le simulateur 3D et la capture de mouvement. Pour cela, nous avons développé une classe *PERSON* (*Annexe L*) qui, à partir de la position des capteurs 3D de l'OPTOTRAK, nous fournit la posture, la position et l'orientation dans l'espace de la personne afin de les reproduire sur le simulateur 3D de BIP.

Classe PERSON	
Méthodes	**Description**
Person	Constructeur
calculatePosture	calcule la posture de la personne traquée à partir de la position des tags
getPosture	renvoie les $q_{1..21}$ optimisés
getParams	renvoie les $q_{22..25}$ optimisés
calculateJacobian	calcule la jacobienne de T
calculateHessian	calcule l'approximation de la Hessienne de la fonction à minimiser
calculateGradient	calcule le gradient de la fonction à minimiser
badTag	Renvoie la matrice après avoir supprimé des lignes

FIG. 1.2 – Description de la Classe PERSON

1.3 Documentation des différentes librairies

Ce rapport ne se veut pas une documentation complète sur l'utilisation des différentes librairies développées pour ce projet. Par contre, une documentation automatique HTML a été écrite pour chaque librairie à l'aide de Doxygen.

Doxygen est un système de génération de documentation à partir de fichier de code C++, C, Java, IDL et certains PHP :

- il peut générer une documentation en ligne (en HTML) et/ou un manuel hors-ligne(en Latex) à partir de code commenté. La documentation est directement extraite des commentaires du code source.

- Doxygen peut aussi, à partir du code source (non commenté) créer des graphes d'héritage, de collaboration, le tout entièrement généré automatiquement.

Chapitre 2

Intégration matérielle et premières expériences

2.1 Intégration matérielle

Afin de pouvoir effectuer des tests de suivi, nous avons créé un petit personnage en polystyrène (cf Figure 2.1) avec quasiment les mêmes ddl que BIP. Cela permet de disposer d'un sujet pour nos expériences.

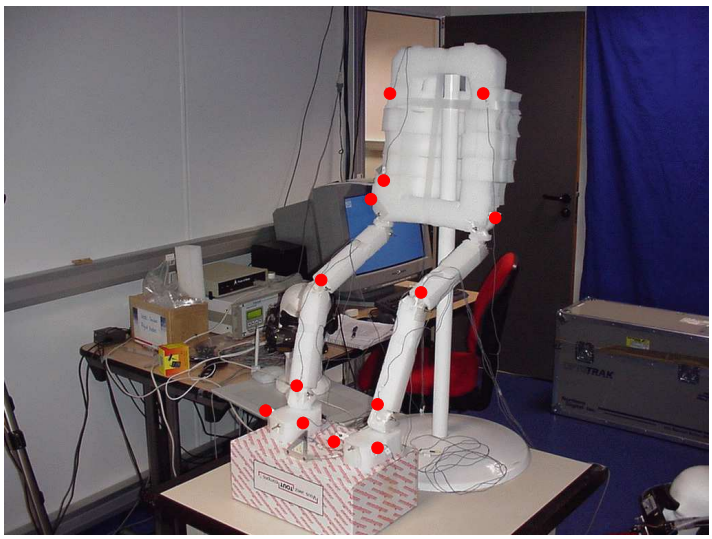


FIG. 2.1 – Pantin de polystyrène

On remarque que les capteurs (cf Figure 2.2) sont positionnés aux mêmes endroits stratégiques que ceux définis lors des simulations numériques, et qu'ils sont tous collectés dans les strobers.

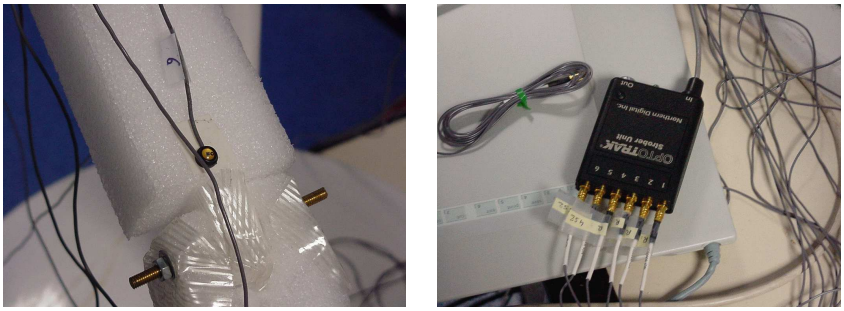


FIG. 2.2 – Capteurs IR et *Strober* de l'OPTOTRAK

De même, on remarque dans la figure 2.3 que le pantin est exactement en face de l'OPTOTRAK, dès que le sujet se tourne, des occultations peuvent intervenir. Pour de plus amples mouvements, il faudrait utiliser plusieurs OPTOTRAK. Cependant pour de premières expériences, le type de disposition illustrée par la figure 2.3 suffit amplement à donner des résultats très satisfaisants.

FIG. 2.3 – Disposition de l'OPTOTRAK par rapport au pantin

2.2 Premières expériences

Les premières expériences de capture de mouvements se sont très bien passées, l'OPTOTRAK nous a fourni de bonnes données, malgré des occultations intempestives.

Notre algorithme d'inversion est très robuste et tolérant à ce type d'erreurs, ce problème n'interfère donc pas avec la bonne reconstitution de la posture.

Les oscillations dûes au bruit ne sont pas visibles avec un sujet tel que le pantin. Etant donné qu'il ne s'agit pas d'un être humain, il n'y a donc pas de glissements de peau, ni même de bruit lié à la capture des mouvements d'un être humain. Les prochaines expériences, avec un sujet humain, nous fourniraient plus de données quant au bruit inhérent à la personne humaine.

Pour l'instant, les premières expériences ont montré l'efficacité de notre algorithme d'inversion cinématique. Plus généralement, elles ont prouvé que le projet est réalisable, c'est à dire, qu'il est possible, en temps réel de traquer les mouvements et de les reproduire sur un simulateur.

Conclusion

L'objectif de ce projet consistant à capturer les mouvements d'un être humain et de le reproduire sur un simulateur 3D du robot BIP2000 en temps réel a été atteint, j'ai mis en oeuvre une application qui fonctionne. Cependant le travail n'en est qu'à ses débuts, puisque ce projet n'est qu'une infime partie du domaine, encore peu exploré, de la capture de mouvements.

J'ai remarqué les points critiques de la capture de mouvement, qui concernent principalement l'inversion cinématique et l'aspect *temps réel*. Pour ce qui est de l'inversion cinématique, elle doit tenir compte des différences de morphologie entre la personne dont on capture les mouvements et le personnage du simulateur, de la simplification du modèle de la personne, et du bruit dans la capture de mouvements (glissements de peau, ...). La méthode retenue dans notre cas consiste à traduire ce calcul en problème d'optimisation. Cette méthode est très puissante et fonctionne assez bien. Même si dans ce projet, je ne fais que survoler l'optimisation, je me suis rendu compte que ce domaine est complexe et qu'il existe beaucoup d'algorithmes différents, qui s'appliquent plus ou moins bien au problème que l'on veut traiter. Ici, nous avons utilisé un algorithme Quasi-Newton, il donne de bons résultats et semble robuste, mais cela reste à confirmer par une étude numérique approfondie. L'autre point critique du projet concerne l'aspect temps réel. Il faut capturer le mouvement, faire l'inversion cinématique et afficher la posture à l'écran. Grâce à la puissance de calculs de la SGI ONYX, il n'y a, à l'heure actuelle, qu'un léger décalage (constant) entre le mouvement de la personne et sa reproduction sur l'écran.

Malgré ces résultats intéressants et encourageants, il reste encore beaucoup de travail. Plusieurs points restent à aborder :

- continuer les expériences sur des sujets humains,
- faire quelques mesures sur les données fournies par l'OPTOTRAK pour voir si elles sont bruitées et quelles sont les caractéristiques de ce bruit,
- approfondir l'étude du comportement numérique de l'inversion cinématique,
- mettre en place l'équation de la dynamique dans le simulateur 3D du robot BIP,
- améliorer le simulateur pour une aide à la génération de trajectoire du robot BIP.

Annexes

Annexe A

Scene graph du simulateur BIP

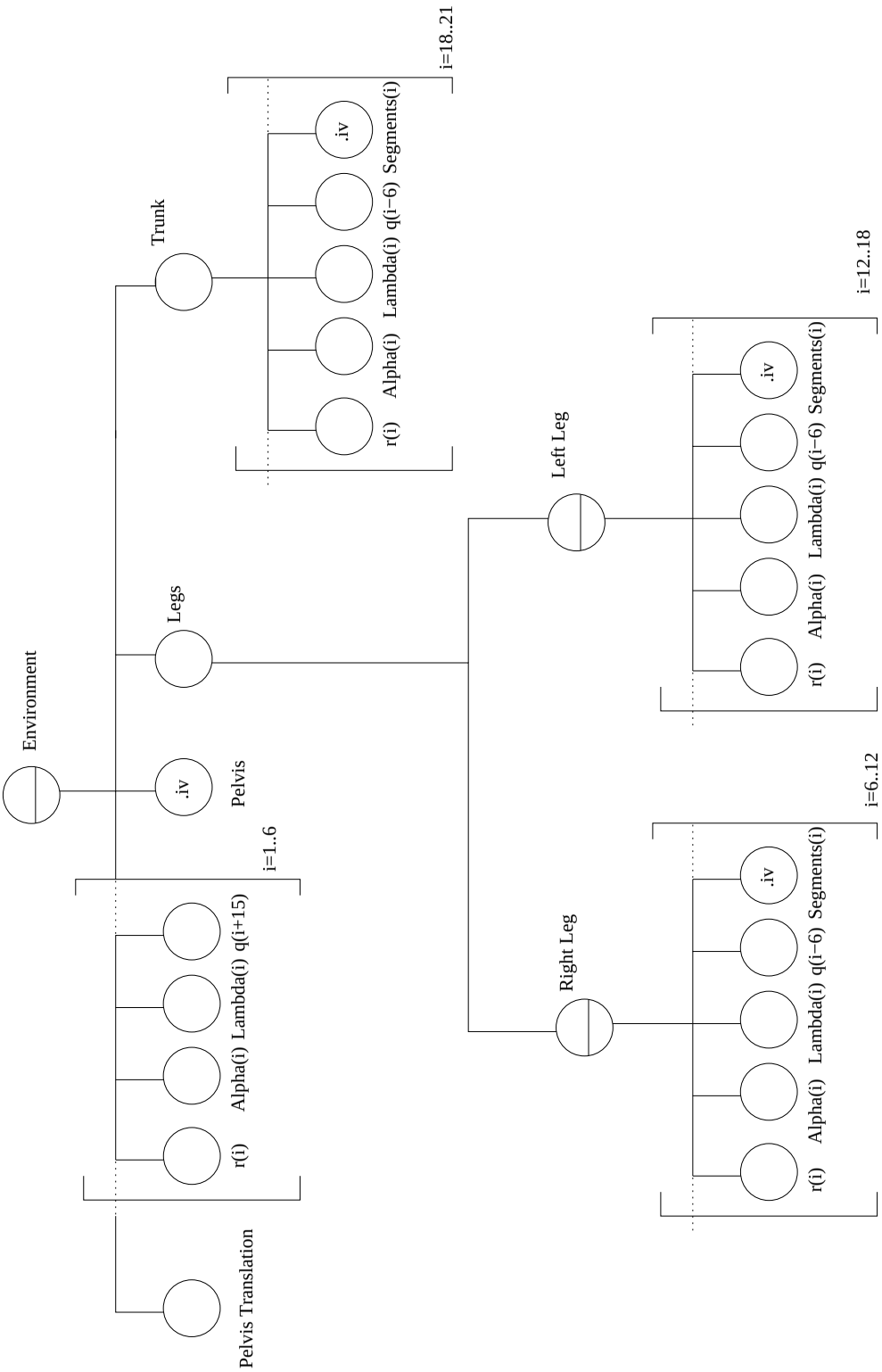


FIG. A.2 – *Scene Graph* du simulateur BIP

Annexe B

Classe Robot

```
// ------------------------------------------------------------------ C++
/*! \file  modeleBIP.h
 *  \brief
 *
 *-------------------------------------------------------------------------<br>
 * Creation    : 2nd of july 2002
 * Author(s)   : WIEBER Pierre-Brice modified by GUILBERT Matthieu    <br>
 *
 *-------------------------------------------------------------------------<br>
 * Contents:
 *      - Open Inventor model : Robot BIP model Class                  <br>
 *      -> Robot builder defined by a Denavit Hartenberg model         <br>
 *      -> change the posture and the position of the robot            <br>
 *
 */

// ------------------------------------------------- Standard Includes
#include <Inventor/SoDB.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/SoInput.h>
#include <Inventor/nodes/SoGroup.h>
#include <Inventor/nodes/SoRotationXYZ.h>
```

```
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/fields/SoSFVec3f.h>
// ------------------------------------------------------- Own Includes

#ifndef MODELEBIP_H
#define MODELEBIP_H

//------------------------------------------------- Constant Declaration
#define N_REPERES 21
static const double pi = 3.1415926;

//----------------------------------------Independant function Prototype
SoNode *get_segment_from_file(char *);

// -------------------------------------------------------Class Definition
////////////////////////////////////////////////////////////////////////
// Robot Class                                                        //
////////////////////////////////////////////////////////////////////////
/*!
 *  \class Robot
 *  \brief  Open Inventor BIP model
 *
 *  This Class permit :                                          <br>
 *                        - create the BIP robot                 <br>
 *                        - move the robot in a scene            <br>
 *
 *  \note How to use this Class :                                <br>
 *        - create the robot                                     <br>
 *        - move it in the scene
 *
 *  \code Robot *BIP = new Robot(...); // Create BIP
 *        BIP->SetNewPos(...);         // Set a new position
 *        BIP->Move();                 // make it moving
          ....
  delete BIP;
 *  \endcode
 *
 *  \note This class cannot be used alone, a viewer is needed    <br>
 *        I advise to use the QGLViewer from Gilles Debunne
 *        (Projet IMAGIS : INRIA Rhone-Alpes)                    <br>
 *
```

```
 */
class Robot
{
public :
   /*! @name --------------------------- Constructor(s) - Destructor(s)*/
  //@{
  /*! Constructor:                                                    <br>
   *              -> Create the inventor DataBase                    <br>
   *              -> Put all the DOFs (Rotations and translations)   <br>
   *              -> Denavit Hartenberg Transformations to build the
   *                 robot                                           <br>
   *              -> Import all the segments (.iv files) from a
   *                 directory specified in parameter                <br>
   *              -> build the robot from the DH parameters          <br>
   *
   * @param descriptionPath (char *) : directory where are the files (.iv)
   *
   * \note ALLOCATION OF MEMORY for the SoRotations, SoTranslations
   *
   */
   Robot(char *);
  //@}

  /*! @name ------------------------------------------- Robot movement*/
  //@{
  /*! move: update the SoRotations and the SoTranslations           <br>
   *
   * \note Updated from the articular coordinates
   *       (private property : coord[])
   */
   void Move(double *);
  //@}


  /*! @name -------------------------------- open Inventor root database*/
  //@{
  /*!
   * root : Open Inventor Robot root Database
   *
   * \note Needed by Open Iventor
   */
```

```
  static SoSeparator *root;
  //@}

private :

  static double r[N_REPERES];
  static double alpha[N_REPERES];
  static double lambda[N_REPERES];
  static double theta[N_REPERES];
  static double signe[N_REPERES];
  static int   indice[N_REPERES];
  static int   element[16];
  SoSeparator *Environment;
  SoGroup     *Legs;
  SoGroup     *Trunk;
  SoSeparator *RightLeg;
  SoSeparator *LeftLeg;

  SoNode * segment_node[17];
  static double coord[21];
  static SoRotationXYZ * articulation[N_REPERES];
  static SoTranslation * pelvistranslation;

};

#endif
```

Annexe C

Classe BIPViewer

```
// ------------------------------------------------------------------ C++
/*! \file  BIPViewer.h
 *  \brief
 *
 *-------------------------------------------------------------------<br>
 * Creation          : 2nd of july 2002
 * Author(s)         : GUILBERT Matthieu           <br>
 *
 *-------------------------------------------------------------------<br>
 * Contents: - Derived class from QGLViewer, adapted to
 *             BIPViewer project
 *           -> Animation of the robot (Movement())
 *           -> loads Inventor BIP elements
 */

// ------------------------------------------------- Standard Includes

// ------------------------------------------------------ Own Includes
#include <qglviewer.h>
#include "modeleBIP.h"
#include <Matrix.h>
```

```
#ifndef BIPVIEWER_H
#define BIPVIEWER_H
#define NTAGS 13

class BIPViewer : public QGLViewer {
Q_OBJECT

public :
  BIPViewer();
  ~BIPViewer();
  void Init();
  void GetPositionCamera(); //To get the position of the camera
  void Movement(double *);
  static volatile bool stopMainLoop; //In order to quit the Viewer

public slots :
  void tellMainLoopToStop();

protected :
  void draw();
  void keyPressEvent(QKeyEvent *);

private :
  Robot *BIPOU;                  // The robot
  Matrix *currentTagsPosition;
};

#endif
```

Annexe D

Champ de vision de l'OPTOTRAK

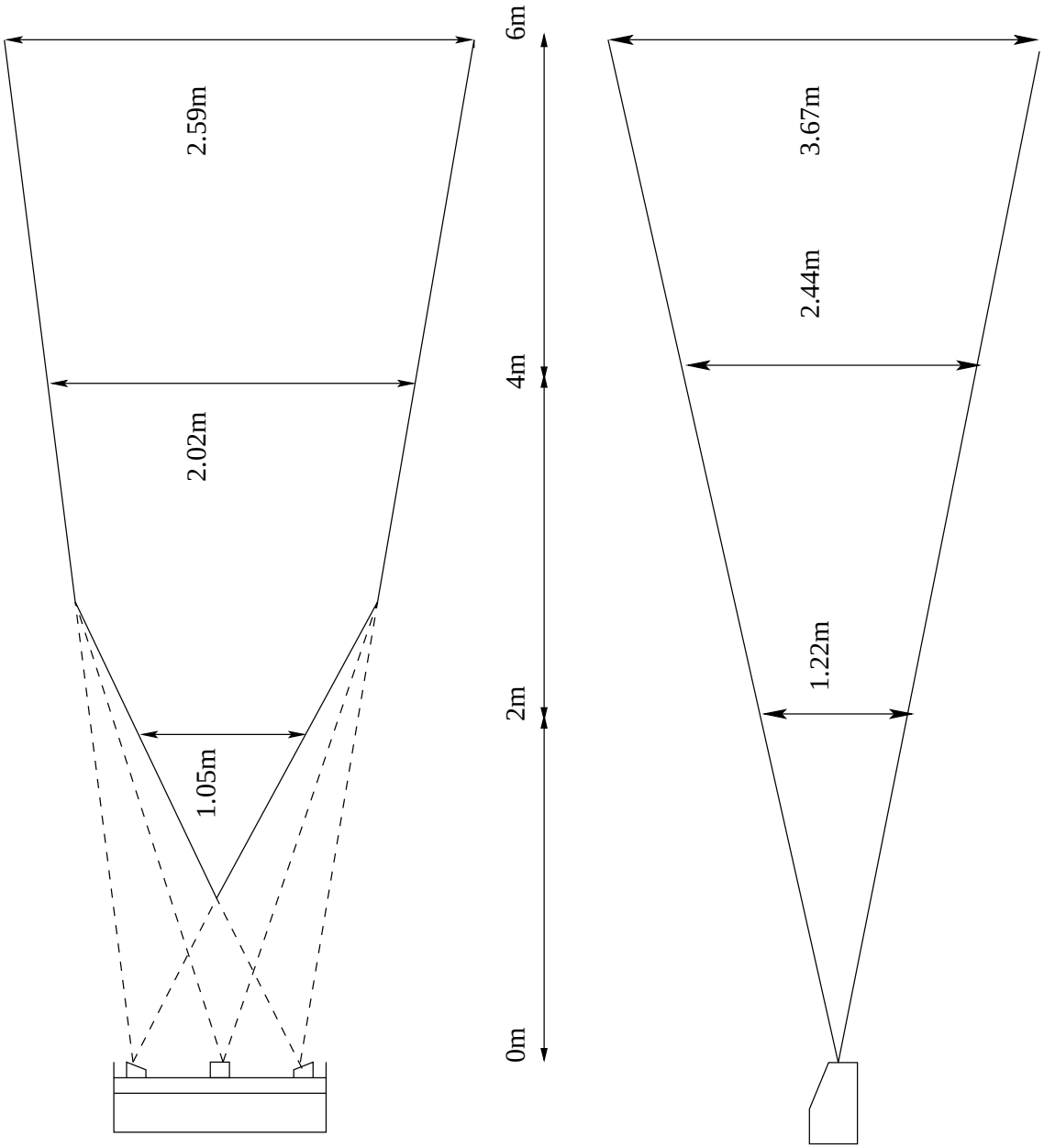


FIG. D.2 – Champ de vision de l'OPTOTRAK

Annexe E

Classe Optotrak

```
// ------------------------------------------------------------ C++
/*! \file  Optotrak.h
 *  \brief
 *
 *-------------------------------------------------------------<br>
 * Creation          : 30th of march 2002
 * Author(s)         : Laurence Boissieux           <br>
 *
 *-------------------------------------------------------------<br>
 * Contents: - Using of Optotrak IR sensors
 *           - the Optotrak class allows to
 *             - initialize the system
 *             - retrieve 3D realtime position(s) with
 *               blocking or non-blocking methods
 *             - shutdown the system
 *
 */

#ifndef OPTOTRAK_H
#define OPTOTRAK_H
```

```
// ---------------------------------------------- Standard Includes
#include <vector>
#include <string>

// XML include
#include <qdom.h>
#include <qfile.h>


// ------------------------------------------------ Own Includes
#include "Vec3.h"
#include "ndtypes.h"
#include "ndpack.h"
#include "ndopto.h"

#define MAX_NB_RIGID 10
#define RAD_TO_DEG 57.2957

////////////////////////////////////////////////////////////////
//  Optotrak Class                                            //
////////////////////////////////////////////////////////////////

/*! \class Optotrak
 *  \brief Class to interface with Optotrak Sensor device
 */
class Optotrak
{
public:
  //! @name Constructor(s) - Destructor(s)
  //@{
  /*
   *
   */
  Optotrak(int nbM=1, int nbR=0,int nbSrober1=1, int nbStrober2=0,
   int nbStrober3=0, int nbStrober4=0, float mFreq=2800,
   float fFreq = 100,
   int thresh=30, int mGain=160, int sData=1,
   float dCycle=0.35, float volt=7.0, float cTime=5.0,
   int cFlags= OPTOTRAK_BUFFER_RAW_FLAG,
   int rFlags= OPTOTRAK_QUATERN_RIGID_FLAG | \
   OPTOTRAK_RETURN_QUATERN_FLAG);
```

```
  ~Optotrak();
  //@}

  //! @name Functional Methods
  //@{
void parse(char*);


/* Sets up the optotrak system by determining its config,
   * loading and initalizing its transputers, loading its
   * camera sensors parameters.
   * Must be done once at the beginning of the session
   */
  void init(bool creatNIF=false, char *collectionFile=NULL);

  void addRigidBody(int id, int firstMarker, char* file);

  /* Starts optotrak device by activating markers */
  void start();

  /* Blocking method to retrieve data: request data, block til some
     are available and receive */
  void update();


  /* Non-blocking methods to retrieve data*/
  /* Request the latest data (but doesn't receive it and doesn't
     block while waitintg for it)*/
  void requestData();
  /* Check availability of data, must be called before any
     receiving*/
  bool dataReady();
  /* Effectively get data : receive it. Data must be requested
     (requestData) and checked (dataReady) before any receiving */
  void receiveData();

  Vec3 getPos(int markerIndex);


  void getRigidTransform(int rigidBodyIndex,
 Vec3& translation,
```

```
 Vec3 &eulerAngles);

  void getRigidTransform(int rigidBodyIndex,
 Vec3& translation,
 Quaternion& rotation);

  /* Set the static reference frame to the rigid body specified
     by its ID*/
  void setStaticReferenceFrame(int rigidBodyIndex);

  void setMovingReferenceFrame(int rigidBodyIndex);

  /* Turns off the markers */
  void stop();

  /* Shutdowns the optotrak system */
  void shutdown();
  //@}

  /* Prints values of all optotrak attributes
   */
  void status();

int getNbMarkers() { return nbMarkers; }


  /* Gives optotrak error diagnostic and shutdowns the system */
  void error();

private :
  int nbSensors;
  int nbMarkers; /*! < total number of markers */
  int nbMarkersOnStrober1;/*! < number of markers on Strober 1*/
  int nbMarkersOnStrober2;/*! < number of markers on Strober 2*/
  int nbMarkersOnStrober3;/*! < number of markers on Strober 3*/
  int nbMarkersOnStrober4;/*! < number of markers on Strober 4*/
  float frameFrequency; /*! <data collection frame frequency*/
  float markerFrequency;
  int threshold;

  int minimumGain;
```

```
  int streamData;
  float dutyCycle; /*!< percentage of time a marker is actually turned
     on [0..1] */
  float voltage; /*!<voltage level used for strobing the markers */
  float collectionTime; /*! Duration time for data collection
  [0..99999] */
  int collectionFlags; /*! */

  char errorMessage[MAX_ERROR_STRING_LENGTH + 1];

  Position3d* posData;

  int rigidBodyFlags;
  int nbRigidBody;
  bool hasRigidBody;
std::vector<std::string> rigidBodyNames;
std::vector<int> rigidBodyFirstMarkers;


  struct OptotrakRigidStruct* rigidData;

};

#endif
```

Annexe F

Classe Capture

```
// ------------------------------------------------------------------ C++
/*! \file  Capture.h
 *  \brief
 *
 *-------------------------------------------------------------------<br>
 * Creation          : 5th of july 2002                               <br>
 * Author(s)         : GUILBERT Matthieu     <br>
 *
 *-------------------------------------------------------------------<br>
 * Contents: - Sub-Class of the Optotrak Class adapted to motion Capture
 *           -> Getting Positions of all the markers
 *           -> Getting all the occultations
 */
// ---------------------------------------------------Standard Includes
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <iostream.h>
#include <Matrix.h>
// ------------------------------------------------------- Own includes

#include "/h/pimprenelle/imagis/boissieu/SRC/OPTOTRAK/Optotrak.h"
```

```
#ifndef CAPTURE_H
#define CAPTURE_H

class Capture : public Optotrak {

public :
  Capture(int,int,int,int,int);
  ~Capture();
  double *getPosition();
  int    *getOccultations();
  int    getNOccultations();

private :
  int    NMarkers;
  double *pos;
  int    *occultation;
  int    Noccultation;
};

#endif
```

Annexe G

Extrait de TrackCine.maple

```
#
# Cinematique du Robot BIP2000
#
# BASE LIBRE == pelvis
#
# Paramètres de DENAVIT-HARTENBERG MODIFIÉ (KHALIL-KLEINFINGER)
#
# Pierre-Brice le 2/3/01
#
# Adaptee par Matthieu GUILBERT 12/06/02 pout la track d'une personne
#

N_TAG := 19;
# Nombre de solides
NSOL := 17+6:

# Nombre de degrés de liberté
NDDL := 15+6:

#Nombre de parametres a determiner de DENAVIT HARTENBERG
NPAR := 4:
```

```
# Définition des coordonnées et vitesses généralisées
q := vector(NDDL+NPAR):
qdot := vector(NDDL):

# Coordonnees de la base de reference
base := vector([q[16], q[23]+q[24]+q[17], q[18]]):

# Repère 1 : translation (annulee)
ref_1    := 0:
r_1      := 0:
lambda_1 := 0:
alpha_1  := -Pi/2:
theta_1  := 0:

# Repère 2 : translation (annulee)
ref_2    := 1:
r_2      := 0:
lambda_2 := 0:
alpha_2  := Pi/2:
theta_2  := Pi/2:

# Repère 3 : translation (annulee)
ref_3    := 2:
r_3      := 0:
lambda_3 := 0:
alpha_3  := Pi/2:
theta_3  := Pi/2:

# Repère 4 : rotation Y lacet
ref_4    := 3:
r_4      := 0:
lambda_4 := 0:
alpha_4  := Pi/2:
theta_4  := Pi/2+q[20]:

# Repère 5 : rotation Z tangage
ref_5    := 4:
r_5      := 0:
lambda_5 := 0:
alpha_5  := Pi/2:
theta_5  := Pi/2+q[21]:
```

Annexe H

Influence des tags sur la reconstruction

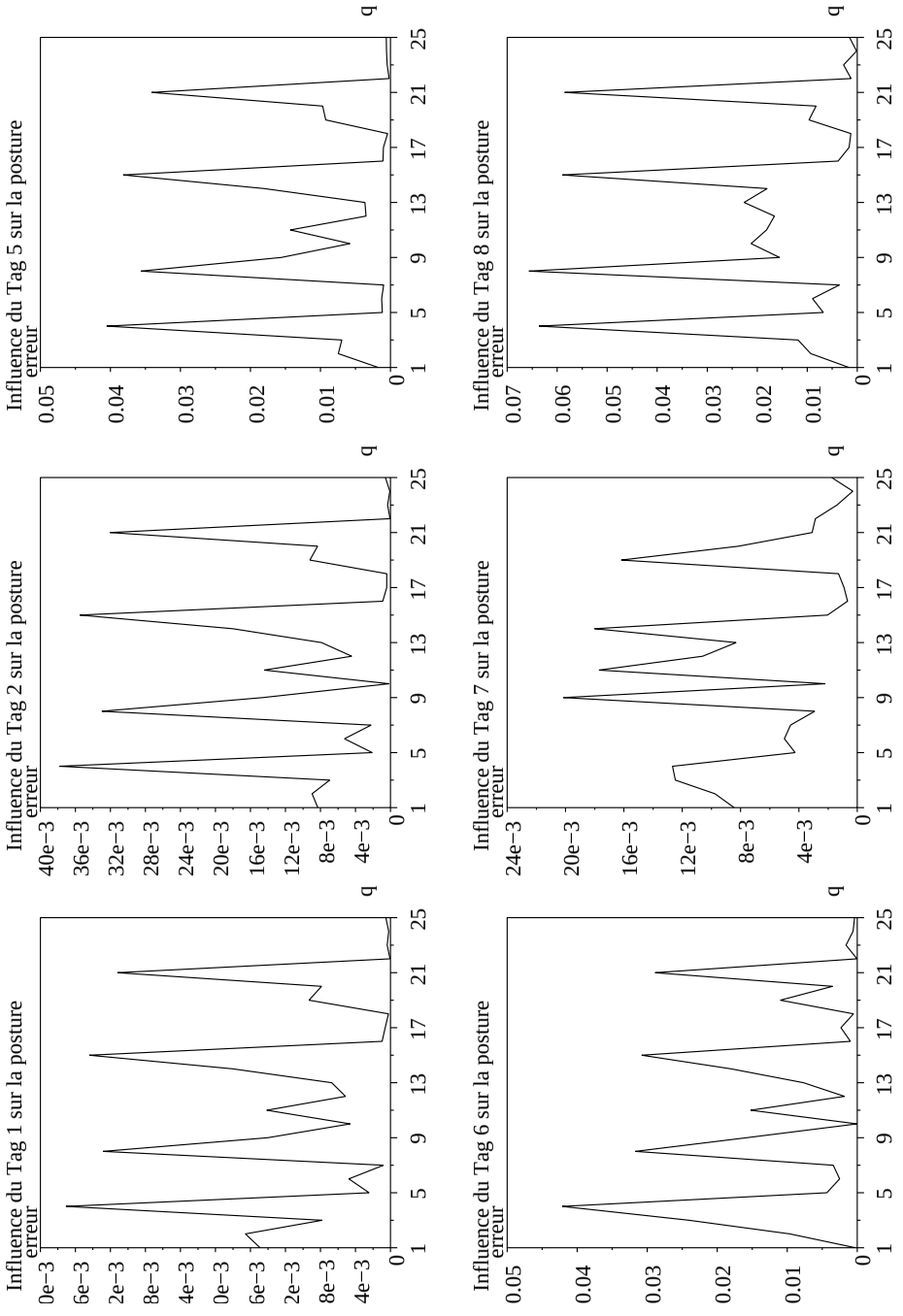


FIG. H.2 – Influence des tags sur la reconstruction

Annexe I

Influence des doublets sur reconstruction

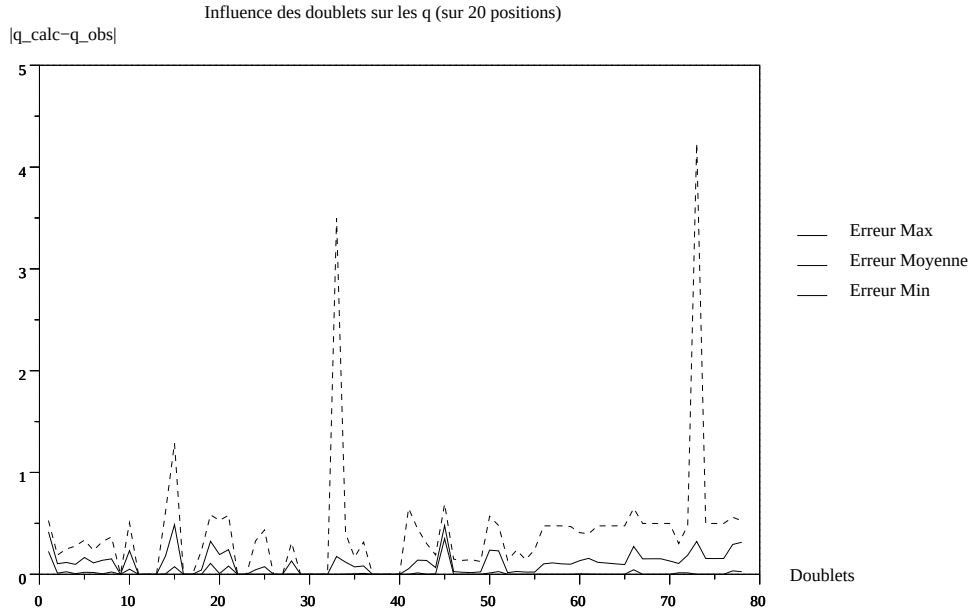


FIG. I.2 – Influence des doublets sur reconstruction des q

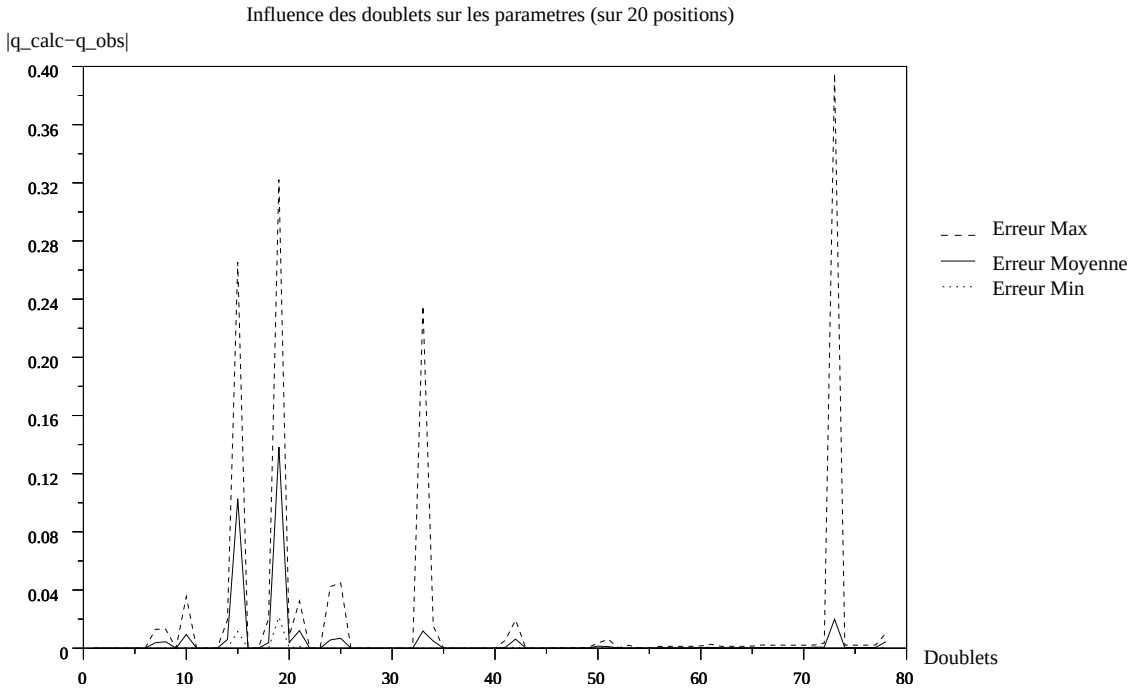


FIG. I.3 – Influence des doublets sur reconstruction des paramètres

Annexe J

Classe Matrix

```
// ------------------------------------------------------------- C++
/*! \file  Matrix.h
 *  \brief
 *
 *-------------------------------------------------------------------<br>
 * Creation          : 8th of july 2002                              <br>
 * Author(s)         : GUILBERT Matthieu                             <br>
 *
 *-------------------------------------------------------------------<br>
 * Contents: - Matrix Library <br>
 *           -> Matrix Definition <br>
 *           -> Determinant <br>
 *           -> multiplication <br>
 *           -> Inverse <br>
 *           -> transpose <br>
 *           -> co-Factor matrix <br>
 *           -> scaling <br>
 *
 *        Class implemented in Matrix.c
 */
// ------------------------------------------------Standard Includes
#include <stdio.h>
#include <stdlib.h>

// ------------------------------------------------- Own includes

#ifndef MATRIX_H
#define MATRIX_H
```

```
////////////////////////////////////////////////////////////////////
// Matrix Class                                                   //
////////////////////////////////////////////////////////////////////
/*!
 *  \class Matrix
 *  \brief  Linear Algebra Library : Class for Matrix manipulation
 *
 *  This Class permit :                                          <br>
 *               - to define a Matrix                            <br>
 *               - to manipulate Columns and Rows of a Matrix    <br>
 *                       - Matrix summation                      <br>
 *                       - Matrix multiplication                 <br>
 *                       - Matrix inversion                      <br>
 *                       - Matrix transposition                  <br>
 *                       - Matrix scaling                        <br>
 *                       - to calculate a Matrix Determinant     <br>
 *                       - to calculate the Co-Factor Matrix     <br>
 *
 *  \note All Matrix are dynamically allocated                   <br>
 *    When a method returns a Matrix : the method allocates
 *    memory for the returned Matrix.
 *    For Example : Creating a Matrix                            <br>
 *  \code Matrix *M = new Matrix(...);
 *  \endcode
 *
 *  \note Then when a matrix is manipulated by a method          <br>
 *        (for example : an Inversion) :                         <br>
 *
 *  \code Matrix *M = new Matrix(...);
 *        Matrix *Inversed_M;
 *        Inversed_M = M->inverse();
 *   ...
 *        delete Inversed_M;
 *        delete M;
 *  \endcode
 *
 *  \warning Don't forget to delete the Matrix each time you
 *   create a Matrix
 */
class Matrix
```

```
{
public :
  /*! @name ---------------------- Constructor(s) - Destructor(s)*/
  //@{
  /*! Contructor:
   *              - allocate memory for the matrix
   *              - set the number of columns and rows
   *              - fill the Matrix with elements of the array
   *  @param elements : unidimensional double array where rows are
   *                              concatanated
   *    (Example Array :
   *                               {a(1,1), a(1,2), a(2,1), a(2,2)})
   *  @param  NRow    : Number of Rows    (integer)              <br>
   *  @param  NCol    : Number of Columns (integer)              <br>
   *
   *  \note ALLOCATION OF MEMORY
   */
  Matrix(double [], int, int);
   /*! Destructor :
    *              -  disallocates name from shared area if not NULL
    *              -  disallocates all the Memory allocated by the
    *                 constructor for the Matrix
    */
   ~Matrix();
  //@}

  /*! @name ---------------------------------------- Matrix display*/
  //@{
  /*! Display:
   *          - Display Matrix like Scilab                       <br>
   *            For a 2x2 Matrix:     <br>  |a(1,1) a(1,2)|      <br>
   *                                        |a(2,1) a(2,2)|      <br>
   */
  void display();
  //@}

  /*! @name ---------------------------------------- Number of Rows*/
  //@{
  /*!getNRows : give the number of Rows of the Matrix             <br>
   */
  int getNRows();
```

```
//@}

/*! @name ------------------------------------ Number of Columns*/
//@{
/*!getNColumns : give the number of Columns of the Matrix       <br>
 */
int getNColumns();
//@}

/*! @name ------------------------Putting an element in the Matrix*/
//@{
/*!setElement : Put an element in the matrix                    <br>
 *
 * @param value   (Double) : the value to put in the matrix     <br>
 * @param row     (int)    : row location                       <br>
 * @param column  (int)    : column location                    <br>
 *
 */
void setElement(double, int, int);
//@}

/*! @name --------------------- Getting an element from a location*/
//@{
/*!getElement : getting an element from the Matrix              <br>
 *
 * @param row       (int)    : row location                     <br>
 * @param column    (int)    : column location                  <br>
 * @return element  (double) : the value from the location      <br>
 *
 */
double getElement(int, int);
//@}

/*! @name ------------Copy all the element of a matrix in an other*/
//@{
/*!copyMatrix : Matrix copy                                     <br>
 *
 * @param Matrix (Matrix *) : Data Matrix (Matrix where are the
 *                            values to copy)                   <br>
 *
 * \note NO MEMORY ALLOCATION HERE
```

```
 */
void copyMatrix(Matrix*);
//@}

/*! @name ------------------------------ Determinant Calculation*/
//@{
/*! det : Determinant of the Matrix calculation                 <br>
 *
 * @return determinant (double) : Determinant of the Matrix     <br>
 * \note Principle :
 *                   - First column development                 <br>
 *                   - Recursive function                       <br>
 */
double det();
//@}

/*! @name ---------------------------------- Matrix Transposition*/
//@{
/*! transpose : transpose the matrix
 *
 *  @return T_Matrix (Matrix *): a pointer to the transposed matrix
 *
 *  \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *transpose();
//@}

/*! @name ------------------------------------- Matrix Summation*/
//@{
/*! sum : summation of two matrix
 *
 *  @param  M  (Matrix *): a pointer to the matrix to sum
 *  @return SM (Matrix *): a pointer to summed matrix           <br>
 *
 * \note Be 'A': the current matrix, Be 'B' the parameter Matrix<br>
 *       To calculate M=A+B :                                   <br>
 * \code Matrix *A = new Matrix (...) //Matrix A Definition
 *       Matrix *B = new Matrix (...) //Matrix B Definition
 *       Matrix *M;
 *       M=A->sum(B);
 *       ...
```

```
 *       delete A;
 *       delete B;
 *       delete M
 * \endcode
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *sum(Matrix*);
//@}

/*! @name ---------------------------------- Matrix Multiplication*/
//@{
/*!mult : multiplication of two matrix
 *
 * @param  M  (Matrix *): a pointer to the matrix to multiply
 * @return MM (Matrix *): a pointer to the multiplied matrix    <br>
 *
 * \note Be 'A': the current matrix, Be 'B' the parameter Matrix<br>
 *       To calculate M=A*B :                                   <br>
 * \code Matrix *A = new Matrix (...) //Matrix A Definition
 *       Matrix *B = new Matrix (...) //Matrix B Definition
 *       Matrix *M;
 *       M=A->mult(B);
 *       ...
 *       delete A;
 *       delete B;
 *       delete M;
 * \endcode
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *mult(Matrix*);
//@}

/*! @name ---------------------------------------- Matrix Scaling*/
//@{
/*! scale : Matrix scaling
 *
 *  @param  factor (double)    : scaling factor (a scalar)
 *  @return SM     (Matrix *) : scaled Matrix
 *
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
```

```
 Matrix *scale(double);
 //@}

 /*! @name ---------------------------------------- Matrix Power*/
 //@{
 /*! power : stand up a Matrix to a certain power
  *  (Not a good english !!! Stand up ?)
  * \note A->power(n) = A^n
  *  @param  pow (int)      : Power
  *  @return PM  (Matrix *) : Matrix stood up to the power
  *
  * \note MEMORY IS ALLOCATED for the returned matrix
  *
  */
 Matrix *power(int);
 //@}

 /*! @name -------------------------- Getting rows of the Matrix*/
 //@{
 /*! getRows : return a part of the matrix
  *
  * @param begin (int) : the first row we need
  * @param end   (int) : the last row  we need
  *
  * \note Example :                                          <br>
  *       Be A a matrix :                                   <br>
  *          | 11 22 33 44 55 66 |                          <br>
  *          | 71 82 94 10 11 12 |                          <br>
  *          | 13 14 15 16 17 18 |=A                       <br>
  *          | 19 20 21 22 23 24 |                          <br>
  *          | 25 26 27 28 29 30 |                       <br><br>
  *       Do M=A->getRows(2,4)                             <br><br>
  *          | 71 82 94 10 11 12 |                          <br>
  *          | 13 14 15 16 17 18 |=M                       <br>
  *          | 19 20 21 22 23 24 |                          <br>
  *
  * \note MEMORY IS ALLOCATED for the returned matrix
  */
 Matrix *getRows(int, int);
//@}
```

```
/*! @name ------------------------- Getting columns of the Matrix*/
//@{
/*! getRows : return a part of the matrix
 *
 * @param begin (int) : the first column we need
 * @param end   (int) : the last column  we need
 *
 * \note Example :                                              <br>
 *       Be A a matrix :                                       <br>
 *          | 11 22 33 44 55 66 |                              <br>
 *          | 71 82 94 10 11 12 |                              <br>
 *          | 13 14 15 16 17 18 |=A                            <br>
 *          | 19 20 21 22 23 24 |                              <br>
 *          | 25 26 27 28 29 30 |                            <br><br>
 *       Do M=A->getColumns(2,4)                               <br><br>
 *          | 22 33 44 |                                         <br>
 *          | 82 94 10 |                                         <br>
 *          | 14 15 16 |=M                                       <br>
 *          | 20 21 22 |                                         <br>
 *          | 26 27 28 |                                      <br><br>
 *
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *getColumns(int, int);
 //@}

/*! @name ---------------------------------Matrix Concatenation*/
//@{
/*! cat : concatenate two matrix
 *
 * @param M  (Matrix *) : a pointer to the matrix to concatenate
 * @param rc (char)     : a character to know if we want to
 *                        concatenate columns or rows
 * \note if rc ='c' or 'C' : columns concatenation            <br>
 *       if rc ='r' or 'R' : rows concatenation
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *cat(Matrix*,char);
//@}

/*! @name ---------------------------Delete a row of the matrix*/
```

```
//@{
/*! delRow : delete the specified row
 *
 * @param  r (int)      : specify the row to delete
 * @return M (Matrix *) : a pointer to the matrix without the row
 *
 * \note MEMORY IS ALLOCATED for the returned matrix
 *
 */
Matrix *delRow(int);
 //@}

 /*! @name ----------------------Delete a column of the matrix*/
//@{
/*! delCol : delete the specified column
 *
 * @param  c (int)      : specify the column to delete
 * @return M (Matrix *) : a pointer to the matrix without the
 *                        column
 *
 * \note MEMORY IS ALLOCATED for the returned matrix
 *
 */
Matrix *delCol(int);
 //@}

/*! @name ---------------------- Generate the co-factor matrix*/
//@{
/*! coFact : generate the co-factor matrix
 *
 * \note Be A<r/s> the matrix (n-1)*(n-1) obtained
 *       by deleting the rth row and
 *       the sth column                                    <br>
 *       Be A' the co-factor Matrix                        <br>
 *       A'(r,s) = (-1)^(r+s)*det(A<r/s>)                <br><br>
 * \note MEMORY IS ALLOCATED for the returned matrix
 */
Matrix *coFact();
//@}

/*! @name ------------------------ Generate the inverse matrix*/
```

```
  //@{
  /*! inverse : generate the inverse  matrix
   *
   * \note A^(-1) = (1/det(A))*transpose(A')
   *      (with A' the co-factor matrix)    <br><br>
   *
   * \note MEMORY IS ALLOCATED for the returned matrix
   */
  Matrix *inverse();
 //@}
  double* convert();
  void    changeValues(double []);
  double **getArray();
private :
  double **mat;
  int   Ncol;
  int   Nrow;
};

#endif
```

Annexe K

Classe qld

```
#include <Matrix.h>

#ifndef QLD_H
#define QLD_H

//Optimisation des fonctions du type : 0.5*x'*H*x + G'*x
//Sous contrainte : A(J)*X + B(J)  = 0, J = 1..Me
//                  A(J)*X + B(J) >= 0, J =Me+1 .. M
//                  XL <= X <= XU

class qld
{
public :
  qld(Matrix *, Matrix *, Matrix *, Matrix *, Matrix *, Matrix *, int, int);
  ~qld();
  Matrix *getOptimalX();
  Matrix *getLagr();
  int getInfo();

private :
  Matrix *X;
  Matrix *Lagr;
  int Info;

};

#endif
```

Annexe L

Classe PERSON

```
#include <Matrix.h>

class Person
{
public :
  Person(Matrix *, int, int, int);
  ~Person();
  void calculatePosture(Matrix *,int *,int);
  Matrix * getPosture();
  Matrix * getParams();

private :
  int NTags, NQ, NParams;
  Matrix *Q;
  double *QArray;
  Matrix *qMin;
  Matrix *qMax;
  Matrix *tagsCoord;
  Matrix *Jacobian;
  Matrix *Gradient;
  Matrix *Hessian;
  void calculateJacobian();
  void calculateHessian();
  void calculateGradient(Matrix *,int *,int);
  Matrix * badTag(Matrix *,int *,int);
};
```

Bibliographie

[CLT] Jian L.Zhou Craig Laurence and André L. Tits. User's guide for cfsqp version 2.5. Technical report, Electrical Engineering Department and Institute for Systems Research, University of Maryland, College Park.

[Dig] Northern Digital. Optotrak : System guide. Documentation technique : Optotrak, Northern Digital.

[KK86] W. Khalil and J.F. Kleinfinger. *A new geometric notation for open and closed loop robots*. IEEE International Conference on Robotics & Automation, 1986.

[MFD] Cani-Gascuel Marie-Paule Multon Franck, France Laure and Gilles Debunne. Computer animation of human walking : a survey. IRISA, BIP/INRIA, iMAGIS-GRAVIR/IMAG.

[PEFT] Hans Bruun Nielsen Poul Erik Frandsen, Kristian Jonasson and Ole Tingleff.

[Sar00] P. Sardain. Paramètres de BIP2000. Rapport technique, Laboratoire de Mécanique des Solides (LMS) de Poitier, 2000.

[SM] Bruno Arnaldi Stéphane Ménardais, Franck Multon. A global framework for motion capture. Technical report, INRIA.

[Stu] Peter Sturm. Outils mathématiques - optimisation. INRIA Rhône-Alpes.

[TMT96] R.Boulic T. Molet and D. Thalman. A real time anatomical converter for human motion capture. *Eurographics Workshop on Computer Animation and Simulation*, pages 79–94, September 1996.

[Wer94] J Werneke. *The Inventor Mentor*. Open Inventor Architecture Group, 1994.

[Wie00] P.B. Wieber. *Modélisation et commande d'un robot marcheur anthropomorphe*. Thèse de doctorat, Ecole des Mines de Paris, 2000.