# INTERFACE POUR LE PILOTAGE ET L'ANALYSE DES ROBOTS BASÉE SUR UN GÉNÉRATEUR DE CINÉMATIQUES

THÈSE N$^{O}$ 1897 (1998)

PRÉSENTÉE AU DÉPARTEMENT DE MICROTECHNIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Lorenzo FLÜCKIGER

Ingénieur en microtechnique diplômé EPF
originaire de Genève (GE)

acceptée sur proposition du jury:

Prof. R. Clavel, directeur de thèse
Dr Ch. Baur, rapporteur
Dr B. Hine, rapporteur
Prof. H. Hügli, rapporteur
Prof. C. Nicollier, rapporteur
Dr F. Sternheim, rapporteur

Lausanne, EPFL
1998

*Novembre 1998*

*à la mémoire de Sita*

# RÉSUMÉ

La robotique s'appuie sur des outils informatiques aussi bien dans les phases de conception, de programmation, de modélisation (géométrique, cinématique, dynamique, contraintes, etc.) que de simulation. Ces logiciels qui sont de précieuses, et déjà souvent indispensables, aides à la maîtrise globale d'un système robotique, sont à ce jour presque exclusivement réservés à des spécialistes. En particulier, les outils permettant d'aborder les aspects de simulation ou de programmation hors-ligne de robots restent d'utilisation complexes.

La modélisation d'un nouveau robot nécessite d'habitude un investissement en temps important. En outre, la définition de tâches n'est en général pas envisageable pour un non spécialiste. Le projet CINEGEN qui fait l'objet de cette thèse se propose de combler ces lacunes.

CINEGEN définit un nouvel outil, simple d'utilisation et rapide à mettre en oeuvre, pour la simulation cinématique de robots manipulateurs à structure quelconque dans un environnement de type réalité virtuelle. La description de la structure du robot est facilement maîtrisable. Quant à la simulation du robot (en vue de son étude ou de la programmation de tâches), elle s'effectue de manière interactive en temps réel, ce qui la rend accessible au néophyte.

La mise en oeuvre rapide d'une nouvelle simulation s'effectue grâce à une description, dans un fichier texte, des paramètres géométriques élémentaires de la structure du robot. Ce fichier décrit également les différentes contraintes que la structure comporte; à savoir d'une part les boucles cinématiques et d'autre part le contrôle d'éléments du robot par des périphériques de saisie. Ce fichier est analysé par le programme qui construit de manière automatique un modèle numérique de la cinématique du robot en vue de satisfaire toutes les contraintes. En fonction des consignes données interactivement par l'utilisateur, le "moteur" cinématique calcule les mouvements que le robot doit effectuer tout en respectant les contraintes externes. La méthode de résolution des contraintes permet de faire évoluer le robot suivant sa cinématique directe (contrôle de chaque articulation indépendamment) ou inverse (contrôle de l'organe terminal), ceci de manière transparente pour l'utilisateur.

La définition de tâches et l'étude intuitive du comportement du robot se basent sur l'utilisation d'un environnement virtuel qui permet des interactions fortes et directes entre l'utilisateur et le modèle à piloter. Ceci nécessite trois facteurs fondamentaux: une visualisation d'un monde tridimensionnel, des périphériques de saisie adaptés et une réponse du système en temps réel. La visualisation d'un espace tridimensionnel permet à l'utilisateur

d’appréhender directement le robot et le monde dans lequel il évolue sans passer par une représentation symbolique. Le développement d’un périphérique dédié au contrôle de robot, incluant un retour de force, facilite la définition de consignes et complète les autres périphériques commerciaux utilisés pour la saisie. Le temps réel, nécessitant un traitement des données et rafraîchissement de l’image à une fréquence de 25 Hz minimum, est obtenu par l’utilisation d’une station graphique, d’outils numériques performants et surtout d’un moteur de résolution de contrainte adapté.

Le projet CINEGEN répond donc à deux besoins:

- le prototypage rapide de robots manipulateurs à structure quelconque en vue de leur analyse (cinématique, comportement, etc.),
- la création aisée d’une interface intuitive pour la définition de tâches d’un nouveau robot (environnement de téléopération, etc.).

Ces buts sont atteints par la mise en oeuvre d’une interface homme-robot interactive réalisée grâce à un nouveau fichier de description des structures de robots, à un moteur de résolution de contraintes fonctionnant en temps réel ainsi qu’à des outils d’entrée et sortie naturels pour l’utilisateur.

# SUMMARY

In robotics we depend on software tools during design, modeling, programming and testing. These tools are essentials, often indispensable aids for developing and operating sophisticated robotic systems. At the same time, these tools are complex and usually too difficult to be used by non specialists. For example tools used for simulation or off-line programming require significant knowledge and skill.

The goal of this thesis is to provide novice users with an intuitive tool (CINEGEN) for designing, studying and controlling robot manipulators without programming. In particular the tool addresses two main problems: 1) modeling a new robot requires an significant amount of time; 2) robot tasks (motion, actions, etc.) are generally difficult for novice users to specify.

CINEGEN is a novel tool for kinematic simulation of robot manipulators in a virtual environment. It is easy to use and is capable of handling generic kinematic structures. With CINEGEN the description of robots is easy to perform and enables rapid prototyping. Additionally, CINEGEN's capability for real-time interactive simulation allows novice users to quickly specify and evaluate robot tasks.

A new simulation can be created very rapidly by describing the robot in a simple text based configuration file. In this file, robots are defined by the properties of each link and their relationships. Robots are defined as a tree structure from the base to the end-effector. For robots with kinematic loops, each loop is represented with two open sub-chains which are closed using a simple constraint. This same type of constraint is used to define which part of the robot must follow movements generated by input devices to the simulation. Once defined, this file is parsed by CINEGEN which automatically constructs the robot structure and its numerical kinematic model to satisfy all the constraints. Then the kinematic solver computes the robot movements regarding the user inputs and the internal constraints. This allows the user to interactively control the robot in two modes: direct kinematics (independent control of each joint) or inverse kinematics (control of the end effector). This constraint solver scheme provides the user with a unified interface to control robots without requiring thought about direct or inverse kinematics.

The user interacts with the model of the robot using a virtual reality based interface. This interface gives the user a direct and intuitive means to study a robot's behavior. The virtual reality based interface implies three fundamentals needs: a visualization of 3-dimensional world, appropriate input devices and real-time simulation. The visualization of the robot in a three dimensional space allows the user to understand the robot and the world in which it

moves without any symbolic representation. The design of a new haptic input device extends the use of commercial devices employed, making it easier to generate control inputs as well as to “feel” the robot response. Real-time performance (refresh at more than 25Hz) of the complete simulation (graphics as well as kinematics) is obtained via efficient numerical tools and a constraint solver dedicated to robot kinematics.

In short, the project developed in this thesis answer to two principal needs:

- rapid prototyping and analysis of robot manipulators with general kinematic structure,
- an intuitive interface for teleoperation (task definition) of new robots without programming.

# REMERCIEMENTS

Je remercie le Docteur Charles Baur pour la confiance totale qu'il m'a accordée ainsi que pour les rôles clefs qu'il a tenu dans ce travail de recherche: initiateur du projet, conseiller durant les 5 années passées dans son équipe, et enfin, membre du jury de thèse dont les critiques m'ont été précieuses.

Je remercie le Professeur Reymond Clavel pour m'avoir accueilli au sein du Département de Microtechnique et pour avoir accepté la responsabilité de directeur de thèse.

Je remercie vivement les membres du jury de thèse, pour l'intérêt qu'ils ont montré face à mon travail ainsi que pour l'éclairage personnel qu'ils ont apporté grâce à leurs compétences dans leur domaine scientifique respectif: le Professeur Heinz Hügli, le Professeur Claude Nicollier et le Docteur Federico Sternheim. Le Docteur Butler Hine mérite un remerciement particulier pour son accueil lorsqu'il était directeur du groupe IMG de la NASA et pour son effort de lecture dans une langue étrangère.

Dans la foulée, j'adresse un remerciement outre-atlantique au Docteur Michael Sims qui m'a permis de participer aux activités de l'IMG, ainsi que ma reconnaissance à tous les membres de ce groupe pour leur aide et les échanges d'idées qui ont eu lieu durant les 5 mois que j'ai passé dans ce groupe: Cesar, Dan, Deanne, Eric, Hans, Kurt, Laurent et Maria.

Du côté Suisse je tiens à remercier les collaborateurs, étudiants et amis de l'ex-IMT et de l'EPFL en général qui ont touché de près ou de loin, au niveau technique ou sur le plan amical, à ce travail de thèse. Plus spécialement ma gratitude va à Denis, David, Didier, Jean-Jacques, Jimmy, Laurent, Olivier, Peter, Pere, Roland et Willy ainsi qu'aux membres du Groupe VRAI.

Special thanks to Terry who is always ready to help you and to fix any problems, and from whom I have learned so many things.

Enfin, je remercie ma famille qui m'a soutenu pendant cette période parfois chaotante, en particulier mes parents qui ont contribué au travail ingrat mais ô combien nécessaire de correction, Simon et l'aide qu'il m'a accordé et Anna pour ses attentions.

Finalement je remercie ma tendre Cécile qui a partagé avec moi (et de manière synchrone!) toutes les périodes de ma thèse pendant lesquelles le moral oscille souvent comme les variations du baromètre dans les caraïbes.

*Ce travail a été financé par le Fonds National Suisse de la Recherche Scientifique.*

# SOMMAIRE

CHAPITRE

# 1

# INTRODUCTION

L'homme conçoit et construit des robots pour l'assister ou le remplacer dans un large éventail de travaux. Ces développements conduisent à des machines hautement perfectionnées qui peuvent réaliser des tâches de plus en plus sophistiquées. Mais trop souvent, la difficulté à maîtriser ces robots augmente avec la complexité du système.

Le but de cette recherche consiste à améliorer l'interface homme-machine dans le cas particulier des robots manipulateurs. La synthèse originale de deux systèmes développés dans ce travail a permis de concrétiser cet objectif. D'une part un nouveau type d'interface dédié aux robots, basé sur les environnements virtuels, a été élaboré. D'autre part, un moteur de résolution de contraintes pour les structures articulées a été créé; il libère l'opérateur du problème mathématique sous-jacent au contrôle des robots. Ces concepts ont aboutit à la réalisation d'un programme permettant de concevoir d'étudier, d'évaluer et de piloter des robots manipulateurs de manière intuitive.

## 1.1 Problématique

Afin de bien cerner les contributions de ce travail de thèse, nous allons d'abord situer le sujet dans le contexte de la robotique. Cela mettra en évidence la nécessité d'une interface homme-robot et introduira le problème de la cinématique des robots.

### 1.1.1 Situation du projet

Ce travail se situe dans le cadre des recherches portant sur la robotique menées par le Département de Microtechnique de l'EPFL (Ecole Polytechnique Fédérale de Lausanne). Plus précisément, il s'inscrit dans les développements du Groupe VRAI (Virtual Reality and Active Interfaces) et de ses collaborations avec l'IMG (Intelligent Mechanisms Group) du NASA Ames Research Center (Californie). Ces deux groupes de recherche conçoivent de nouvelles interfaces pour piloter des systèmes robotisés.

Le terme *robotique* désigne la science des robots en tant qu'entités. Toutefois dans le domaine industriel la "robotique" regroupe couramment les techniques et les connaissances permettant l'*automatisation*. Or l'automatisation industrielle peut (souvent) être réalisée sans robot, grâce à des machines spécialisées, accompagnées d'une collection de périphériques. D'où l'intérêt de préciser la terminologie en proposant l'hypothèse qui suit.

L'automatisation regroupe deux domaines interconnectés, d'une part la *productique* concernant "l'art de produire des biens manufacturés en utilisant les technologies les plus récentes" [Coiffet 92], et d'autre part la robotique s'occupant des robots proprement dits. La robotique se développe actuellement en deux grandes branches.

- La robotique manufacturière: utilisée comme un outil de la productique, destinée à la production de biens. Ce domaine de la robotique regroupe tous les robots industriels.
- La robotique non-manufacturière: utilisée comme outil pour réaliser des tâches difficiles ou impossibles pour l'humain seul. Ce domaine regroupe essentiellement les robots de télémanipulation, les robots d'exploration, les robots mobiles et les simulateurs.

La Figure 1-1 positionne donc la robotique manufacturière comme outil de la productique alors que la robotique non-manufacturière est surtout utilisée dans le domaine des services. Par services nous entendons l'emploi de robots dans des tâches qui ne servent pas directement à la production de biens.

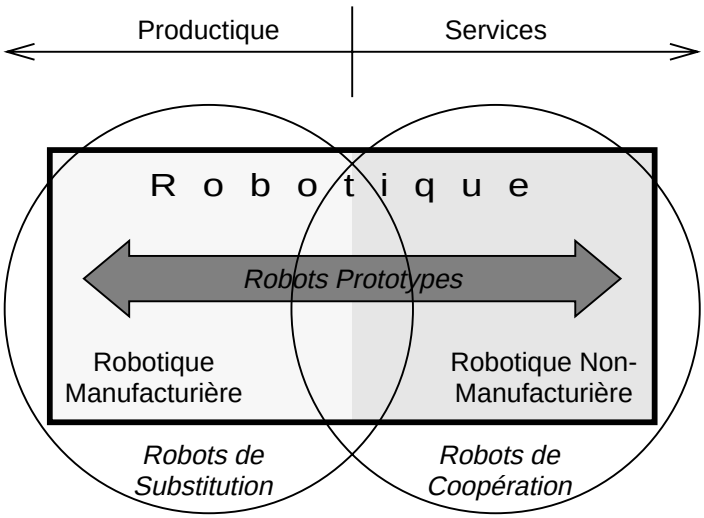



**Figure 1-1:** *Productique, robotique et robots*†

†. Les limites entre ces secteurs et classes de robot ne sont pas définies de manière stricte, et il existe de nombreuses (et souhaitables) interactions entre ces différents domaines.

Il est intéressant de mettre en correspondance ces deux domaines avec les types de robots existants: aux domaines de la robotique manufacturière et non-manufacturière correspondent respectivement les robots de substitution et les robots de coopération. Ces catégories de robots émanent des deux grandes motivations qui ont poussé l'homme à réaliser ce type de machine:

- Remplacer l'homme dans des tâches répétitives par une machine automatique et "intelligente[1]" que l'on programme à l'avance suivant le travail à réaliser: les *robots de substitution*.
- Aider l'homme dans des tâches qu'il ne peut réaliser lui-même (opération dans des milieux dangereux, exploration de milieux hostiles ou lointains, manipulations à l'échelle microscopique): les *robots de coopération*.

La réalisation de tout robot commence par la conception de prototypes. Cet ensemble de robots sera classé dans une catégorie dénommée "robots prototypes". Lorsqu'ils sont destinés à la robotique manufacturière, leur durée d'existence est limitée (aboutissement sur une version commerciale) alors que dans le monde de la recherche académique ils sont en évolution permanente. Toutefois, quelle que soit leur destinée, ces robots représentent un sous-ensemble particulier de la robotique, comme le représente la Figure 1-1.

C'est essentiellement à cette dernière catégorie de *robots prototypes* que s'adresse la présente recherche.

### 1.1.2 Interface Homme-Robot

Les robots de substitution employés dans l'industrie sont destinés à des travaux fixes et répétitifs dans un environnement structuré connu a priori. Les méthodes de programmation classiques ou de *CFAO* sont adaptées à la définition de ce genre de tâches (cf. Section 2.1.1). Les robots de coopération destinés à la manipulation d'objets dans des environnements non structurés se trouvent face à des situations nouvelles à chaque opération. Ces robots sont le plus souvent commandés de manière directe par téléopération (cf. Section 2.1.2).

Hors de ces deux précédentes catégories de robots pour lesquelles des techniques éprouvées existent, les problèmes suivants subsistent:

- La définition d'une mission pour un robot manipulateur n'est pas accessible à un non spécialiste avec les outils classiques. Or l'expérience montre que le spécialiste du domaine dans lequel évolue le robot (chimiste, géologue, océanographe, etc.) est souvent plus à même de définir les tâches à effectuer que le constructeur ou programmeur du robot.
- Des outils de simulation très performants existent pour la modélisation des robots. Toutefois ces techniques nécessitent un investissement en temps important et requièrent une personne spécialiste du logiciel utilisé.

Le projet «CINEGEN» (pour "Cinématique Générale") faisant l'objet de cette thèse se propose de combler ces lacunes en fournissant un outil qui permette d'une part d'étudier rapidement le comportement de nouvelles structures de robots et d'autre part de définir des tâches de manière intuitive

1. Le concept de machine intelligente était en tout cas l'espoir premier lors de l'apparition des robots. On se rend compte maintenant que l'on reste encore très loin du robot même un tout petit peu "intelligent".

pour des robots existants. Par comportement nous englobons la cinématique des mouvements, l'interaction avec l'environnement ou encore l'évolution de paramètres divers en fonction de certaines tâches.

Pour réaliser ces buts, il faut procurant à l'utilisateur un outil qui lui permette d'interagir de manière naturelle avec le robot manipulateur. Ceci implique plusieurs critères à respecter:

- la présentation des informations (robots, mouvements, objets, etc.) doit être facilement interprétable par un non spécialiste,
- des moyens de définition d'une tâche (les trajectoires du robot) doivent utiliser des dispositifs intuitifs et simples d'utilisation,
- la réaction du système doit se faire sans délai (temps réel) afin de permettre une évaluation directe des consignes générées.

Le projet «CINEGEN» propose de satisfaire ces critères par l'exploitation des propriétés des environnements virtuels. En effet, grâce à de tels environnements l'opérateur visualise le robot manipulé en trois dimensions et peut interagir avec lui en temps réel grâce à des périphériques adaptés.

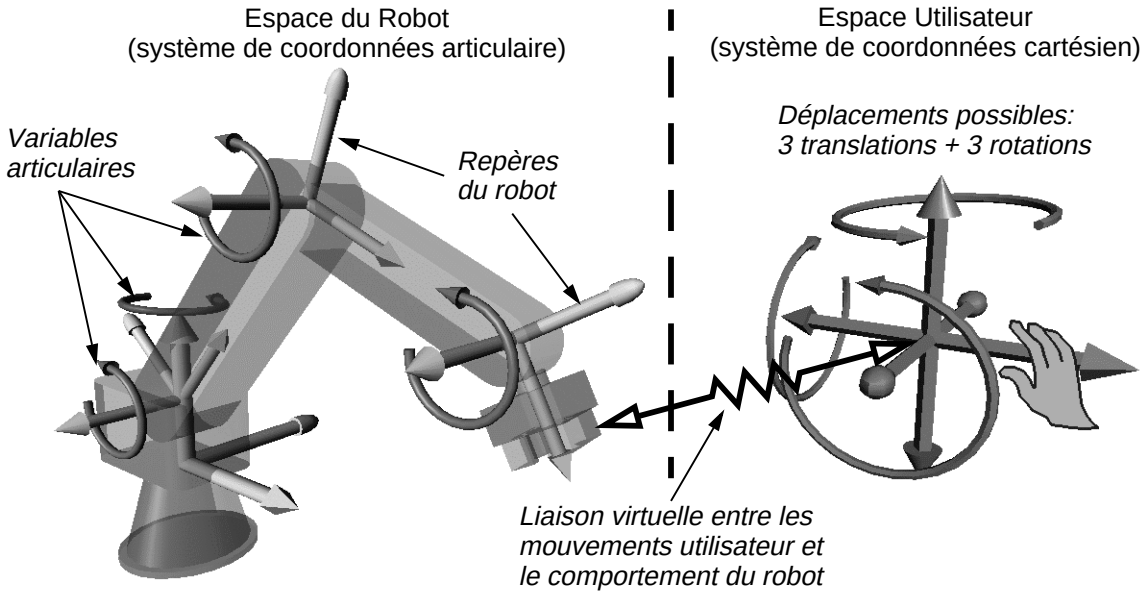



**Figure 1-2:** *Relations entre les différents systèmes de coordonnées*

Interagir avec le robot consiste dans ce contexte à saisir n'importe qu'elle partie du robot et à la déplacer (dans le sens général: translations et rotations) dans l'espace tridimensionnel pour effectuer le mouvement désiré. L'action doit sembler à l'utilisateur aussi simple qu'une opération "glisser-déplacer" (drag and drop) avec une souris classique. La Figure 1-2 illustre le mode de contrôle proposé et montre les différents systèmes de coordonnées impliqués dans le contrôle des robot.

### 1.1.3 Cinématique[2] générale

L'interface proposée, bien que simple au niveau conceptuel, engendre un problème important au niveau de la réalisation. Il s'agit de la cinématique inverse des robots manipulateurs: la transformation de consignes exprimées dans l'espace opérationnel (espace utilisateur) en consignes au niveau des moteurs du robot (variables articulaires). Ce problème étudié depuis les débuts de la robotique par des spécialistes de nombreux domaines (mathématique, ingénieurs, etc.) reste très complexe dans le cas général (cf. Section 2.2).

De plus, il est aussi intéressant de contrôler le manipulateur suivant sa cinématique directe, pour étudier comment se comporte la structure en fonction de consignes définies directement sur les moteurs. Or cette transformation est également difficile à résoudre dans le cas des manipulateurs parallèles. La Figure 1-3 résume les conversions à réaliser pour offrir à l'utilisateur un mode de contrôle intuitif des robots manipulateurs.

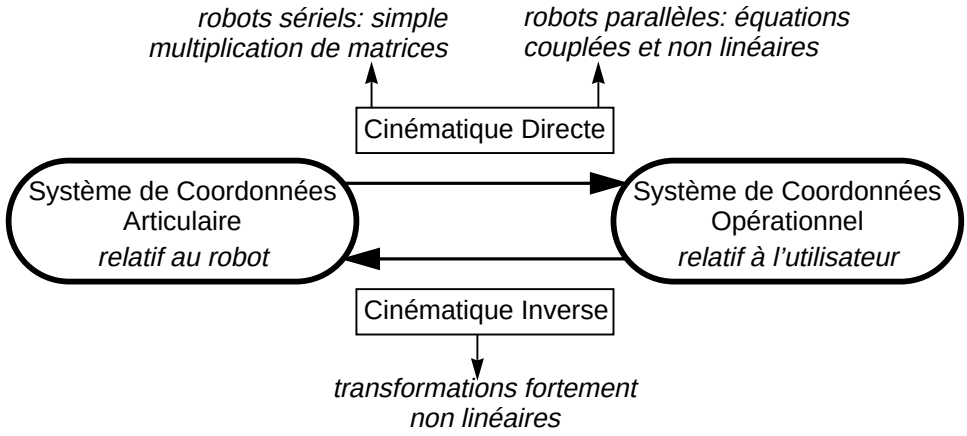



**Figure 1-3:** *Transformations impliquées dans le contrôle des robots*

En plus d'une interface permettant le contrôle de robots par un non spécialiste nous désirons aussi fournir un outil simple à mettre en oeuvre pour la définition de nouvelles structures de robots. Dans cette optique, les robots seront décrits dans un fichier texte standard suivant une notation élémentaire et explicite. Ce fichier sera lu par le programme, qui sans aucune recompilation construira automatiquement l'environnement virtuel contenant le robot et son modèle permettant de le piloter.

Finalement, dans le cadre du l'Institut des Systèmes Robotiques (ISR, affilié au Département de Microtechnique) spécialisé dans les robots parallèles, et pour assurer une généralité accrue au programme, nous désirons ne pas être limité aux cas des robots sériels (les plus courants dans le monde robotique, et caractérisés par des modèles plus simples). Par conséquent, des structures sérielles, des structures parallèles ou des structures hybrides devront pouvoir être définies et simulées suivant le même mode.

2. Par "cinématique des robots" nous exprimons le concept de "robot kinematics" en anglais: l'étude des positions, vitesses et accélération des éléments d'un robot. Une définition plus formelle sera donnée dans le Chapitre 3, "Concepts théoriques".

Ces deux derniers critères (modélisation rapide et robots à structure quelconque) ont conduit à concevoir un générateur automatique de cinématique générale qui fonctionne en synergie avec l'environnement virtuel.

### 1.1.4 Contributions

La situation du projet et sa définition met en évidence l'argument central de ce travail de thèse:

> fournir un outil d'interaction Homme-Robot
> intuitif et simple à mettre en oeuvre

Cet objectif est réalisé grâce aux contributions originales suivantes:

- une interface basée sur les environnements virtuels permettant à n'importe quel utilisateur de contrôler et étudier des robots manipulateurs,
- une description de robots à structure quelconque, simple, unifiée et réalisable par un non spécialiste,
- une méthode de résolution de la cinématique directe et inverse des robots à structure quelconque en temps réel grâce à un algorithme de résolution de contraintes spécifique,
- un nouveau concept de périphérique de saisie dédié au contrôle de robots avec retour d'efforts qui complète le système "entrée-sortie" de l'interface utilisateur,
- un programme évolutif et ouvert sur l'extérieur, permettant de partager des informations avec d'autres systèmes de visualisation, de modélisation ou de commande de robots,
- une simulation interactive fonctionnant en temps réel et fournissant des outils pour l'aide à l'appréciation de la tâche (virtualité augmentée).

Ces contributions ont abouti à un programme nommé "*VirtualRobot*" qui concrétise le projet «CINEGEN» et répond à un double besoin:

- le prototypage rapide de robots manipulateurs à structure quelconque en vue de leur étude par rapport à une tâche donnée,
- la mise en oeuvre rapide et aisée d'une interface complète pour un nouveau robot permettant une définition intuitive de tâches.

## 1.2 Conception

Cette section décrit la conception du système «CINEGEN». La présentation de l'architecture globale du programme permet de comprendre son mode de fonctionnement qui s'appuie sur le concept de réalité virtuelle et de la modélisation des robots.

### 1.2.1 Architecture

Le fonctionnement du programme *VirtualRobot* est schématisé dans la Figure 1-4.

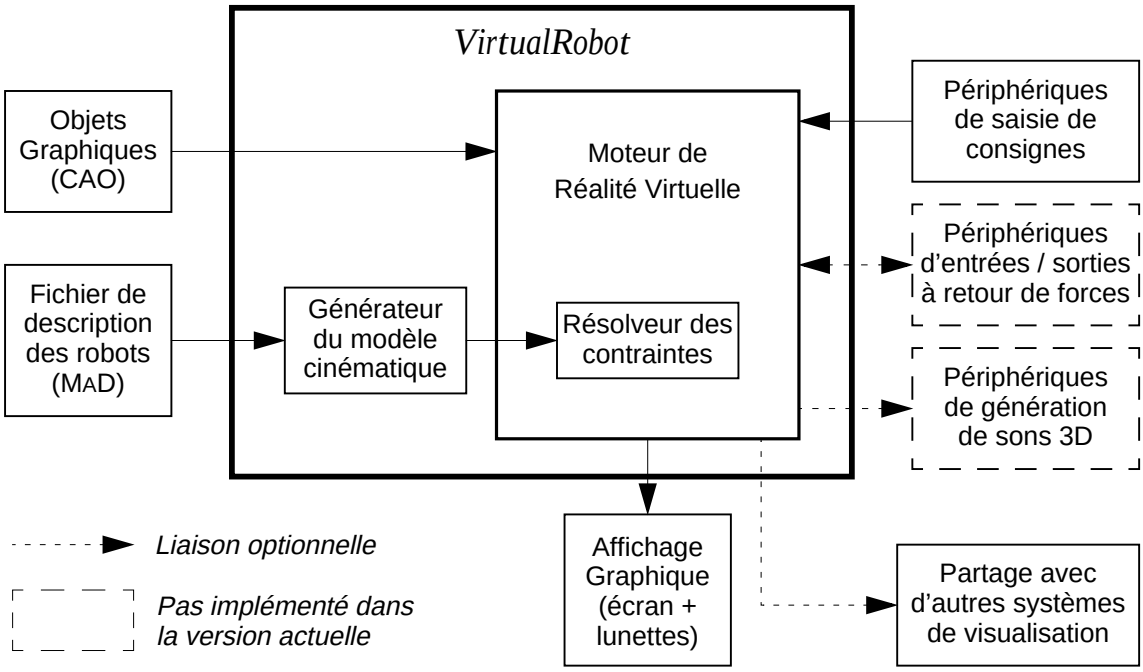



**Figure 1-4:** *Architecture globale du système «CINEGEN»*†

†. Il faut noter que la liaison avec des robots réel (génération d'ordres pour la commande) n'est pas représentée sur cette figure car cette fonctionnalité n'est pas traitée dans ce projet.

Le concepteur d'une nouvelle architecture de robot ou le créateur d'un environnement destiné au pilotage d'un robot spécifique décrit la structure cinématique du robot désiré dans un fichier texte. Ce fichier possède un format particulier – mis au point dans ce travail – dédié à la description des robots manipulateurs: le format MAD (Manipulator Description).

Ce fichier alimente le générateur automatique de cinématique générale qui construit le modèle du robot sous forme d'un ensemble de contraintes à résoudre. Celles-ci proviennent des relations structurelles du robot, ainsi que des consignes fournies par l'utilisateur pour manipuler certains éléments du robot.

Lors de la manipulation du robot par un opérateur dans l'environnement virtuel, le moteur de résolution de contraintes tente de satisfaire toutes les contraintes en permanence, ceci en fonction des actions de l'opérateur. Ce "résolveur" de contraintes est le cœur du système de réalité virtuelle qui constitue l'interface avec l'utilisateur.

L'interface basée sur les techniques de la réalité virtuelle procure le moyen d'effectuer les échanges bidirectionnels entre l'opérateur et le modèle du robot:

- elle traite les consignes que l'opérateur produit grâce à des périphériques de saisie,

- elle fournit les “résultats” des actions de l’opérateur sous forme d’affichage graphique tridimensionnel du comportement du robot.

La représentation graphique du robot est réalisée par l’affichage d’objets graphiques modélisant les différentes parties du robot. Ces objets graphiques peuvent être créés avec n’importe quel logiciel de dessin tridimensionnel assisté par ordinateur (DAO ou CAO). La position et l’orientation de ces objets sont en permanence actualisées par le résolveur de contraintes; il prend en compte les consignes de l’opérateur qui pilote le robot virtuel.

Il est possible d’améliorer la qualité de l’interface par l’ajout de périphériques supplémentaires à l’environnement virtuel. Les plus importants sont un système de pointage tridimensionnel à retour de forces (développé dans le cadre de ce projet, cf. Section 6.1.3) ainsi que la gestion d’un environnement sonore. Toutefois ces systèmes d’entrées/sorties ne sont pas intégrés dans la version actuelle de *VirtualRobot*.

Enfin, il est possible de partager la perception du comportement des robots avec d’autres systèmes de visualisation. Par exemple, une liaison a été réalisée avec le nouvel environnement de visualisation distribué à travers le réseau développé par l’IMG (Intelligent Mechanisms Group): Viz [wwwVIZ].

De la présentation du projet «CINEGEN» il découle que la maîtrise de deux domaines différents sont nécessaires à sa réalisation: les environnements virtuels et la modélisation des robots. Ces deux domaines sont donc brièvement exposés dans les sections suivantes.

### 1.2.2 Réalité Virtuelle

Il est difficile de proposer une définition de la Réalité Virtuelle qui satisfasse tous les spécialistes des différents domaines où elle est utilisée. Nous nous contentons de la définir dans le cadre de la robotique par rapport à ses fonctionnalités. On peut parler Réalité Virtuelle (Virtual Reality en anglais, abrégé couramment “VR”) dès que l’on procure à un opérateur un environnement non matériel dans lequel il peut évoluer et agir. Pour réaliser de tels environnements trois éléments au minimum sont nécessaires:

1) un système de visualisation représentant un monde tridimensionnel dynamique qui est synthétisé par ordinateur,

2) un ou plusieurs périphériques de saisie permettant à l’opérateur d’effectuer des actions dans ce monde virtuel,

3) une réponse – aux actions de l’opérateur – du système en temps réel (sans délais perceptibles pour les différents sens de l’homme).

Ces éléments permettent à l’utilisateur de se sentir immergé dans un environnement sans réalité matérielle et d’agir dans ce monde pour réaliser une tâche. La Réalité Virtuelle est donc caractérisée par les propriétés “Interactivité” et “Immersion”[3]. Un système de Réalité Virtuelle peut contenir

3. Burdea et Coiffet proposent d’ajouter la notion d’ “Imagination” au couple “Interactivité-Immersion” [Burdea 93]. Ils caractérisent alors la Réalité Virtuelle par les trois “I”.

beaucoup plus de fonctionnalités que les trois énoncées ci-dessus. En effet, dans le but d'immerger l'opérateur dans un monde qui lui paraisse réel, un système de Réalité Virtuelle cherche à utiliser tous les sens de l'humain. Nous ne détaillerons pas ces techniques qui sont présentées dans plusieurs ouvrages de référence: [Burdea 93], [Burdea 96], [Vince 95] [Pimentel 95].

Les caractéristiques "Interaction-Immersion" différencient clairement la Réalité Virtuelle de la simulation. La simulation tridimensionnelle proposée par de nombreux logiciels de modélisation de systèmes (mécaniques, robotiques, matériaux, etc.) est une présentation de l'information sous forme graphique en fonction de calculs pré-définis: l'utilisateur est spectateur d'une scène. Les techniques de simulations sont très utiles pour la mise en forme efficace de données en vue d'une analyse d'un système. Par contre, avec un système de Réalité Virtuelle l'utilisateur devient acteur puisqu'il est immergé dans ce monde et prend une part active au déroulement des actions[4].

Le challenge de la Réalité Virtuelle en ingénierie n'est pas de plonger et isoler l'humain dans un monde fantastique, mais de fournir à un utilisateur une nouvelle technique permettant principalement deux types de fonctionnalités:

- aide à la conception et à l'analyse grâce à une présentation interactive et intuitive du problème.
- téléopération et/ou téléprésence dans un environnement inaccessible en réalité (mission dans l'espace, milieu sous-marin, zones nucléaires, monde microscopique).

Nous classifions les systèmes de Réalité Virtuelle en deux catégories selon le degré d'immersion de l'utilisateur: immersion totale et immersion partielle.

#### 1.2.2.1 Immersion totale

L'immersion totale consiste à projeter l'utilisateur dans un environnement autre en le coupant totalement du monde réel. Le plus souvent on utilise un casque de visualisation (HMD en anglais pour "Head Mounted Display") fournissant à chaque oeil séparément une image du monde virtuel calculé en fonction de sa position respective dans l'espace ainsi que de l'orientation de la tête. Pour cela ces casques sont couplés à des systèmes de positionnement afin que l'utilisateur puisse regarder dans n'importe quelle direction et que le monde virtuel soit actualisé en conséquence. L'interaction avec l'environnement se fait grâce à des périphériques que l'utilisateur porte sur lui comme par exemple des gants tactiles positionnés dans l'espace ou des exosquelettes permettant des retours de forces.

Si de tels systèmes permettent un haut degré d'immersion, ils ne sont pas encore utilisables pour un travail réel de téléopération efficace et fiable. En effet il est pénible de devoir porter une "tenue" spécifique qui n'est pas encore assez ergonomique pour se faire oublier. De plus une fatigue due à la

4. Il est toutefois possible d'effectuer une simulation dans un monde virtuel. Par exemple après avoir défini activement une trajectoire, l'utilisateur peut la rejouer pour la vérifier. Il retourne alors temporairement au stade de spectateur d'une simulation.

mauvaise accoutumance à ces environnements artificiels ainsi que des malaises dus à la qualité médiocre des casques de visualisation peuvent apparaître [Kalawski 93].

#### 1.2.2.2 Immersion partielle

Un système d'immersion partielle se compose de périphériques plus simples que pour l'immersion totale; ils ne procurent pas un isolement complet du monde réel. Classiquement on utilise un système d'affichage sur un écran d'ordinateur et la vision de la profondeur est obtenue par le port de lunettes stéréoscopiques. L'utilisateur interagit avec le monde virtuel grâce à des périphériques qu'il saisit lorsqu'il en a besoin (SpaceBall, joysticks dédiés et autres).

Ce type d'environnement virtuel permet une immersion suffisante pour la plupart des applications. De plus l'opérateur peut "lâcher" les périphériques à tout moment pour se reposer. Enfin la personne immergée conserve un contact avec le monde réel ce qui est très utile pour recevoir d'autres informations que celles reçues par le canal virtuel (conseil d'un autre opérateur, vérification sur un autre écran, etc.)[5].

#### 1.2.2.3 Système employé

Le projet «CINEGEN», comme les précédents travaux [Piguet 94] [Flückiger 94] qui ont motivé cette recherche, utilise un système d'immersion partielle pour les raisons précitées.

Le système de visualisation est constitué d'un écran graphique haute résolution que l'opérateur peut regarder avec des lunettes stéréoscopiques pour obtenir la perception de profondeur. L'affichage stéréo est créé par l'alternance d'une vue calculée par une caméra "droite" et une caméra "gauche" (correspondant aux deux yeux de l'utilisateur). Cette alternance, effectuée à une fréquence supérieure à la fréquence de fusion de l'oeil (25Hz), est synchronisée avec des lunettes stéréo obstruant alternativement chaque verre. L'utilisateur perçoit donc (derrière les lunettes) avec chaque oeil l'image droite ou gauche correspondante: il dispose d'une réelle vision stéréoscopique.

Les systèmes de saisie utilisés sont des périphériques du marché qui permettent de mesurer des consignes dans l'espace (3 valeurs de position + 3 valeurs d'orientation). Actuellement deux périphériques sont utilisés: une "Souris 3D"[6] à ultrasons et un joystick 3D "Space Mouse"[7]. Tout périphérique de saisie employé en conjonction avec le programme *VirtualRobot* sera regroupé sous le terme "senseur". Un *senseur* est un

5. On peut ajouter aux avantages de l'immersion partielle qu'il est à tout moment possible de "s'échapper" du monde virtuel. Ceci est important car si l'opérateur est pris de malaise ou d'angoisse, il peut sortir du monde virtuel instantanément sans provoquer de dégâts éventuels dus à des mouvements inconsidérés.
6. Souris 3D de Logitech Inc., produit discontinué mais distribué par d'autres compagnies.
7. Space Mouse de Virtual Reality Technologies GmbH, licence achetée par Logitech Inc. qui commercialise le même produit sous le nom de "Magellan".

périphérique avec lequel l'opérateur peut fournir des consignes aux robots manipulés dans l'environnement virtuel.

Dans la suite de ce rapport, nous utiliserons les termes "environnement virtuel" ou "monde virtuel" plutôt que Réalité Virtuelle. En effet un utilisateur de *VirtualRobot* travaille dans un monde qui ne cherche pas à être une représentation fidèle de la réalité, mais un environnement qui propose des outils interactifs et intuitifs grâce à l'utilisation des techniques provenant de la Réalité Virtuelle.

### 1.2.3 Modélisation des robots

L'interface proposée par le programme *VirtualRobot* repose sur un générateur automatique de cinématique pour les robots à structure quelconque afin de libérer l'opérateur de toute programmation compliquée. Or la résolution des modèles cinématiques inverse et direct des robots est très compliquée dans le cas général.

Les méthodes existantes pour traiter le problème de la cinématique des robots sont nombreuses (cf. Section 2.2), mais le cas général reste insoluble de manière algébrique. Plusieurs programmes proposent des outils pour résoudre les modèles géométriques complets de structures particulières. De même les outils pour la modélisation du comportement dynamique des robots sont de plus en plus nombreux (cf. Section 2.3). Tous ces outils sont des aides précieuses pour la conception des robots et des systèmes mécaniques de manière générale.

Le projet «CINEGEN» est orienté sur la partie interface interactive avec les robots. Le programme *VirtualRobot* calcule donc uniquement un modèle cinématique des robots (dynamique non prise en compte). Ce type de résolution est tout à fait satisfaisant vis-à-vis du type d'interface proposé (cf. Section 3.1). De plus la méthode proposée se base sur une résolution originale dédiée au cas des robots, ce qui permet un comportement en temps réel en fonction de multiples consignes générées par l'utilisateur.

«CINEGEN» effectue une résolution des modèles géométriques direct et inverse de robots sériels, parallèles ou hybrides en travaillant dans l'espace des vitesses. Ceci permet une linéarisation des modèles et une résolution de contraintes interactive. La méthode proposée implique de connaître la position initiale du robot. Elle est décrite en même temps que la structure du robot dans le fichier MAD.

## 1.3 Organisation du document

Le Chapitre 2, "Domaines impliqués et état de l'art" recense les différents techniques et produits pour la programmation des robots, leur modélisation et leur simulation. Il dégage les méthodes utilisables dans le projet «CINEGEN» ainsi que les différences de *VirtualRobot* avec les logiciels existants.

Le Chapitre 3, "Concepts théoriques" présente les bases de la modélisation des robots utilisée dans ce projet (de la Section 3.1 à la

Section 3.3). L'explication de ces principes connus en robotique permet de comprendre l'approche proposée pour la génération de la cinématique de robots à structure quelconque de manière automatique. Cette méthode, décrite dans la Section 3.4, est basée sur une description des chaînes articulées par un graphe et sur une résolution des contraintes apparaissant dans ce graphe.

Le Chapitre 4, "Fichier de description MAD" décrit en détail la structure et le format du fichier de description des robots mis au point dans ce projet. Pour illustrer la facilité de mise en oeuvre d'un environnement virtuel permettant de piloter un nouveau robot grâce au programme *VirtualRobot*, un exemple de structure articulée est traité de manière complète, de sa description géométrique à l'écriture du fichier MAD la représentant.

Le Chapitre 5, "Implémentation de l'outil" explique la conception et la réalisation du programme *VirtualRobot*. Pour cela le chapitre commence par déterminer la plateforme de développement nécessaire pour élaborer une telle application (Section 5.1 et Section 5.2). Ensuite la présentation de l'architecture du programme dans la Section 5.3 permet de mettre en évidence comment *VirtualRobot* s'appuie sur des bibliothèques externes existantes. Ces différents outils employés sont présentés dans la Section 5.4. Finalement la Section 5.5 détaille l'implémentation du programme *VirtualRobot*.

Le Chapitre 6, "Fonctionnalités et étude de cas" présente les résultats obtenus avec le programme *VirtualRobot*. Il montre les différentes méthodes permettant de définir une tâche, dont un système mécanique à six degrés de liberté à retour d'efforts, et les moyens d'évaluer cette tâche. Ce chapitre se termine par une étude de différentes structures simulées afin de mettre en évidence les fonctionnalités de *VirtualRobot*.

La conclusion clôt ce rapport en introduisant les développements futurs à envisager. Une bibliographie regroupant les articles et ouvrages cités suit la conclusion. Certaines références citées ne sont pas publiées sur papier, elles sont regroupées dans la liste d'adresses web fournie après la bibliographie.

### *Conventions utilisées dans ce document*

Des notes en marge du document sont utilisées pour procurer une information supplémentaire au lecteur. Ce type d'information, contrairement aux notes en bas de page qui sont directement reliées au texte, n'est fourni qu'à titre d'illustration et peuvent être sautées.

Certains termes utilisés dans ce document possèdent une signification particulière dans le cadre de ce projet. Ils sont regroupés dans un glossaire page -145 (situé avant les annexes). La première fois qu'ils apparaissent dans le texte, ces termes – comme *syntaxeur* ou *robotique* – sont mis en évidence par une convention typographique.

La notation pour identifier les "objets" d'un programme est identique à celle utilisée couramment dans les ouvrages d'informatique: fonte courrier. Les classes commencent par une majuscule (comme `Senseurs`) alors qu'une instance de cette classe (un objet) commence par une minuscule et reste au singulier (comme `robot`).

CHAPITRE

# 2

# DOMAINES IMPLIQUÉS ET ÉTAT DE L'ART

Le projet «CINEGEN» regroupe plusieurs domaines de la robotique traités usuellement de manière indépendante: la programmation des robots, la résolution de la cinématique des structures articulées et la simulation de systèmes robotisés. Ce chapitre expose brièvement l'état des recherches dans ces trois domaines afin de mieux cerner les problématiques sous-jacentes au projet «CINEGEN» et l'originalité du traitement proposé.

## 2.1 Programmation des robots

L'homme a conçu les premiers robots dans un style le plus anthropomorphique possible puisqu'ils étaient alors destinés à remplacer un geste humain. Le fait de devoir "imiter" les mouvements d'un opérateur humain, ainsi que les capacités de l'électronique à cette époque (vers 1960), impliquait un système de commande relativement sommaire. Il a été possible dès le début de la robotique de distinguer clairement deux classes de robots: les robots de substitution, et les robots de coopération (cf. Section 1-1). Dans le premier cas on "apprenait" une trajectoire au robot qui pouvait ensuite la reproduire à l'infini, sans erreur (en théorie), mais aussi sans aucune adaptation. Dans le deuxième cas on réalisait une télécommande directe du robot avec une transmission de l'information d'abord mécanique puis électrique et enfin assistée par l'informatique.

Les domaines de la robotique manufacturière et non-manufacturière (cf. Section 1.1.1) utilisent différents systèmes de commande et programmation qui varient en outre suivant le type d'application et l'investissement. Dans cette section nous allons en décrire les principaux types afin de dégager leurs utilisations et limitations.

### 2.1.1 Monde industriel

Un robot est une machine permettant de réaliser des fonctions multiples. Tant que des systèmes complètement autonomes n'existent pas, il revient donc à l'opérateur humain de décrire la tâche à effectuer: cette phase s'appelle la programmation du robot. Dans le monde des robots industriels, la programmation se résume le plus généralement à une description de trajectoires dans l'espace. Ces trajectoires peuvent être décrites suivant de nombreuses modalités (points de contrôles, courbes, mouvement pose-dépose, etc.) de manière statique ou dynamique et incluent les paramètres de vitesse ou d'accélération. La programmation des robots industriels s'effectue soit "en-ligne" lorsque l'on utilise le robot réel lors de la phase de programmation, soit "hors-ligne" si le robot n'est pas nécessaire pendant la phase de conception du programme.

#### 2.1.1.1 Méthodes en-ligne

La *programmation en-ligne* immobilise le robot et ses périphériques pendant toute la phase d'apprentissage et de mise au point. Cette méthode permet par contre de visualiser immédiatement le comportement, et interactivement de l'améliorer en modifiant et adaptant la programmation de la machine en question. Selon que les moteurs du robot sont passifs ou actifs durant cette phase, ce mode d'apprentissage peut encore être divisé en deux familles.

##### *Robot passif*

Les premiers enregistrements de trajectoire de robots se faisaient sur bande magnétique.

Ce mode de programmation était l'approche la plus utilisée jusque dans les années 1975. L'opérateur manipule directement l'extrémité du robot dont tous les moteurs sont débrayés. Un système stocke l'information des valeurs articulaires du robot (normalement en fonction du temps), afin de les restituer ensuite au contrôleur. Par exemple, un opérateur manipule directement le pistolet à peinture monté sur le poignet du robot[1]. Le robot doit alors posséder suffisamment de degrés de liberté pour reproduire la tâche que l'homme effectue manuellement. Ainsi le robot pourra ensuite reproduire exactement le geste du peintre (y compris par exemple l'ouverture et la fermeture de la buse), avec toutes les subtilités (et les maladresses) des gestes de l'opérateur. Dans une telle approche, il est donc nécessaire que le robot garantisse une *répétabilité* suffisante.

De nombreux problèmes sont inhérents à ce mode d'apprentissage. Principalement se sont la nécessité d'avoir un système moteur-transmission réversible, ainsi que le problème dû à la gravité qu'il faut essayer de compenser.

##### *Robot actif*

Pour palier en partie aux problèmes précédents, tout en gardant la possibilité de reproduire un geste créé par un opérateur humain, on peut décider d'utiliser le robot avec ses moteurs et son contrôleur sous tension et de le piloter grâce à

1. Cet exemple est historique. S'il est impressionnant de voir un robot effectuer la peinture d'une portière de voiture, des techniques beaucoup plus efficaces sont utilisées actuellement (projection capacitive).

un système externe. Dans ce cas, l'opérateur télécommande le robot à distance et le contrôleur enregistre la trajectoire ainsi générée. L'opérateur observe le résultat de ses actions en gardant un contact visuel constant avec le robot. Il dispose de diverses possibilités pour le pilotage:

- soit un manche à balai ("joystick") qui doit posséder autant de degrés de liberté (souvent par combinaison de touches) que le robot,
- soit un pantin (structure articulée légère avec capteurs) qui doit si possible être équilibré,
- ou encore un bras maître[2] (le robot que l'on désire programmer étant le bras esclave), réplique exacte ou à un facteur d'échelle près du robot esclave.

Ce mode d'apprentissage reste toutefois très limité car créer une trajectoire précise ou/et bien "coulée" est généralement conditonné par la qualité du système de pilotage, de la démultiplication entre le système de saisie et le bras du robot, et surtout de la dextérité de l'opérateur. En effet, avant de pouvoir "apprendre" au robot des trajectoires optimisant le temps de cycle, une phase d'entraînement de l'opérateur lui-même au système de pilotage est indispensable.

Une autre technique d'apprentissage, dite de la "*teach box*", est mieux adaptée que les précédentes à la programmation des robots industriels pour les tâches classiques de manipulation d'objets. Elle consiste à piloter le robot par l'intermédiaire d'un boîtier d'apprentissage permettant de produire des consignes et programmer des séquences. La teach box possède un certain nombre de touches[3] qui permettent d'actionner les différentes articulations du robot ou de piloter son déplacement suivant un mode cartésien (la transformation de coordonnées étant effectuée par le contrôleur) dans différents référentiels (classiquement absolu ou outil). Pour créer une trajectoire, l'opérateur place le robot dans une série de configurations importantes, et enregistre chaque fois la position complète du robot. Ensuite, il suffit d'indiquer par quels points l'on désire que le robot passe, et le contrôleur génère automatiquement les valeurs de commande pour les axes moteurs afin de réaliser la trajectoire reliant ces points clés. De plus il est parfois possible d'indiquer les profils de vitesse ou d'accélération à suivre (triangulaire, optimum pour les moteurs, temps minimum, etc.). Il faut noter que cette méthode en-ligne permet la définition de points de contrôle et non pas la génération d'une trajectoire.

#### 2.1.1.2 Méthodes hors-ligne

Il existe depuis quelques années des systèmes de *CFAO* (Conception et Fabrication Assistée par Ordinateur) permettant la *programmation hors-ligne* des robots, mais ils commencent seulement depuis quelques années à être

2. La notion de bras maître peut se rapprocher parfois de celle du pantin lorsque celui-ci possède exactement les même caractéristiques de structure (à un facteur de dimension près) que celle du robot piloté, ce qui n'est pas nécessairement le cas.
3. Une teach box peut de plus être équipée d'un petit joystick ou système équivalent pour faciliter le contrôle des mouvements du robot.

réellement implantés dans l'industrie car au vu de la complexité de la tâche, ils sont difficiles d'apprentissage et onéreux.

Le pont entre un système en-ligne et un système totalement hors-ligne est réalisé par les langages de programmation qui permettent de décrire le comportement d'un robot.

### *Langages de programmation*

Pour effectuer une définition des tâches que devra effectuer un robot grâce à un langage de programmation, l'opérateur écrit un programme informatique sur une machine dédiée ou un ordinateur classique. Ce programme rassemble une suite d'instructions interprétables par le contrôleur du robot qui générera les signaux à envoyer aux moteurs pour effectuer les mouvements désirés. Les langages de programmation pour les robots se répartissent en trois classes comme le montre le Tableau 2-1.

**Tableau 2-1:** *Classes de langages de programmation de robots*†

| **Langages Spécifiques, développés par un constructeur pour commander un ou une gamme de robots.** | **Bibliothèque de Robot, pour permettre à un langage existant de piloter des robots.** | **Nouveau Langage, général et complet avec des routines spécifiques pour certains robots.** |
|---|---|---|
| - VAL, VALII, Unimation Corp., développé originellement à Stanford University, puis Adept<br>- AL, Stanford University | - JARS, développé par le Jet Propulsion Laboratory de la NASA | - AML développé par IBM<br>- KAREL de GMF Robotics |

†. Les quelques exemples fournis le sont uniquement à titre d'illustration (ce n'est donc pas un catalogue complet). Source: [Parent 83], [Ránky 85], [Craig 89], [Dorf 90] et [Megahed 93].

Il existe évidement des solutions mixtes. Par exemple on peut utiliser un langage de programmation courant qui est exécuté sur un ordinateur indépendant du robot, et qui ne fait que transmettre des ordres (et recevoir des informations) au contrôleur du robot dans une syntaxe alors propre à celui-ci (exemple dans [Flückiger 94]).

Tous ces langages permettent des structures de contrôle (répéter *n* fois une tâche par exemple) et des structures de décision permettant par exemple de sélectionner une routine du programme en fonction de la valeur d'un capteur. Malheureusement, pour visualiser l'exécution, vérifier le fonctionnement ou même relever des points de contrôles, l'opérateur est obligé de travailler avec le robot réel en faisant exécuter le programme par le contrôleur. On ne peut donc pas vraiment parler de programmation hors-ligne totale, d'autant moins que la phase de mise au point est finalement souvent la plus longue du processus de programmation du robot.

### *Systèmes de CFAO pour la robotique*

Pour essayer de s'affranchir de l'utilisation du robot pendant toute la phase de développement d'un programme, plusieurs systèmes de programmation hors-ligne ont été développés: une station graphique se charge alors de la représentation du robot et de son comportement en trois dimensions. Le

développeur définit la tâche du robot soit par programmation comme précédemment, soit en utilisant une interface graphique (mise en correspondance de points, de lignes, ou autres constructions géométriques dans l'espace) à l'aide d'une souris ou d'un autre périphérique dédié. Le logiciel de programmation hors-ligne possède un "moteur" qui permet de simuler le comportement cinématique (voire dynamique) de différents robots conformément à leur vrai contrôleur. De plus, le logiciel doit disposer des descriptions géométriques tridimensionnelles du robot, de ses outils et des autres éléments de l'environnement.

Les systèmes de CFAO deviennent presque indispensables dans les entreprises pour garantir un temps de développement minimum. Toutefois nous sommes encore loin de la génération entièrement automatique d'un programme de robot en fonction de la description d'un produit à usiner[4] ou assembler. Ceci est dû d'une part à l'énorme complexité du processus de création d'un produit, d'autre part aux logiciels qui ne permettent pas encore de prendre totalement en compte l'environnement du robot ainsi que l'adaptation de son comportement à des variations de paramètres. Ces variations peuvent être internes (usure, répétabilité) ou externes (collision, bourrage). Finalement, la calibration entre le monde simulé et le monde réel pose encore de gros problèmes et impose des phases de test avec le robot en-ligne.

### 2.1.2 Autres robots: téléopération

Nous allons considérer dans cette section la programmation des robots ne faisant pas partie du monde manufacturier:

- les robots expérimentaux développés par les nombreux laboratoires de robotique,
- les robots industriels utilisés pour des tâches non répétitives[5],
- les robots développés pour des tâches de maintenance dans des milieux hostiles,
- les micro-robots destinés à la manipulation et l'assemblage de pièces de taille de l'ordre du centième de millimètre,
- les robots mobiles (nombreux dans le domaine de la recherche et l'exploration, quasiment uniquement à l'état de prototypes dans des applications industrielles ou de service).

Contrairement au domaine de la productique, où l'on programme une tâche qui va être répétée des milliers de fois tant qu'on ne change pas de gamme de produits, ces robots sont destinés à effectuer des actions diverses et nouvelles[6] dans un environnement qui peut se modifier. Ce sont donc

4. Malgré des algorithmes très complexes, il est parfois difficile pour un programme de déterminer directement une trajectoire en fonction d'une tâche demandée.
5. Cette classification des robots dans une catégorie manufacturière ou non-manufacturière n'est donc pas seulement en fonction du type du robot, mais aussi de son utilisation.
6. Des tâches nouvelles dans le sens où souvent l'opérateur est confronté à une situation pas encore rencontrée auparavant et pour laquelle il doit donc inventer une solution.

essentiellement des robots de coopération, des outils permettant d'agir où l'homme ne peut pas directement intervenir. Il est évident que cette catégorie de robots ne peut se programmer selon une des méthodes abordées jusqu'ici, puisqu'elles sont caractérisées par une programmation figée (série de mouvements invariants, programme défini).

C'est pourquoi la recherche en robotique a apporté d'autres solutions adaptées à des situations et tâches évolutives. Ces solutions sont de deux types: l'autonomie et la téléopération.

Plus un robot possède une grande autonomie, plus il libère l'homme de la tâche de contrôle et de supervision. Malgré les efforts en intelligence augmentée[7], dans le domaine du raisonnement neuronal ou des techniques adaptatives, nous sommes encore très loin de pouvoir fournir une autonomie suffisante pour qu'un robot évolue dans un environnement réel sans intervention humaine. Nous allons donc nous focaliser dans la suite de cette section sur les méthodes faisant intervenir l'homme dans le processus de contrôle du robot, grâce au concept de téléopération[8].

### 2.1.2.1 Téléopération directe: télécommande

En 1947 Raymond Goertz a fabriqué le premier télémanipulateur mécanique et a ensuite introduit (1954) une liaison électrique entre les systèmes maître et esclave.

Les premières téléopérations sont issues de la nécessité pour l'homme de se protéger dans la manipulation de produits nucléaires. Vu le concept de reproduction directe des mouvements par le manipulateur esclave en fonction des consignes du maître, nous précisons le type de téléopération en nommant ce principe télécommande.

Le schéma classique comprend un système mécanique articulé esclave placé dans l'environnement hostile, et un système maître (pratiquement identique à l'esclave) que l'opérateur actionne. La correspondance du mouvement entre les deux sites est dans les premiers temps assurée par une transmission en général mécanique. Le rapport d'échelle ainsi que les forces en jeu sont presque toujours unitaires. Le contact visuel entre l'opérateur et le lieu de travail du robot est habituellement possible (proximité du système maître et esclave dicté par le mode de transmission mécanique).

L'introduction d'une liaison électrique entre le maître et l'esclave permet ensuite d'éloigner les sites et de varier les forces appliquées. Le système maître n'est plus obligatoirement une réplique du manipulateur et peut donc aussi se présenter sous forme de manettes et boutons. Le contact visuel n'est plus forcément assuré, et la transmission de l'image peut alors se faire par l'intermédiaire d'un moniteur dans la salle de contrôle. Celui-ci affiche l'image saisie par une ou plusieurs caméras sur le site d'opération[9]. Il faut

7. Nous préférons utiliser intelligence augmentée plutôt que intelligence artificielle étant donné l'orientation actuelle de cette branche.
8. Les approches les plus prometteuses pour la commande d'un robot de coopération sont considérées comme un mélange d'autonomie et de téléopération: un contrôle hybride.
9. Il est intéressant de noter que ce progrès technique de l'introduction d'un écran de contrôle diminue grandement les capacités d'évaluation de l'opérateur par la perte de la vision tridimensionnelle. La vision stéreoscopique peut toutefois être recréée, mais au prix de systèmes plus complexes et au détriment du confort de l'opérateur.

noter que l'image peut être rehaussée par des informations supplémentaires (réalité augmentée) comme des indications de température par exemple.

Toutefois, la véritable innovation dans un système de téléopération est l'introduction de l'informatique. Elle permet alors une assistance à l'opérateur qui peut se passer de trois manières différentes (selon [Coiffet 93]):

1) en alterné, par exemple lorsque les mouvements d'approche d'une pièce sont réalisés par l'opérateur, puis la phase d'insertion délicate de la pièce est prise en charge par le robot;
2) en parallèle, par exemple lorsque le système fournit des informations sur l'état du système, ou qu'une caméra "suit" la zone d'intérêt pendant que l'opérateur actionne le robot;
3) en série, par exemple lorsque le système informatique filtre les mouvements de l'opérateur afin d'éviter une collision avec un obstacle connu sur le site d'opération.

#### 2.1.2.2 Téléopération par immersion: téléprésence

Une variante du concept de téléopération par télécommande se nomme téléprésence: au lieu de présenter l'information provenant du site de travail à l'opérateur sous forme d'images vidéo, on essaie de projeter l'opérateur artificiellement sur le lieu manipulation à l'aide d'une immersion totale (cf. Section 1.2.2.1) réalisée grâce à des techniques de vision stéréoscopique, de retour de forces et tactile, de sonorisation spatiale. Selon [Coiffet 93], la différence entre la téléopération assistée par ordinateur et la téléprésence réside dans le type d'informations transmises:

- lors d'une téléopération, on essaie dans la mesure du possible de "nettoyer" (pour n'en garder que l'essentiel) l'information que l'on transmet à l'humain, et de lui fournir en plus le maximum de grandeurs mesurables pertinentes (que l'humain seul ne pourrait pas forcément observer) afin qu'il puisse prendre une décision,
- la téléprésence transporte "virtuellement" l'homme sur le site d'opération afin qu'il perçoive de lui-même le maximum d'informations; c'est l'homme seul qui juge de la situation par exemple en ressentant une augmentation de chaleur.

Toutefois, la téléprésence, de par la méconnaissance du type d'informations à fournir à l'homme, et par la piètre qualité des stimuli sensoriels artificiels que l'on sait générer pour le moment, n'est explorée qu'avec des prototypes de laboratoire de recherche.

#### 2.1.2.3 Téléopération différée: définition de tâche

Dans nombre de situations la téléopération classique n'est pas réaliste:

- Dans les situations où la tâche à réaliser comporte des risques (pour le robot ou pour l'environnement), il est dangereux d'avoir une liaison directe entre les mouvements de l'opérateur et l'action du robot (le filtrage des mouvements accidentels pourrait alors améliorer la situation, comme dans le mode de télécommande série vue plus haut).

- Dans les situations où la liaison entre le site de travail et le lieu de contrôle possède soit une bande passante très faible (transmission sous-marine sans câble), ou relativement bruitée, on ne peut obtenir des informations suffisantes pour effectuer une téléopération sur des base visuelles.
- Lorsque le site de travail est très éloigné du lieu de contrôle (mission interplanétaire ou même déjà terrestre à longue distance), les temps de transmission sont tels qu'il n'est pas possible d'effectuer une téléopération car le délai compromet la boucle de contrôle homme-machine. De même dans le domaine du microscopique les délais d'obtention d'une image (par exemple avec un microscope AFM) n'autorisent pas une manipulation directe.

Dans ces cas il est nécessaire de pouvoir planifier la tâche du robot dans un environnement similaire à celui dans lequel il évoluera: l'utilisation d'un environnement virtuel permet cela[10]. Une fois cette planification effectuée, elle est envoyée au robot réel qui effectuera la tâche avec adaptation éventuelle aux variations des conditions initiales. Lorsqu'un problème est rencontré, un nombre minimum d'informations peut être renvoyé au centre de contrôle afin de mettre à jour l'environnement virtuel et de permettre à l'opérateur de réévaluer la situation. Cette méthode doit être accompagnée d'un système d'acquisition des parties de l'environnement réel non connues a priori afin d'actualiser l'environnement virtuel avec ces données.

### 2.1.3 Conclusion

Il est clair que les méthodes de programmation industrielles des robots ne sont pas adaptées à des environnements non structurés et dynamiques. Pour cette raison le projet «CINEGEN» dont l'objet est entre autre la création rapide d'environnements virtuels pour le pilotage des robots de coopération adopte le concept de téléopération.

La classification des méthodes de téléopération présentée ci-dessus est relativement rigide et chaque groupe de recherche propose des concepts différents pour la téléopération de systèmes robotisés. Par exemple, une autre classification des interfaces 3D pour la téléopération, basée sur le degré d'abstraction qu'elles fournissent, est proposée dans [Schenker 94]. On peut citer quelques exemples de téléopération qui diffèrent sensiblement de l'approche proposée.

- Le concept de "teleprogramming" [Funda 91] pour piloter des robots sous-marins: l'opérateur pilote un robot virtuel, et le robot réel effectue les mouvements correspondant avec un léger différé (délai de transmission) grâce à une autonomie partielle.
- L'utilisation de "predictive display" [Askew 93] utilisée par le Johnson Space Center pour les missions de maintenance spatiale: la

10. Un système de CFAO ne peut pas être considéré comme un système de téléopération vu que l'échange d'informations entre le lieu de contrôle et le robot n'est ni bidirectionnel, ni dynamique.

superposition du modèle virtuel et d'un modèle filaire dont les positions proviennent du robot réel permettent d'améliorer le pilotage avec délais.

- Le "teaching by showing" [Ogata 94]: une tâche est définie de manière totalement indépendante du robot dans un environnement virtuel, elle est ensuite analysée de manière symbolique et transformée en commandes pour un robot particulier.

Toutefois, l'approche de téléopération séquentielle (définition de la tâche dans le monde virtuel puis exécution de la tâche par le robot réel) avec autonomie locale du robot proposée dans la Section 2.1.2.3 possède aussi des atouts certains, comme le découplage temporel total ou la possibilité de vérifier la tâche avant l'exécution. Cette méthode a été testée avec succès entre autres dans les missions suivantes:

- téléopération de robots sous-marins, robots mobiles terrestres divers par l'Intelligent Mechanisms Group du NASA Ames Research Center [Hine 95] [Piguet 95] [Christian 97],
- téléopération transatlantique d'un robot manipulateur par le Département de Microtechnique de l'EPFL [Flückiger 94] [Natonek 95].

Il faut noter que le programme *VirtualRobot* n'est pas forcément lié à une des méthodes de téléopération proposée et peut s'interfacer de différentes manières suivant les situations désirées. *VirtualRobot* fournit en effet le "moteur" permettant d'interagir avec un robot virtuel quelconque, il ne fige pas les moyens de communication avec l'environnement réel.

## 2.2 Résolution de la cinématique des robots

Les équations cinématiques[11] modélisant le comportement des robots proviennent soit des équations de contraintes aux articulations soit des équations de fermeture de boucle. Dans les deux cas, ces équations comportent une description mathématique en terme de fonctions trigonométriques et de leurs produits. Ceci amène à des équations fortement non linéaires posant de sérieux problèmes pour leur résolution.

De nombreuses approches ont été mises au point pour étudier la cinématique des robots. La plupart s'appuient sur des résultats antérieurs utilisés pour l'étude des mécanismes ou sur des outils mathématiques[12] plus généraux.

Il est évidement possible de classifier ces méthodes suivant plusieurs critères. Elles sont présentées dans cette section suivant leurs caractères quand aux solutions fournies: les méthodes algébriques qui aboutissent normalement

11. Dans cette section le terme "cinématique" sera utilisé suivant sa signification courante anglo-saxonne: "*kinematics*" (l'étude du comportement des robots aussi bien en position que vitesse ou accélération).
12. L'étude des robots s'intensifie depuis trois décades, alors que l'étude des mécanismes propose des méthodes depuis plusieurs siècles.

à des solutions sous forme symbolique exacte, les méthodes itératives (numériques ou non) procédant par approximation et quelques méthodes originales.

Cette section n'a pas la prétention de faire un état de l'art complet des méthodes utilisées pour résoudre la cinématique des robots. Par contre les principales méthodes sont brièvement décrites afin de comprendre leur domaine d'application privilégié et leurs limitations les plus importantes. Le lecteur se reportera aux références citées pour approfondir un sujet particulier.

## 2.2.1 Méthodes symboliques

Les solutions algébriques, les plus élégantes au niveau mathématique, permettent de mettre en évidence certaines propriétés des robots et sont les plus performantes (quand elles existent) au niveau des calculs informatiques.

La première pierre dans la recherche du modèle géométrique inverse symbolique des robots sériels à été posée en 1968 par Pieper [Pieper 69] avec une solution pour certains manipulateurs six axes particuliers. Le cas général du robot à 6 degrés de liberté reste encore un challenge. Pour les robots parallèles, les solutions aux modèles géométriques directs ont été fournies en fonction du développement récent de manipulateurs particuliers. On peut citer par exemple [Clavel 91] (robot en translation) et [Gosselin 92] [Gosselin 94a] (robot sphérique). Pour des structures parallèles plus complexes comme la plateforme de Gough-Stewart datant de 1960, il a fallu attendre cette décennie pour simplement borner le nombre de solutions à 40 [Lazard 92]. Plus récemment Husty a obtenu une formulation algébrique de ces solutions [Husty 94].

### 2.2.1.1 "fait main"

La première approche pour le problème de la cinématique d'un robot consiste à essayer de poser et résoudre à la main les équations le gouvernant. La réflexion permet de déceler des solutions à des problèmes complexes. L'aide de programmes de calcul symbolique facilite aussi la résolution dans les cas impliquant des équations "lourdes".

La plupart des ouvrages de références en robotique comme [Asada 86], [Craig 89], [Dombre 88], [Megahed 93], [Paul 83], [Ránky 85], [Vukobratovic 89] et [Yoshikiwa 90] proposent des exemples de résolutions de cinématiques sérielles particulières et quelques procédures pour y arriver de manière rigoureuse. Dans le cas des robots parallèles, Merlet regroupe et décrit les solutions pour plusieurs classes de structures parallèles dans [Merlet 97].

#### *Solutions géométriques*

La plupart du temps les équations sont posées à partir de considérations géométriques sur la structure du robot. Ceci est réalisable pour des structures simples (deux ou trois degrés de liberté). Dans le cas de structures spatiales plus compliquées, on essaie de décomposer le problème en sous-problèmes plans [Craig 89]. Un exemple de résolution géométrique pour un

manipulateur parallèle particulier (le robot Delta) peut être trouvé dans [Clavel 91].

#### *Solutions algébriques*

Les équations peuvent aussi provenir du modèle direct (dans le cas de robots sériels, inverse dans le cas de robots parallèles) que l'on essaie d'inverser. Par une suite de manipulations algébriques, on isole les variables désirées. Ces manipulations peuvent soit être inspirées par la réflexion mathématique, soit suivre une méthodologie qui existe pour certaines classes de structures.

Toutefois la complexité des équations devient très grande pour des structures "utiles" (plus de 4 degrés de liberté) et elles deviennent alors le plus souvent insolubles de manière symbolique. Il existe toutefois des solutions pour des classes de structures particulières. Par exemple pour les robots à six degrés de liberté dont les axes de trois articulations successives se croisent en un point, on peut appliquer la méthode de "Pieper" [Craig 89]. D'ailleurs beaucoup de robots industriels sont conçus de manière à posséder de telles propriétés afin de trouver un modèle acceptable.

#### *Conclusion*

Les méthodes "fait main" aboutissent (lorsqu'elles aboutissent) à des solutions souvent optimales au niveau de l'implémentation informatique (solution algébrique minimale) et sont donc encore largement utilisées pour réaliser la commande de robots qui doit être très rapide. Toutefois, il est évident que ces méthodes résolvent les problèmes au cas par cas et ne peuvent donc pas être utilisées pour une approche générique.

### 2.2.1.2 Bases de Gröbner

Une recherche d'automatisation et de généralisation de ces méthodes de résolution symbolique à la main a bien sûr été envisagée. Un outil largement utilisé à cette fin sont les bases de Gröbner. Plusieurs ouvrages de référence cités dans [Heck 96] traitent de la théorie de bases de Gröbner qui permettent d'exprimer un ensemble d'équations polynomiales sous une forme "standard" facilitant ensuite la résolution. Les bases de Gröbner sont le plus souvent obtenues à l'aide de l'algorithme de Buchberger [Buchberger 85] ou de certaines versions améliorées de cet algorithme [Becker 93].

#### *Utilisation d'outils pour le calcul polynomial*

Les bases de Gröbner ont été utilisées pour résoudre le modèle géométrique inverse de robots sériels (par exemple [GL 93]) ou trouver le modèle géométrique direct de manipulateurs pleinement parallèles comme le montre Tancredi dans son travail de thèse [Tancredi 95]. Dans ce dernier cas, la résolution se fait avec des coefficients numériques à l'aide de logiciels de calcul de base de Gröbner.

#### *Utilisation de résolveurs de contraintes*

Une application intéressante des bases de Gröbner et de l'algorithme de Buchberger est la réalisation de résolveur de contraintes permettant de gérer des équations non-linéaires [Hollman 93] [Fron 94]. Un des ces résolveurs,

CAL (Contrainte Avec Logique) a été utilisé dans [Sato 93] pour résoudre le problème du modèle direct et inverse de robots sériels. Les équations représentant le robot sont simplement obtenues par la multiplication des matrices de transformation de chaque lien du robot. Il est ensuite possible de demander au système de manière unifiée quelle est la posture finale en fonction d'angles fournis aux articulations, ou alors, quels sont ces angles en fonction d'une posture donnée. La réponse sous forme symbolique, mais avec des paramètres numériques, est fournie par le résolveur de contraintes en quelques secondes (2 à 30 suivant le nombre de degrés de liberté).

*Conclusion*

Les bases de Gröbner semblent être un outil prometteur qui est pour le moment limité à des cas simples de par le temps de calcul nécessaire pour les obtenir[13] et l'impossibilité de manipuler des équations avec trop de variables (obligation de résolution numérique pour certains paramètres fixés). De plus [Weiss 93] montre que les bases de Gröbner sont moins efficaces pour la résolution des cinématiques inverses de manipulateurs sériels que les méthodes par élimination (cf. paragraphe suivant).

#### 2.2.1.3 Méthodes par élimination

La théorie des mécanismes fait souvent recours à des méthodes par élimination qui permet de formuler un ensemble d'équations non-linéaires sous forme d'un système augmenté linéaire grâce à l'introduction de variables supplémentaires. Cette méthode était limitée à des cas simples sous peine d'introduction d'un grand nombre de solutions non désirées. Mais [Roth 93] a apporté une modification à l'élimination dialytique de Sylvester qui permet de l'appliquer à des problèmes pratiques de cinématique de robots.

Les méthodes par élimination ont permis notamment de trouver une solution algébrique au modèle géométrique inverse des manipulateurs à six degrés de liberté en rotation [Raghavan 90] dans le cas général, ainsi que des manipulateurs à cinq degrés de liberté dérivés. L'élimination a aussi été utilisée par Husty [Husty 94] pour le calcul final de la solution du modèle géométrique direct de la plateforme de Stewart-Gough généralisée et apporte ainsi la preuve qu'il admet 40 solutions au maximum (obtention d'un polynôme uni-variable du $40^{ième}$ degré).

*Conclusion*

Si les méthodes par élimination ont permis d'aboutir à des résultats remarquables dans le domaine de la cinématique des robots, elles restent délicates à mettre en oeuvre et difficilement automatisables pour des structures articulées générales. En effet ces méthodes sont très sensibles au nombre d'inconnues et au degré des équations [Tancredi 96]. De ce fait la paramètrisation choisie pour la modélisation du problème influence grandement les résultats.

13. Pour des équations de taille raisonnable on ne peut pas obtenir une résolution en temps réel, pour des cas plus complexes le temps de calcul explose et il ne faut plus compter obtenir une solution.

## 2.2.2 Méthodes itératives

Les modèles cinématiques des robot n'étant pas les seuls problèmes non-linéaires que l'ingénieur rencontre, il existe toute une classe de méthodes itératives destinées à trouver des solutions acceptables à ce type de problème. Ces méthodes itératives procèdent par approximations successives jusqu'à obtenir une erreur sur la solution inférieure à un seuil pré-défini.

Les méthodes itératives (numériques ou non) ont été utilisées dans tous les domaines de l'ingénierie où des problèmes non-linéaires apparaissent. Elles sont donc souvent beaucoup plus anciennes que les solutions symboliques déterminées au coup par coup en fonction des nouveaux types de robots inventés. Toutefois, de nombreuses adaptations de ces méthodes ont été proposées pour une application optimale à la cinématique des robots.

### 2.2.2.1 Linéarisation du modèle

L'approche classique pour résoudre un problème non-linéaire consiste à le linéariser en utilisant une série de Taylor (du premier degré généralement). Ensuite on peut utiliser une méthode de Newton pour calculer une solution approchée (voir par exemple [Vukobratovic 86] ou [Coiffet 92]). La méthode de Newton possède une convergence quadratique rapide, mais donne évidemment une seule solution, la plus proche du point de départ. D'autres algorithmes comme celui du gradient conjugué peuvent être utilisés pour obtenir une solution par approximation.

Les méthodes itératives de ce genre ne fournissent toujours qu'une seule solution alors que les systèmes mécaniques articulés possèdent presque toujours de nombreuses solutions. On peut donc schématiser comme suit les résultats obtenus par une méthode itérative:

- si on fournit un mauvais choix initial (pour l'approximation), on obtient une convergence très lente, voir une divergence;
- si on fournit un bon choix initial, on obtient une solution unique la plus proche du choix initial.

La linéarisation du modèle revient à dériver partiellement les équations en fonction de leurs variables. L'expression de ces dérivées partielles est utilisée pour former la matrice jacobienne (cf. Section 3.1.3). On peut donc assimiler la méthode de Newton qui procède par incréments de positions, à une méthode résolvant le problème dans l'espace des vitesses si l'on rend les incréments infinitésimaux.

La formulation du problème dans l'espace des vitesses est couramment utilisée pour le contrôle de trajectoire et se sert de l'expression de la matrice jacobienne comme base de résolution. L'inversion de la matrice jacobienne dans certains cas particuliers peut s'effectuer symboliquement, mais la plupart du temps on utilise une méthode numérique pour l'inversion. L'inversion numérique de la matrice jacobienne est une méthode largement utilisée pour le contrôle des robots, mais pose des problèmes de singularité lorsque cette matrice perd un rang [Asada 86]. Des méthodes pour la décomposition symbolique en valeurs singulières ont aussi été proposées [Kircanski 95] afin

d'augmenter considérablement (facteur 7 à 14) la vitesse de l'inversion de la matrice jacobienne pour les robots sériels redondants.

*Conclusion*

Les méthodes itératives numériques ne conduisent pas à toutes les solutions lors d'un calcul de cinématique inverse, mais sont néanmoins largement utilisées car souvent le problème se limite à suivre une trajectoire (points de contrôles proches les uns des autres ne nécessitant pas de changer de *configuration*). Si l'inversion numérique de la matrice jacobienne n'était généralement pas envisageable pour un contrôle en temps réel jusque dans les années 1990, l'augmentation de la puissance des ordinateurs autorise actuellement cette approche. Elle sera retenue pour le projet «CINEGEN».

#### 2.2.2.2 Méthode par homotopie[14]

L'idée de base d'une méthode par homotopie pour résoudre un système polynomial non-linéaire, repose sur le constat que de petits changements dans les paramètres du système ne produisent que des petites variations dans les solutions. Le principe est alors le suivant: si l'on doit résoudre un système S, on choisit un système C proche de S dont on connaît les solutions. On "suit" alors les solutions qui évoluent en fonction des changement des paramètres du système C pour arriver au système S. Comme expliqué dans [Wampler 90], la résolution d'un système S par homotopie se compose alors de trois éléments: un système de départ C dont on connaît les solutions, un procédé pour faire évoluer les paramètres de C vers ceux de S, et une méthode pour suivre le changement des solutions en fonction de la transformation du système.

Les systèmes d'homotopies ont été utilisés pour trouver les solutions complètes de manipulateurs à cinq et six degrés de liberté généraux: [Tsai 85], [Li 90] et [Wampler 91] entre autres. Elles ont aussi permis d'apporter la preuve qu'il existe au maximum 40 solutions pour l'assemblage (les différentes configurations possible) d'une plateforme de Stewart généralisée par [Raghavan 91].

*Conclusion*

Les méthodes par homotopies ont permis de formuler des preuves importantes dans le domaine de la cinématique des robots pour des cas généraux. Toutefois ces méthodes sont sensibles à la formulation du système polynomial du modèle cherché. De plus le système de départ n'est pas forcément facile à trouver.

Finalement les méthodes par homotopies n'ont pas vraiment "percé" dans l'analyse cinématique. Selon [Roth 94] elles sont plus limitées que les méthodes par élimination et moins efficaces.

### 2.2.3 Autres méthodes

En dehors des approches "classiques" algébriques ou itératives utilisées pour résoudre la cinématique des robots, il existe quelques méthodes originales

14. "Continuation method" dans la terminologie anglo-saxonne.

provenant d'autres domaines et appliquées à la robotique. Nous citerons entre autres une méthode basée sur le raisonnement géométrique et une deuxième basée sur les approches "neuronales". Ces méthodes, testées sur certains problèmes de robotique avec succès, restent toutefois restreintes à quelques problèmes spécifiques.

### 2.2.3.1 Géométrique

Le premier outil utilisé pour l'étude des mécanismes par l'ingénieur était le raisonnement géométrique. A partir de constructions géométriques sur du papier, on essayait de résoudre des contraintes que devait satisfaire un mécanisme donné. Souvent ces problèmes restaient limités au plan de par la difficulté des constructions géométriques dans l'espace. L'ingénieur a ensuite explicité les relations entre les éléments formant un mécanisme sous forme d'équations algébriques. Le problème géométrique se transforme alors en un problème algébrique que l'on peut solutionner soit symboliquement, soit numériquement. Kramer dans son travail de thèse [Kramer 92] propose de réutiliser un raisonnement purement géométrique pour calculer l'analyse et la simulation des mécanismes (plans et spatiaux).

Le système mécanique articulé est alors exprimé sous forme d'un ensemble de contraintes géométriques que le système va résoudre en appliquant des règles géométriques. Par exemple la solution du système à deux barres de la Figure 2-1 consiste simplement à trouver l'orientation des deux corps en calculant l'intersection de deux cercles.

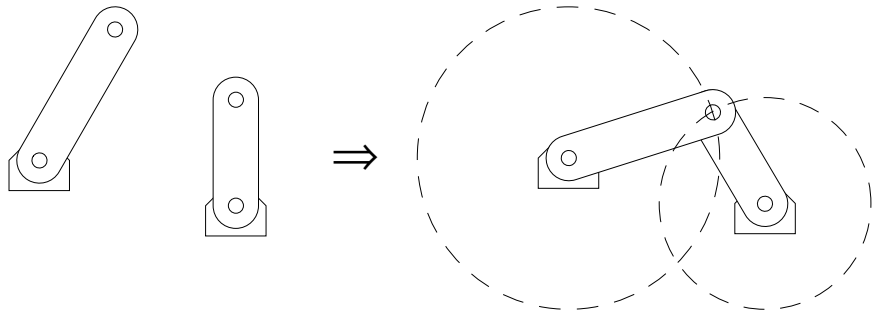


**Figure 2-1:** *Résolution d'un mécanisme à deux barres par raisonnement géométrique*

Chaque solide est référencé par des "markers" qui sont en quelque sorte des référentiels entre lesquels on va pouvoir définir des relations et calculer des mesures. Le système basé sur un résolveur de contraintes LISP essaye de "construire" le mécanisme en satisfaisant successivement chaque règle. Chaque fois qu'une règle entre deux corps est satisfaite, elle enlève un ou plusieurs degrés de liberté aux éléments concernés: le système mécanique est "réduit" avec moins de degrés de liberté jusqu'à aboutir à la solution finale.

Des comparaisons entre le programme TLA de simulation développé par Kramer et le logiciel ADAMS[15] effectuées sur des systèmes complexes[16] ont

15. Logiciel de simulation de systèmes mécaniques du commerce basé sur une résolution numérique des contraintes.
16. Dont la simulation d'un dépliage du mécanisme d'un "sofa" comprenant 16 corps, 22 joints et 7 boucles cinématiques.

montré une vitesse d'exécution environ deux fois supérieure pour TLA. De plus les problèmes doivent être spécifiés avec beaucoup plus de soins dans le cas de ADAMS pour éviter de tomber artificiellement dans un système sur-contraint ou bien qui dégénère près des singularités.

*Conclusion*

L'approche géométrique semble très efficace car elle permet de résoudre, à l'aide d'un raisonnement sur des règles de base, des problèmes à caractère hautement non-linéaire lorsqu'ils sont spécifiés sous forme d'équations. Cette approche a montré son efficacité et a été reprise dans d'autres travaux [Brunkhart 94], [Salomons 95] et [Noort 98].

Toutefois si cette approche permet de résoudre des problèmes où apparaissent un grand nombre de boucles cinématiques, elle ne fonctionne pas quand les boucles contiennent plus de cinq éléments[17]. Dans ces cas, on pourrait envisager d'utiliser un résolveur de contraintes numérique. L'approche mixte semble certainement prometteuse (utilisée pour l'éditeur dynamique de mécanisme plan, MECH de [Brunkhart 94]), mais elle reste difficile à implémenter dans un cadre général de mécanismes hybrides généraux et spatiaux.

### 2.2.3.2 Réseaux de neurones

Les réseaux de neurones, ou plus généralement les approches neuronales ont été appliquées pour trouver des solutions à des problèmes divers dans le domaine de la robotique[18]. Les principaux travaux de recherche sur ce thème sont cités dans [Omidvar 97]. La littérature sur les réseaux de neurones appliqués aux robots manipulateurs ne considère que les cas de structures sérielles et traitent donc du problème de leur cinématique inverse. D'autre part la majorité des exemples sont limités à des simulations de structures planes.

Les réseaux de neurones présentent un grand potentiel au niveau de l'adaptation et des transformations non-linéaires. La caractéristique adaptative des réseaux de neurones nous intéresse moins dans le cadre de ce projet car elle est antinomique avec le concept de stabilité et de répétabilité cyclique. Par contre, l'aptitude des réseaux de neurones à traiter de problèmes non-linéaires est un atout pour la commande des robots manipulateurs. En effet, la puissance d'un réseau de neurones pour effectuer des transformations non-linéaires est considérable par rapport au coût de son implémentation matérielle[19], ce qui permettrait par exemple de concevoir des contrôleurs embarqués simples et bon marchés.

La mise en oeuvre d'un réseau de neurones comprend toujours une phase d'apprentissage (quel que soit le type de structures neuronales et de contrôle utilisé). Il est donc nécessaire d'avoir un modèle (au moins partiel) du système

17. On ne pourrait donc pas résoudre la cinématique d'un robot à six degrés de liberté.
18. Dans la recherche comportementale et adaptative des systèmes, d'analyse de l'environnement, de la navigation des robots mobiles, de simulation et contrôle des bras de robot.
19. Un "chip" neuronal est largement plus puissant qu'un processeur très rapide dans ce domaine.

à simuler afin d'entraîner le réseau. Ceci permet de faire une taxinomie primaire des réseaux de neurones appliqués à la robotique en deux groupes:

1) les méthodes directes qui utilisent le modèle cinématique inverse du robot à contrôler dans la phase d'apprentissage,

2) les méthodes indirectes qui se contentent du modèle cinématique direct (beaucoup plus simple à obtenir) durant la phase d'apprentissage.

Les méthodes directes sont évidemment plus efficaces, sous réserve de pouvoir obtenir le modèle inverse du manipulateur considéré. Les méthodes indirectes peuvent être améliorées grâce à l'ajout d'une boucle de rétroaction [Gardner 93]. Les réseaux de neurones sont soit utilisés pour obtenir les variables articulaires en fonction des variables cartésiennes fournies en entrée, soit pour calculer les valeurs de la matrice jacobienne inverse, toujours en fonction des variables cartésiennes. Des mises en oeuvre ont été réalisées pour déterminer le modèle géométrique inverse, le modèle cinématique [Kuroe 94], ou même le modèle dynamique [Wu 94].

Il est aussi possible d'exploiter le fait qu'un réseau de neurones travaille par approximations locales. Par exemple, on peut entraîner un réseau de neurones avec un ensemble de vecteurs positions cartésiennes / valeurs articulaires en évitant les singularités du manipulateur. Pendant le contrôle réel, lorsqu'une consigne cartésienne générant une singularité avec le modèle mathématique, est injectée, le réseau de neurones va fournir une proposition légèrement fausse, mais hors de la singularité.

#### *Conclusion*

L'approche neuronale possède des atouts au niveau informatique vu le caractère non linéaire du modèle des robots manipulateurs. Toutefois elle est limitée dans le cadre de «CINEGEN» car elle nécessite de connaître au minimum le modèle direct, que l'on ne peut obtenir de manière générale pour les robots parallèles. De plus, une fois "entraîné", le réseau reste statique dans le sens où il ne peut pas prendre en compte une modification de structure[20].

Les réseaux de neurones sont donc surtout intéressants pour la réalisation de contrôleurs de robots particuliers et pour l'amélioration du comportement.

## 2.3 Logiciels de simulation robotique

Parallèlement au développement des robots, de nombreux outils pour les simuler en vue de leur étude ont vu le jour. Certains logiciels relèvent de la recherche, mais de nombreux outils dans ce domaine sont proposés par le marché.

### 2.3.1 Logiciels commerciaux

Les développements de simulations par ordinateur de systèmes mécaniques datent de plus de 20 ans. L'avancée des recherches dans ce domaine,

20. Par exemple une patte de robot marcheur constitue une boucle fermée lorsqu'elle est au sol, puis ouverte lorsqu'elle se lève.

combinée à la croissance de puissance des ordinateurs a abouti sur de nombreux logiciels atteignant une maturité commerciale depuis plus de 10 ans. La large palette de l'offre d'outils de simulation dans ce domaine est évidemment directement issue de la demande de l'industrie. Le monde industriel a besoin d'outils de simulation pour optimiser et mieux comprendre le fonctionnement de systèmes complexes. La simulation permet donc d'éviter de construire des prototypes dans la première phase de développement, de réduire le temps de conception d'un produit et d'améliorer sa qualité.

On peut classer les logiciels de simulation en deux groupes: les logiciels à vocation générale pour les systèmes mécaniques quelconques et les logiciels plus spécialisés pour les robots industriels.

#### 2.3.1.1 Logiciels d'analyse cinématique et dynamique

Les outils généraux permettent la simulation de systèmes mécaniques à corps multiples quelconques (suspensions de voitures, antennes de satellites, machines de construction dans le bâtiment, etc.). De par leur généralité, ces outils permettent aussi de simuler les mouvements et la dynamique des robots qui ne sont qu'un cas particulier de systèmes mécaniques. Dans cette catégorie, on peut citer les principaux logiciels qui sont largement implantés dans le monde industriel et les universités[21]:

- ADAMS de Mechanical Dynamics Corporation [wwwADA],
- DADS de CADSI [wwwDAD],
- MECHANICA de Parametric Technology Corporation [wwwMEC],
- WorkingModel de Knowledge Revolution [wwwWMO].

Ces produits comportent tous les mêmes fonctionnalités de base: principalement un moteur de résolution qui formule les équations automatiquement et fournit les solutions pour la statique, les mouvements ou la dynamique du système mécanique étudié. Ces outils proposent des modules de visualisation permettant d'effectuer une simulation en trois dimensions du comportement calculé. Un accent est mis sur les potentialités à modéliser le comportement dynamique de systèmes à corps rigides multiples formant plusieurs boucles cinématiques. En effet, l'analyse dynamique prend une place croissante dans la conception des systèmes de plus en plus complexes afin de trouver les comportements optimums et une fiabilité accrue.

Les moteurs de résolution de ces outils sont principalement basés sur des méthodes numériques. Le système complet est exprimé sous forme de matrices "sparse" souvent de grandes dimensions, regroupant la décomposition du système en composants simples. La résolution du système d'équations ainsi formé permet de fournir des solutions à des systèmes extrêmement complexes généralement non solvables analytiquement.

21. Les références données dans cette section ne se trouvent pas dans la bibliographie mais dans la liste d'adresses web (url) fournie au chapitre "URLs" à la page 141. En effet les manuels de références de ces produits sont souvent difficiles à obtenir. Le lecteur peut donc trouver une information sur ces produits à l'adresse du site web de ces compagnies.

Ces logiciels comportent des programmes de dessin tridimensionnel assisté par ordinateur, ou alors sont interfacés avec de tels programmes. L'intégration de ces outils de simulation avec les technologies de conception assistée par ordinateur amène au concept de "Virtual Prototyping": la conception de prototypes virtuels.

Finalement, nous mentionnons encore certains programmes qui utilisent des méthodes un peu différentes pour la résolution de la cinématique/dynamique des systèmes mécaniques:

- James de Simulog [wwwJAM] propose une analyse d'un système poly-articulé par des manipulations symboliques. Ensuite un code est généré dans différents langages de programmation (Fortran, C, Ada ou Matlab) pour effectuer la résolution numérique du système d'équations symboliques. Ceci améliore les performances du simulateur et permet de l'interfacer avec d'autres programmes.
- Analityx de Saltire Software [wwwANA] propose des programmes de simulation cinématique et dynamique de systèmes articulés dont une partie de la solution peut-être calculée par des algorithmes géométriques.

#### 2.3.1.2 Logiciels de simulation de robots industriels

Une gamme de logiciels dédiés à la simulation des robots permet aux concepteurs et utilisateurs de robots de mettre en oeuvre virtuellement des robots dans leur environnement de travail. Ces outils permettent la simulation de robots dans une cellule de production complète en vue de la validation d'un nouveau système ou/et de l'optimisation d'une cellule existante. En plus de la conception et de l'analyse de cellules de production, ces programmes sont essentiellement utilisés pour la programmation hors-ligne des robots. Cette fonctionnalité permet la programmation complète d'une installation robotisée sans immobiliser les ressources. Les produits les plus connus sont:

- Cim Station (Robotics) de Silma [wwwCIM],
- iGrip de Deneb Robotics [wwwIGR],
- Robcad de Tecnomatix Technologies Ltd. [wwwROB],
- Workspace[22] de Robot Simulations Ltd. [wwwWSP].

Ces outils permettent la simulation de robots industriels ainsi que de la périphérie (convoyeurs, bols d'alimentation, etc.). Ils possèdent chacun certaines caractéristiques propres, mais proposent plus ou moins les mêmes fonctionnalités de base. La plupart de ces programmes sont fournis avec une bibliothèque de robots existants permettant la résolution de leur cinématique (voire dynamique). Ceci permet une simulation performante et exacte de mouvements des robots. Lorsque le modèle n'est pas inclus, il est possible de le programmer ou d'utiliser parfois un générateur automatique de

22. Workspace, premier logiciel de simulation de robot fonctionnant sur ordinateur personnel (PC), est disponible sur le marché depuis 1989.

cinématique. Certains modules proposés sont spécialisés pour des tâches particulières comme le soudage, la peinture ou le polissage.

Ces programmes comportent un système d'affichage des simulations calculées dans l'espace tridimensionnel. Cela permet à l'utilisateur de visualiser le comportement du système complet sous différents angles et à différents niveaux de détail. Certains programmes comme iGrip permettent même l'intégration du facteur humain dans la simulation: des modèles virtuels de personnes permettent de visualiser l'interaction d'opérateurs avec les machines ainsi que les temps de déplacements (par exemple pour se rendre sur une machine bloquée).

Finalement ces logiciels de simulation gèrent les différents langages de programmation des robots pour produire le code nécessaire au contrôleur du robot réel. D'ailleurs le modèle du robot est souvent calculé en fonction du comportement de son contrôleur réel recevant des ordres de programmation dans un langage donné. iGrip permet aussi d'interfacer la simulation avec un programme créé par le développeur d'un nouveau contrôleur afin de le tester et de l'optimiser avant implémentation finale sur un robot réel.

### 2.3.2 Logiciels dans la recherche académique

Etant donné l'offre de produits commerciaux pour la simulation des systèmes en général, les logiciels de simulation de robots développés par des laboratoires de recherche sont surtout focalisés sur des problèmes spécifiques. Ce sont des outils d'aide à la conception et à l'analyse de structures particulières. Nous ne pouvons pas faire ici un inventaire détaillé des simulateurs existants, mais simplement en présenter sommairement quelques uns pour comprendre le genre de tâche auxquelles ils sont dédiés:

- SMAPS [Gosselin 94b] est dédié à la simulation de robots parallèles sphériques à trois degrés de liberté. Il permet l'analyse cinématique et dynamique de ces structures, la visualisation de leur enveloppe de travail et des configurations singulières. La résolution des modèles se fait sur la base des solutions algébriques de ces mécanismes.
- SYMORO+ [Khalil 97] ne permet pas une simulation tridimensionnelle directe, mais la génération sous forme symbolique des modèles géométriques, cinématiques et dynamiques de robots. La méthode se base sur un ensemble de modèles connus pour des structures sérielles particulières. Des équations supplémentaires sont ajoutées pour prendre en compte les structures bouclées.
- Robotica [Nethery 93] [Nethery nc] est un module écrit en Mathematica [Bahder 95] permettant de calculer la cinématique directe et la dynamique de robots sériels et d'afficher graphiquement une représentation de ces robots.
- Les logiciels du projet SAGA [wwwSAG] permettent de calculer les modèles géométriques directs de la plupart des robots parallèles suivant une méthode basée sur le calcul polynomial. Ces robots

peuvent aussi être visualisés à l'aide d'un programme dédié représentant les robots en trois dimensions dans un mode filaire.

- Mobile [Kecskeméthy 95] est une bibliothèque écrite en C++ permettant de créer des simulations de systèmes mécaniques articulés généraux. Mobile propose un approche orientée objet pour la formulation et la résolution de la dynamique (et donc aussi cinématique) des systèmes à corps multiples avec boucles cinématiques [Hiller 92] [Kecskeméthy 94]. Cette méthode permet de développer rapidement un programme en assemblant des objets de base en vue de la simulation du système. Une interface graphique permet de visualiser le mécanisme simulé en trois dimensions et des panneaux de contrôle peuvent être programmés pour manipuler le système.

### 2.3.3 Conclusion

Les programmes de simulation ou d'analyse développés par les laboratoires de recherche sont dédiés à des études spécifiques. Ils sont donc très performants pour un domaine particulier, mais ne permettent pas la généralité souhaitée. Même l'approche consistant en la création d'une bibliothèque de structures connues reste limitée, car si les différentes possibilités pour la création de robots sériels sont restreintes, l'imagination est la seule limite pour l'invention de nouvelles structures parallèles. De plus ces outils sont surtout conçus pour la conception et l'analyse de structures particulières et non pas pour le pilotage interactif de robots.

La qualité et les performances des programmes commerciaux commencent seulement à s'accompagner d'une interface relativement conviviale pour la description de simulation de robot. Toutefois, la limitation fondamentale de ces outils par rapport à l'approche proposée dans ce travail est le manque d'interactivité. Ces programmes commerciaux sont des outils de simulation, c'est-à-dire qu'ils fournissent un résultat (sous forme graphique tridimensionnelle par exemple) en fonction de consignes définies à l'avance. L'utilisateur n'est donc que spectateur d'une scène qui se déroule en fonction de paramètres qu'il a définis. De plus les simulations en temps réel ne peuvent être réalisées que par certains programmes dans des cas relativement simples. Il faut noter toutefois qu'une version étendue de iGrip, dénommée teleGrip, permet la réalisation de téléopérations. Par contre la mise en oeuvre d'un tel environnement reste relativement lourde. D'autre part, la dernière version de WorkingModel permet la manipulation directe des modèles simulés avec une souris conventionnelle. Mais ce programme n'est pas spécifiquement dédié à la robotique et donc au pilotage de robots. De plus le mode d'interaction reste relativement pauvre par rapport à l'utilisation d'environnements virtuels.

Finalement il est possible de relever plusieurs avantages à la création d'un programme modulaire spécifique (comme *VirtualRobot* dédié à l'analyse et au pilotage des robots manipulateurs) par rapport à des logiciels d'ordre général (comme les différents produits commerciaux présentés):

- l'exploitation des particularités d'une application permet de réaliser un programme mieux adapté et plus rapide (critère décisif pour une application en temps réel),
- un programme à caractère général est difficilement modifiable et surtout adaptable à des tâches spécifiques (par exemple communication avec un système de vision externe qui fournit des informations sur le monde où le robot évolue),
- l'utilisation d'un programme généraliste de simulation restreint les possibilités d'intégration d'une simulation particulière dans une autre application (par exemple intégrer le pilotage d'un bras de robot dans un environnement de téléopération existant pour contrôler un robot mobile).

## 2.4 Résumé

Ce chapitre a présenté dans un premier temps les différentes techniques de définition de tâche pour les robots utilisés dans l'industrie (programmation en-ligne et hors-ligne) ainsi que pour ceux employés comme outil de coopération (téléopération); le projet «CINEGEN» est destiné à fournir une interface pour la téléopération de robots à travers un environnement virtuel.

La deuxième partie présente les différentes méthodes possibles pour résoudre le problème complexe de la cinématique des robots. On en déduit que seule une approche basée sur la linéarisation du modèle est adaptable aux besoins du projet «CINEGEN».

Enfin, la dernière partie de ce chapitre présente quelques logiciels de simulation utilisés dans l'industrie ainsi que dans les laboratoires de recherche, afin de montrer leurs modes de fonctionnement ainsi que leurs particularités. Si les outils du commerce sont très puissants pour l'analyse poussée des mécanismes, ils sont mal adaptés au type d'interaction homme-robot que propose le projet «CINEGEN».

Le chapitre suivant détaille les outils mathématiques nécessaires pour la résolution du modèle direct et inverse des robots. Ceci permet de présenter dans la deuxième partie du chapitre la méthode de résolution générale et automatisable proposée dans le cadre de ce projet.

CHAPITRE

# 3

# CONCEPTS THÉORIQUES

La réalisation d'un simulateur qui permet de modéliser des structures mécaniques articulées de type quelconque en temps réel, ceci à partir d'un simple fichier de description, nécessite de combiner plusieurs concepts mathématiques utilisés en robotique. La synthèse des ces outils a abouti sur des extensions qui ont permis l'élaboration d'une description de robots originale et d'un "résolveur" de contraintes général.

L'étude de la cinématique des robots existe depuis le début des années 1970 avec un développement spécialement important dans les années 80. Une mesure des recherches dans ce domaine peut-être évaluée par la centaine de thèses portant spécifiquement sur le sujet et les innombrables articles dont les plus importants sont recensés dans [Gogu 97]. Il existe donc de nombreux outils et formalismes mis au point dans le cadre de la modélisation et commande des robots. Il faut toutefois noter que la grande majorité des ouvrages sur la robotique ne prennent en compte que l'étude de la cinématique des robots sériels, car ce sont les plus utilisés et les mieux maîtrisés dans le monde industriel. La discussion des robots parallèles se fait souvent de manière indépendante de la grande famille des robots sériels en utilisant spécifiquement leurs particularités. Le projet «CINEGEN» permet de traiter ces deux familles de façon unifiée.

Ce chapitre décrit d'abord la théorie sous-jacente au problème de la modélisation des robots ainsi que les solutions retenues pour y répondre. Un accent particulier est mis sur les avantages du formalisme utilisé dans le cadre du projet. La suite du chapitre présente la méthode originale élaborée pour résoudre le problème de la cinématique des robots généraux, à l'aide de la matrice jacobienne augmentée et de son inversion, permettant la résolution des contraintes.

## 3.1 Modélisation des robots

Le problème principal dans le projet «CINEGEN» est de trouver une relation entre des consignes données dans l'espace cartésien de la tâche et les *postures* des éléments du robot dans l'espace articulaire. Cette relation, qui doit être

générée automatiquement, permet de fournir une interface masquant le problème mathématique à l'utilisateur pour manipuler de manière interactive un robot.

Dans la suite nous parlerons de chaîne cinématique plutôt que de robot. En effet, l'étude qui suit fait plusieurs hypothèses simplificatrices sur le comportement d'un robot:

- chaque corps composant le robot est assimilé à un solide indéformable,
- chaque articulation (ou joint) entre les différents corps est considérée comme parfaite, sans jeu ni frottement,
- chaque articulation ne possède qu'un seul degré de liberté, les articulations de plus haut niveau (cardan, etc.) pouvant être décomposées en articulations élémentaires.

Ces hypothèses permettent d'obtenir une modélisation des robots (un modèle complet du comportement réel d'un robot n'est pas réalisable). Ces modèles "simplifiés" permettent par là même de résoudre les équations qui les représentent. Une hypothèse supplémentaire peut-être posée dans le cadre de ce projet:

- les masses des éléments du robot n'interviennent pas: le modèle dynamique du robot n'est pas pris en compte.

Si ces hypothèses sont réductrices quant à la modélisation d'un robot réel, elles conduisent à obtenir un modèle valide suffisamment proche du comportement "visible" du robot, et donc apte à fournir un comportement acceptable pour une interaction avec l'utilisateur.

### 3.1.1 Modèle géométrique

Pour le chercheur, la première interrogation face à un robot est "comment calculer sa position". Plus précisément:

- "où va se trouver l'outil du robot si chaque articulation possède tel angle (moteur rotatif) ou allongement (actionneur linéaire)": modèle géométrique direct;
- "comment trouver les valeurs de chaque articulation pour que l'outil du robot atteigne tel point de l'espace (position et orientation)": modèle géométrique inverse.

Le modèle géométrique du robot (qu'il soit direct ou inverse) est utile car il apporte une vision "globale" du comportement d'un robot. Toutefois, il reste limité pour le contrôle réel d'un robot. En effet, il correspond à un contrôle "point à point", qui bien qu'encore largement utilisé dans le monde industriel, répond mal au problème d'un déplacement entre deux points relativement distants. Il est nécessaire alors d'introduire les paramètres de vitesse, voire de dynamique du robot pour obtenir un contrôle du comportement plus efficace. Le contrôle en position n'est donc qu'un modèle restreint par rapport à la réalité robotique.

D'autre part, la génération et résolution du modèle géométrique comporte encore un nombre de problèmes non résolus à l'heure actuelle:

- Le modèle géométrique direct des robots sériels qui est aisé à obtenir fait intervenir des équations fortement non linéaires, comportant des fonctions trigonométriques. Par conséquent l'obtention du modèle géométrique inverse (cf. Tableau 3-1) est forcément complexe, voire impossible.

**Tableau 3-1:** *Modèles géométriques inverses pour les robots sériels*

| | **Solution analytique existante** |
|---|---|
| 4 d.d.l. | Pour toute architecture† |
| 5 d.d.l. | Partielle‡: seulement pour architectures comprenant deux axes d'articulations parallèles ou concourants |
| 6 d.d.l. | Non (uniquement pour des architectures très spécifiques comme 3ddl + poignet par exemple) |
| > 6 d.d.l. | Non (uniquement pour des structures sérielles particulières possédant par exemple plusieurs axes d'articulations parallèles) |

†. [Manseur 92a]
‡. [Manseur 92b]

- Le modèle géométrique inverse des robots parallèles (cf. Tableau 3-2) est dans certains cas facilement calculable, mais le modèle direct reste un challenge de nouveau par sa forte non-linéarité (ou de par le très haut degré des équations polynomiales pouvant être utilisées pour représenter le modèle du robot).

**Tableau 3-2:** *Modèles géométriques pour robots spéciaux*

| | **Modèle direct** | **Modèle inverse** |
|---|---|---|
| Robots complètement parallèles | existant pour plateforme de Gough-Stewart† | trivial (équations découplées) |
| Robots parallèles complexes (plusieurs chaînes couplées) | existant pour géométries spécifiques | compliqué |
| Robots hybrides | Pas dans le cas général | Pas dans le cas général |

†. [Husty 94]

Au vu des deux précédents tableaux, il est clair que les modèles géométriques (direct ou inverse) ne sont pas solvables analytiquement dans le cas général. Quand à leur résolution numérique, elle se heurte au problème des solutions multiples: l'unicité du modèle inverse n'est presque jamais garantie. La plupart des robots peuvent en effet atteindre une *posture* donnée de l'espace avec plusieurs *configurations* différentes.

### 3.1.2 Modèle cinématique

Le modèle cinématique est, littéralement, un modèle des vitesses. Il exprime les relations entre les vitesses articulaires de chaque joint et les vitesses cartésiennes d'un point de la chaîne cinématique, généralement l'organe terminal. Ce modèle est donc un modèle par accroissements infinitésimaux: chaque variation élémentaire de la valeur d'une articulation implique une

variation de position de l'organe terminal, et inversement. Lorsque ces variations infinitésimales sont exprimées par rapport au temps, on peut les considérer comme des vitesses.

Le modèle cinématique permet donc non seulement de compléter éventuellement le modèle géométrique en tenant compte des vitesses, mais aussi de remplacer le modèle géométrique: en agissant par accroissements successifs, on peut se déplacer d'un point donné à un autre.

Le modèle cinématique possède une propriété essentielle: il est une différentiation du modèle géométrique. Il est donc une linéarisation du système d'équations non linéaires représentant le modèle géométrique. Par conséquent on peut toujours facilement obtenir les transformations inverses puisqu'elles proviennent de l'inversion d'un problème linéaire! Toutefois le modèle cinématique comporte aussi des inconvénients:

1) la non-unicité du modèle géométrique inverse implique qu'il existe plusieurs "chemins" pour se rendre d'un point à un autre,
2) le traitement par incrément peut amener à des imprécisions,
3) des singularités, mécaniques et/ou mathématiques apparaissent.

Dans le cadre spécifique de «CINEGEN» ces inconvénients sont évités ou contournés de la manière suivante:

1) Le problème des multiples solutions du modèle inverse n'intervient pas. En effet, le robot possède une configuration initiale connue, et se rend à une autre position à partir de celle-ci. Il ne sert à rien de calculer des configurations de robots qui seraient impossibles à atteindre depuis la position courante.
2) Une haute précision des solutions obtenues n'est pas nécessaire puisque il suffit de fournir à l'utilisateur une vision globale, et que «CINEGEN» ne prétend pas remplacer le contrôleur final de chaque robot particulier. D'autre part, le calcul des accroissements est chaque fois effectué à partir d'une nouvelle configuration exacte du robot.
3) Quand au problème des singularités, il existe plusieurs méthodes mathématiques pour les traiter ou éviter.

Pour ces différentes raisons (linéarité du problème, pas de limitation dues à l'unicité, ni à la précision), le modèle cinématique s'impose comme outil pour le projet «CINEGEN».

### 3.1.3 La matrice jacobienne

L'outil principalement utilisé pour traiter le problème de la cinématique des robots est la matrice jacobienne. Elle représente un opérateur permettant de lier les vitesses des corps d'un robot exprimées dans différents espaces vectoriels.

Considérons le modèle géométrique d'un robot possédant $m$ degrés de liberté évoluant dans un espace à $n$ dimensions ($m$ et $n$ indépendants):

$$\begin{aligned} x_1 &= f_1(q_1, q_2, \ldots, q_m) \\ x_2 &= f_2(q_1, q_2, \ldots, q_m) \\ &\ldots \\ x_n &= f_n(q_1, q_2, \ldots, q_m) \end{aligned} \tag{3.1}$$

où **x** représente le vecteur de la position de l'organe terminal dans l'espace de la tâche, et **q** le vecteur des coordonnées articulaires.

On peut simplifier l'écriture en mettant (3.1) sous forme vectorielle:

$$\mathbf{x} = \mathbf{f}(\mathbf{q}) \tag{3.2}$$

Si maintenant nous différentions l'équation (3.1), nous obtenons:

$$\begin{aligned} \delta x_1 &= \frac{\partial f_1}{\partial q_1}\delta q_1 + \frac{\partial f_1}{\partial q_2}\delta q_2 + \ldots + \frac{\partial f_1}{\partial q_m}\delta q_m \\ \delta x_2 &= \frac{\partial f_2}{\partial q_1}\delta q_1 + \frac{\partial f_2}{\partial q_2}\delta q_2 + \ldots + \frac{\partial f_2}{\partial q_m}\delta q_m \\ &\ldots \\ \delta x_n &= \frac{\partial f_n}{\partial q_1}\delta q_1 + \frac{\partial f_n}{\partial q_2}\delta q_2 + \ldots + \frac{\partial f_n}{\partial q_m}\delta q_m \end{aligned} \tag{3.3}$$

ou sous forme condensée:

$$\delta\mathbf{x} = \frac{\partial \mathbf{f}}{\partial \mathbf{q}} \cdot \delta\mathbf{q} \tag{3.4}$$

Nous posons alors comme définition de la matrice jacobienne J:

$$\left[J(\mathbf{q})\right] \equiv \left[\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right] \tag{3.5}$$

En divisant les deux côtés de l'équation (3.4) par l'élément différentiel de temps, on peut voir la matrice jacobienne comme l'opérateur reliant les vitesses cartésiennes $\dot{\mathbf{x}}$ aux vitesses articulaires $\dot{\mathbf{q}}$.

Si les fonctions $f_1 \ldots f_n$ sont non linéaires, alors leurs dérivées partielles sont fonction des $q_i$. La matrice jacobienne est donc un opérateur linéaire dépendant de la position instantanée du robot.

Le modèle direct qui exprime les vitesses cartésiennes en fonction des vitesses articulaires s'écrit alors:

$$\left[\dot{x}\right] = \left[J\right] \cdot \left[\dot{q}\right] \tag{3.6}$$

Le modèle cinématique inverse est donc représenté par:

$$\left[\dot{q}\right] = \left[J\right]^{-1} \cdot \left[\dot{x}\right] \tag{3.7}$$

La Figure 3-1 représente schématiquement la composition de la matrice jacobienne pour un robot simplifié. La matrice possède trois lignes qui correspondent aux trois degrés de liberté de l'espace de la tâche, et trois colonnes correspondant aux trois degrés de liberté du robot. Les vecteurs de translation différentielle et de rotation différentielle sont respectivement notés $\mathbf{d}_n$ et $\underline{\delta}_n$.

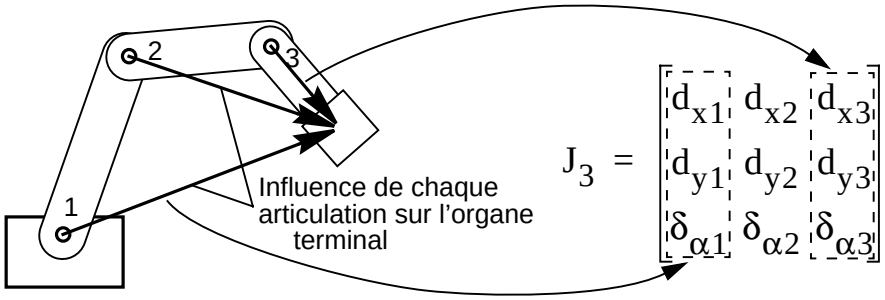



**Figure 3-1:** *Matrice jacobienne d'un manipulateur planaire à trois degrés de liberté*

Dans le cas des robots sériels, la matrice jacobienne est facilement calculable car chaque articulation produit une influence directe sur l'organe terminal. Dans le cas de structures parallèles, il faut modifier la matrice jacobienne afin d'exprimer des relations supplémentaires. Il faut augmenter la matrice jacobienne qui sera alors composée des:

- relations entre les variations de la plateforme mobile et la chaîne cinématique principale,
- relations entre les influences des articulations actives (moteurs) des autres chaînes cinématiques sur les articulations passives de la chaîne principale.

Le procédé utilisé dans le projet «CINEGEN» pour résoudre la cinématique d'un robot dans le cas général est directement tiré de la relation (3.7):

1) la matrice jacobienne augmentée est construite en fonction de la position actuelle du robot suivant la méthode de la Section 3.4.2
2) la matrice jacobienne augmentée est inversée en utilisant sa pseudo-inverse comme le décrit la Section 3.4.3.
3) en multipliant cette inverse par le vecteur déplacement cartésien désiré, on obtient les variations à appliquer sur chaque articulation.

Les deux prochaines sections posent les bases nécessaires à la méthode utilisée dans le projet «CINEGEN» qui sera décrite dans la Section 3.4.

## 3.2 Formalismes de description des robots

Pour pouvoir trouver les modèles géométrique ou cinématique d'un robot, il faut pouvoir décrire sa structure en termes mathématiques. Cette description est essentiellement basée sur la position relative ou absolue des différents

corps du robot dans l'espace. Il est alors nécessaire de choisir des méthodes pour:

- décrire les transformations de coordonnées et leur composition afin de placer les différents corps dans l'espace,
- obtenir une notation cohérente et universelle pour paramétriser une structure articulée.

## 3.2.1 Transformation de coordonnées: matrice homogène

Il existe une grande variété d'outils pour représenter les mouvements des corps solides dans l'espace. On peut en trouver une description détaillée ainsi qu'une comparaison au niveau du nombre de calculs nécessaires dans [Gogu 96]. Par exemple, les rotations de l'espace peuvent être effectuées avec des matrices orthogonales 3x3, des matrices unitaires complexes 2x2, des quaternions ou des matrices exponentielles.

Dans le projet «CINEGEN» nous avons adopté les matrices de transformation homogène 4x4 qui permettent la représentation de n'importe quelle transformation: translation, rotation, mouvement hélicoïdal (vis). De plus les matrices homogènes 4x4 nécessitent un nombre minimum d'opérations élémentaires à effectuer lors d'une transformation. Etant donné que les articulations des robots sont essentiellement à caractère rotatif ou linéaire, les matrices homogènes les représentent adéquatement, même si un mouvement est composé d'une translation et d'une rotation suivant un axe différent.

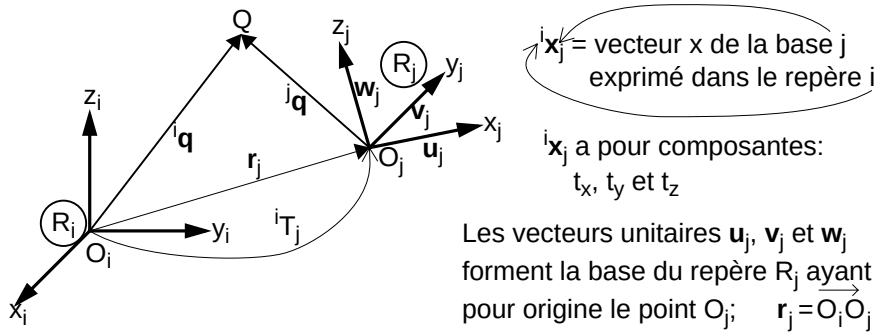



**Figure 3-2:** *Définition des repères pour une transformation de l'espace*

La matrice homogène $^iT_j$ qui représente le repère $R_j$ par rapport au repère $R_i$ suivant les notations de la Figure 3-2 est définie comme suit:

$$^iT_j = \begin{bmatrix} u_x & v_x & w_x & r_x \\ u_y & v_y & w_y & r_y \\ u_z & v_z & w_z & r_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} ^i\mathbf{u}_j & ^i\mathbf{v}_j & ^i\mathbf{w}_j & ^i\mathbf{r}_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.8)$$

Cette même matrice ${}^{i}T_{j}$ est l'opérateur qui permet de trouver les coordonnées d'un point ${}^{j}Q$ ou d'un vecteur ${}^{j}\mathbf{q}$ (exprimé initialement dans le repère $R_j$) dans le repère $R_i$: ${}^{i}\mathbf{q} = {}^{i}T_{j} \cdot {}^{j}\mathbf{q}$

## 3.2.2 Description des paramètres d'une chaîne articulée: Kleinfinger-Khalil

Des matrices homogènes seront donc utilisées dans ce projet pour modéliser les postures des éléments constituant un robot. Il faut de plus adopter une méthode pour assigner des repères aux différents composants du robot afin de décrire toute chaîne cinématique de manière uniforme. Il existe de nombreuses notations différentes pour décrire les chaînes articulées. Un inventaire de ces différentes notations a été synthétisé dans [Gogu 97].

Dans la suite de ce rapport, le terme "lien" d'un robot sera compris comme la combinaison d'une articulation et du corps rigide qui la relie à l'articulation suivante.

Nous présentons ici sommairement deux méthodes classiques pour montrer les deux approches différentes quant à l'assignation des repères sur chaque *lien* du robot. Puis dans un troisième paragraphe nous détaillerons la méthode adoptée pour ce projet, proposée par Kleinfinger dans sa thèse [Kleinfinger 86].

### 3.2.2.1 Notations de Denavit-Hartenberg

Denavit et Hartenberg ont proposé une méthode reposant sur l'assignation d'un repère unique pour chaque lien. La Figure 3-3 présente l'utilisation de cette notation pour deux liens successifs. L'axe $z_i$ du repère est concourant avec l'axe de l'articulation i. Quand à l'axe $x_i$, il est sur la droite perpendiculaire aux axes $z_{i-1}$ et $z_i$. Quatre paramètres sont alors utilisés pour décrire la forme géométrique d'un lien et sa position par rapport au lien précédent.

Le Tableau 3-3 montre les principales caractéristiques de la notation de Denavit-Hartenberg qui ne fonctionne que pour des chaînes cinématiques sérielles (pour des chaînes arborescentes des ambiguïtés apparaissent).

**Tableau 3-3:** *Evaluation de la notation Denavit-Hartenberg*

| Avantages | Inconvénients |
|---|---|
| Seulement 4 paramètres | Non utilisable pour les chaînes arborescentes ou bouclées |
| Nombreux robots existant décrits avec cette méthode† | Décalage dans les indices (source d'erreur) |

†. Surtout dans le monde académique.

Cette notation, apparue très tôt dans le domaine de la robotique est encore largement utilisée par la communauté scientifique pour décrire des robots en vue de leur analyse et/ou modélisation. Quelques variantes relativement proches sont aussi courantes, comme par exemple la notation de Paul [Paul 83]. Cette notation se différencie essentiellement de celle de Denavit-Hartenberg par l'assignation des paramètres relativement aux liens (décalage des indices).

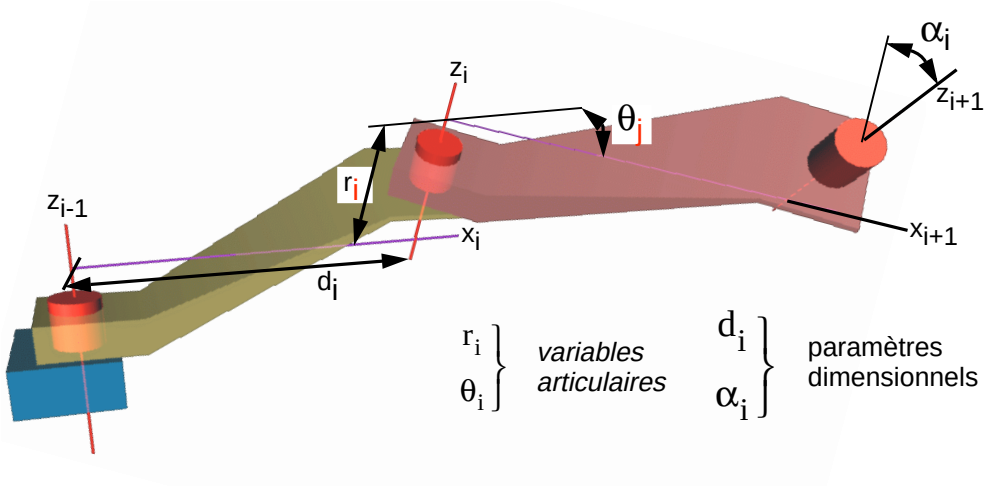



**Figure 3-3:** *Définition des paramètres suivant Denavit-Hartenberg*

### 3.2.2.2 Notations de Sheth-Uiker

Pour pouvoir décrire des chaînes cinématiques complexes (arborescentes et bouclées), Sheth et Uicker ont proposé une nouvelle notation basée sur l'assignation de deux repères par lien. La Figure 3-4 présente cette notation pour deux liens dans une vue "éclatée" afin de visualiser correctement les différents repères. Chaque lien i se voit attribué un repère coïncidant avec son articulation i, et un deuxième coïncidant avec l'articulation i+1.

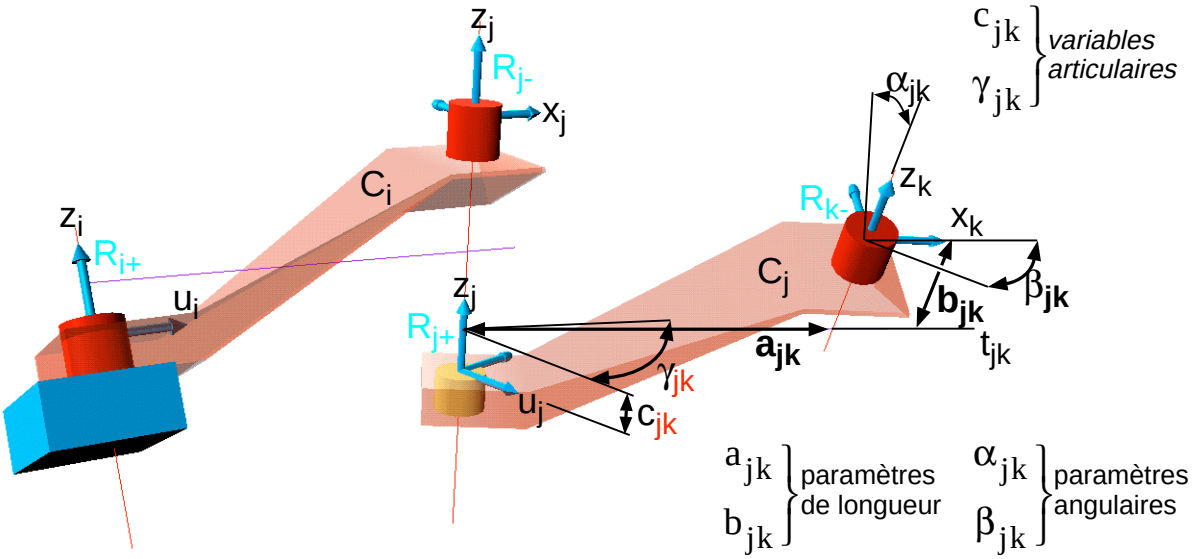



**Figure 3-4:** *Définition des paramètres suivant Sheth-Uicker*

Ainsi chaque lien est modélisé par deux matrices dont les paramètres principaux sont montrés dans la Figure 3-4:

1) une matrice décrivant la forme géométrique du lien et contenant donc des paramètres constants;

2) une matrice décrivant le déplacement du lien par rapport au précédent, contenant donc des paramètres variables.

Le Tableau 3-4 résume les principales caractéristiques de la méthode Sheth-Uicker par rapport au projet «CINEGEN».

**Tableau 3-4:** *Evaluation de la notation Sheth-Uicker*

| Avantages | Inconvénients |
|---|---|
| Méthode complètement générale (non limitée aux robots classiques) | Utilisation de deux repères par lien donc complexe à mettre en oeuvre |
| Modélisation de tout type de structure | Nombre élevé de paramètres |

### 3.2.2.3 Notations de Kleinfinger-Khalil

Kleinfinger et Khalil ont proposé une méthode pour la description des chaînes cinématiques sérielles, arborescentes ou bouclées. La Figure 3-5 présente l'assignation des repères pour cette notation et définit les paramètres nécessaires. Cette figure montre un corps $C_j$ avec articulation rotoïde et un corps $C_k$ avec une articulation prismatique. Ces deux corps sont reliés par leur articulation à un même corps rigide $C_i$ qui forme un embranchement.

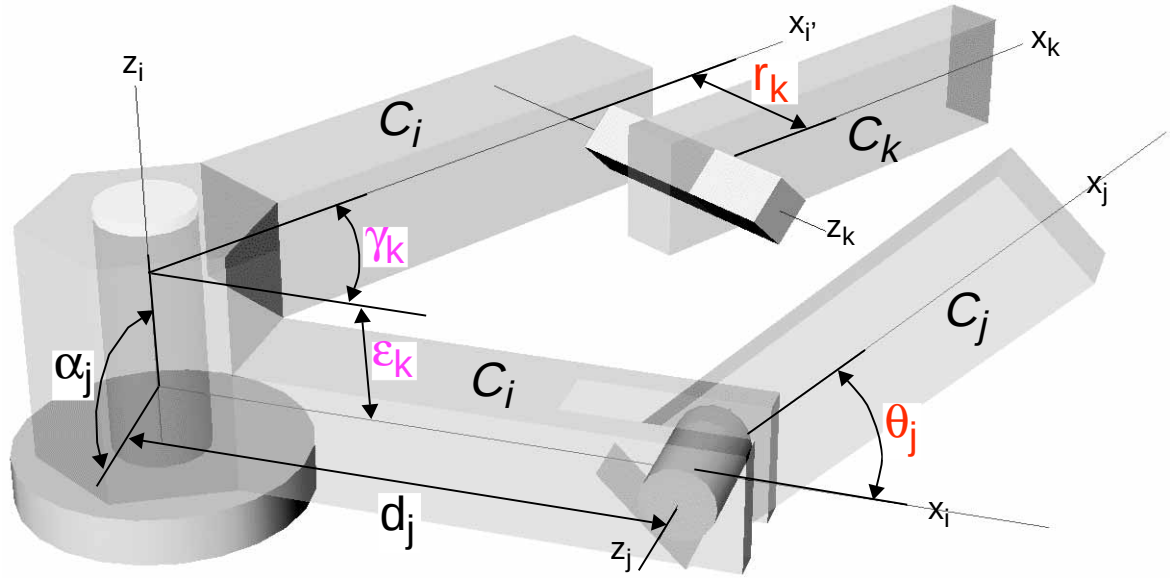



**Figure 3-5:** *Définition des paramètres suivant Kleinfinger-Khalil*

Cette notation utilise un repère par lien, mais utilise six paramètres (donc deux de plus que Denavit-Hartenberg) pour la modélisation de chaque lien. Comme les autres méthodes, deux paramètres représentent les variables articulaires (prismatique ou rotoïde) du lien. Les quatre autres permettent la description de la géométrie du lien. En fait dans la plupart des cas, seulement deux paramètres fixes sont nécessaires (les deux autres restant nuls) et la notation se rapproche de celle de Denavit-Hartenberg, avec toutefois une meilleure cohérence dans l'attribution des indices. Dans les cas complexes et aux embranchements, les deux paramètres fixes supplémentaires sont utilisés.

Le Tableau 3-5 résume les conventions pour l'assignation des repères et la définition des paramètres et la Figure 3-5 montre l'emploi des paramètres

Kleinfinger-Khalil pour une structure avec embranchement et les deux types d'articulations élémentaires.

**Tableau 3-5:** *Notations Kleinfinger-Khalil utilisée dans les fichiers MAD*†

| Règles de construction: | | | $Z_j$ = axe de l'articulation j supportant le corps j<br>$X_j$ = axe ⊥ ($Z_j$ & $Z_{j+1}$) / $X_i$' = axe ⊥ ($Z_i$ & $Z_k$) | | | |
|---|---|---|---|---|---|---|
| **Variable** | **Nom MAD** | **Axe Ref.*** | **De l'axe** | **A l'axe** | **Remarque** | **Type de paramètre** |
| $\theta_j$ | theta | $z_j$ | $x_{j-1}$ | $x_j$ | pour les articulations rotoïdes | variables d'articulation |
| $r_j$ | rpara | $z_j$ | $x_{j-1}$ | $x_j$ | pour les articulations prismatiques | |
| $\alpha_j$ | alpha | $x_{j-1}$ | $z_{j-1}$ | $z_j$ | angle entre les axes de deux articulations | paramètres fixes |
| $d_j$ | dpara | $x_{j-1}$ | $z_{j-1}$ | $z_j$ | distance entre les axes de deux articulations | |
| $\gamma_j$ | gamma | $z_i$ | $x_i$ | $x_i$' | angle spécial pour les embranchements | paramètres d' embranchement |
| $\varepsilon_j$ | epsil | $z_i$ | $x_i$ | $x_i$' | distance spéciale pour les embranchements | |
| * Axe de Référence: | | ou | axe autour duquel l'angle est mesuré<br>axe le long duquel la distance est mesurée | | | |

†. Notations pour un lien composé d'une articulation j et d'un corps Cj. Dans le cas où chaque lien ne supporte qu'un autre lien, on numérote les liens j-1, j, j+1. Dans le cas où 2 corps Cj et Ck sont supportés par un même corps Ci, on aura un premier axe $x_i$ perpendiculaire à $z_i$ et $z_j$ puis un second axe $x_i$' perpendiculaire à $z_i$ et $z_k$.

Les principaux avantages de la méthode Kleinfinger-Khalil pour son utilisation dans le projet «CINEGEN» sont les suivant:

- modélisation de toute structure (sérielle, arborescente et bouclée),
- aussi simple que la notation de Denavit-Hartenberg dans le cas des robots sériels,
- moins complexe à utiliser que la notation Sheth-Uicker,
- bonne cohérence entre les indices des corps, des articulations et des paramètres.

La matrice de transformation associée à la modélisation Kleinfinger-Khalil permettant de passer du repère $R_i$ du lien i au repère $R_j$ du lien j est alors obtenue par composition des transformations liées aux six paramètres:

$$ {}^{i}T_j = R(z_j, \gamma_j) \cdot T(z_j, \varepsilon_j) \cdot R(x_{j-1}, \alpha_j) \cdot T(x_{j-1}, d_j) \cdot R(z_j, \theta_j) \cdot T(z_j, r_j), $$

R(u,q) étant la matrice représentant une rotation d'un angle q autour de l'axe u et T(v,t) étant la matrice représentant une translation d'une distance t le long de l'axe v.

D'où (cf. [Dombre 88] pour les calculs):

$$ {}^{i}T_j = \begin{bmatrix} C\gamma_j C\theta_j - S\gamma_j C\alpha_j S\theta_j & -C\gamma_j S\theta_j - S\gamma_j C\alpha_j C\theta_j & S\gamma_j S\alpha_j & d_j C\gamma_j + r_j S\gamma_j S\alpha_j \\ S\gamma_j C\theta_j + C\gamma_j C\alpha_j S\theta_j & -S\gamma_j S\theta_j + C\gamma_j C\alpha_j C\theta_j & -C\gamma_j S\alpha_j & d_j S\gamma_j + r_j C\gamma_j S\alpha_j \\ S\alpha_j S\theta_j & S\alpha_j C\theta_j & C\alpha_j & r_j C\alpha_j + \varepsilon_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.9) $$

avec:

$$C\alpha_j = \cos\alpha_j \qquad C\theta_j = \cos\theta_j \qquad C\gamma_j = \cos\gamma_j$$

$$S\alpha_j = \sin\alpha_j \qquad S\theta_j = \sin\theta_j \qquad S\gamma_j = \cos\gamma_j$$

En observant la matrice (3.9), on remarque qu'il faut effectuer 6 fonctions trigonométriques[1] et 19 multiplications pour l'obtenir. Pour une structure cinématique à chaîne sérielle, les paramètres $\gamma_i$ et $\varepsilon_i$ s'annulent: il ne reste que 4 fonctions trigonométriques et 6 multiplications à calculer[2].

Certaines notations comme celle du TCS (Travelling Coordinate System) proposée par Gogu-Coiffet-Barraco [Gogu 97] permettent de réduire ce nombre d'opérations élémentaires. Toutefois, si le gain peut-être significatif dans une approche symbolique, il n'est pas suffisant dans le cas d'une résolution numérique (par rapport aux autres opérations coûteuses en temps comme l'inversion d'une matrice).

La méthode Kleinfinger-Khalil présente un optimum entre la complexité des calculs occasionnés au programme et la facilité de mise en oeuvre pour l'utilisateur.

## 3.3 Formation de la matrice jacobienne

Il existe plusieurs méthodes pour calculer la matrice jacobienne définie dans la relation (3.5). Si on écarte les méthodes de calcul symbolique permettant de dériver les équations du modèle géométrique direct, il faut calculer les influences de chaque articulation sur l'organe terminal comme le montre la Figure 3-1. Cette approche, très répandue pour le calcul cinématique, peut s'effectuer suivant différentes moyens qui sont plus ou moins efficaces au niveau des algorithmes et du nombre d'opérations nécessaires. Ces méthodes se caractérisent par le repère dans lequel la matrice jacobienne est exprimée ainsi que par le corps auquel elle correspond, c'est-à-dire par le choix de i et j dans le calcul de ${}^{i}J_{j}$: matrice jacobienne du corps j exprimée dans repère i.

La plupart du temps, on calcule la matrice jacobienne relative à l'organe terminal E. La matrice jacobienne est aussi souvent exprimée dans le repère de base $R_0$ que dans le repère de l'organe terminal $R_E$. Toutefois suivant [Orin 84], le calcul de ${}^{0}J_{E}$ nécessite un plus grand nombre d'opérations mathématiques que le calcul de ${}^{E}J_{E}$ (matrice jacobienne de l'organe terminal exprimée dans le repère de l'organe terminal).

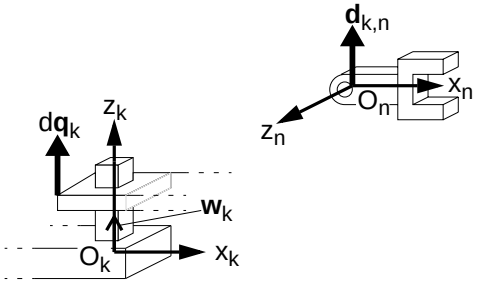
### 3.3.1 Calcul de la matrice jacobienne vectorielle $J_n$.

Cette section décrit le calcul de la matrice jacobienne relative au corps n en composant les influences des vecteurs de translation et rotation différentielle $\mathbf{d}_n$ et $\underline{\delta}_n$.

1. Si la structure géométrique ne change pas, 4 de ces 6 fonctions trigonométriques sont des constantes.
2. Le nombre d'opération à effectuer dans ce cas est équivalent à celui impliqué par la méthode Denavit-Hartenberg.

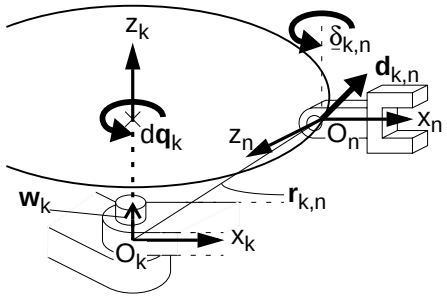
Considérons l'influence de l'articulation k et notons $\mathbf{d}_{k,n}$ et $\underline{\delta}_{k,n}$ les déplacements au niveau du repère lié au corps n, dû à l'articulation k. En utilisant les notations de la relation (3.8), on peut calculer $\mathbf{d}_{k,n}$ et $\underline{\delta}_{k,n}$ en considérant séparément les cas d'une articulation prismatique et d'une articulation rotoïde.

*articulation prismatique*




$$\begin{cases} \mathbf{d}_{k,\,n} = \mathbf{w}_k \cdot \mathrm{d}\mathbf{q}_k \\ \underline{\delta}_{k,\,n} = \mathbf{0} \end{cases} \qquad (3.10)$$

*articulation rotoïde*




$$\begin{cases} \mathbf{d}_{k,\,n} = (\mathbf{w}_k \times \mathbf{r}_{k,\,n}) \cdot \mathrm{d}\mathbf{q}_k \\ \underline{\delta}_{k,\,n} = \mathbf{w}_k \cdot \mathrm{d}\mathbf{q}_k \end{cases} \qquad (3.11)$$

Grâce au théorème de la composition des vitesses, on peut sommer toutes les contributions élémentaires de chaque articulation afin d'obtenir les vecteurs finaux de rotation et translation différentielles.

*articulation prismatique*

$$\begin{cases} \mathbf{d}_n = \sum_{k=1}^{n} \mathbf{w}_k \cdot \mathrm{d}\mathbf{q}_k \\ \underline{\delta}_n = \mathbf{0} \end{cases} \qquad (3.12)$$

*articulation rotoïde*

$$\begin{cases} \mathbf{d}_{k,\,n} = \sum_{k=1}^{n} (\mathbf{w}_k \times \mathbf{r}_{k,\,n}) \cdot \mathrm{d}\mathbf{q}_k \\ \underline{\delta}_{k,\,n} = \sum_{k=1}^{n} \mathbf{w}_k \cdot \mathrm{d}\mathbf{q}_k \end{cases} \qquad (3.13)$$

En écrivant les équations (3.12) et (3.13) sous forme matricielle, on obtient la matrice jacobienne vectorielle formée de n colonnes de vecteurs. Cette matrice transforme les déplacements élémentaires aux articulations en déplacements élémentaires au niveau de l'organe terminal:

$$\begin{bmatrix} \mathbf{d}_n \\ \underline{\delta}_n \end{bmatrix} = \begin{bmatrix} \mathbf{j}_1 \cdots \mathbf{j}_n \end{bmatrix} \cdot \begin{bmatrix} \mathrm{d}\mathbf{q}_1 \\ \cdots \\ \mathrm{d}\mathbf{q}_n \end{bmatrix} = \mathbf{J}_n \cdot \mathrm{d}\mathbf{q} \qquad (3.14)$$

où les vecteurs colonnes $\mathbf{j}_k$ s'écrivent respectivement:

*articulation prismatique*

$$\mathbf{j}_k = \begin{bmatrix} \mathbf{w}_k \\ \mathbf{0} \end{bmatrix} \qquad (3.15)$$

*articulation rotoïde*

$$\mathbf{j}_k = \begin{bmatrix} \mathbf{w}_k \times \mathbf{r}_{k,\,n} \\ \mathbf{w}_k \end{bmatrix} \qquad (3.16)$$

### 3.3.2 Calcul de la matrice jacobienne scalaire ${}^{n}J_{n}$.

Pour obtenir la matrice jacobienne ${}^{r}J_{n}$ sous forme scalaire exprimée dans un repère quelconque $R_r$, il faut projeter les vecteurs $\mathbf{j}_k$ dans le repère $R_r$.

Plus spécifiquement, si on projette $\mathbf{J}_n$ dans le repère $R_n$, on obtient les ${}^{n}j_k$ qui forment la matrice jacobienne ${}^{n}J_{n}$:

$$ {}^{n}J_{n} = \left[ {}^{n}j_1 \ \cdots \ {}^{n}j_n \right] $$

En effectuant des simplifications classiques (cf. [Dombre 88] et[Gogu 97]) on obtient finalement les composantes de la matrice jacobienne en fonction de la matrice de transformation ${}^{k}T_{n}$ (notations suivant équation (3.8)):

**Tableau 3-6:** *Composition des colonnes de la matrice jacobienne ${}^{n}J_{n}$*

| **articulation prismatique** | **articulation rotoïde** |
|---|---|
| $${}^{n}j_k = \begin{bmatrix} \begin{bmatrix} u_z \\ v_z \\ w_z \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{bmatrix} \quad (3.17)$$ | $${}^{n}j_k = \begin{bmatrix} \begin{bmatrix} u_y \\ v_y \\ w_y \end{bmatrix} r_x - \begin{bmatrix} u_x \\ v_x \\ w_x \end{bmatrix} r_y \\ \begin{bmatrix} u_z \\ v_z \\ w_z \end{bmatrix} \end{bmatrix} \quad (3.18)$$ |
| les composants de **u**, **v**, **w** et **r** appartiennent à la matrice ${}^{k}T_{n}$ | |

Pour construire la matrice jacobienne ${}^{n}J_{n}$, il faut donc calculer toutes les matrices de transformation ${}^{k}T_{n}$ avec k variant de 1 à n. Ceci se fait par récursivité en partant du dernier lien:

$$ \begin{cases} {}^{n}T_{n} = I \\ {}^{k}T_{n} = {}^{k}T_{k+1} \cdot {}^{k+1}T_{n} \qquad k = n-1, n-2, \ldots, 1 \end{cases} \quad (3.19) $$

où I est la matrice identité et ${}^{k}T_{k+1}$ la matrice de transformation du lien k.

Il faut encore remarquer que la plupart du temps l'on effectue le calcul de la matrice jacobienne ${}^{E}J_{E}$ (plutôt que ${}^{n}J_{n}$) puisque l'on est généralement intéressé par les déplacements de l'organe terminal. Toutefois le raisonnement pour le calcul de la matrice jacobienne reste identique à la démarche présentée ci-dessus, à la seule différence que la première matrice de la relation de récurrence (3.19) devient ${}^{E}T_{n}$.

### 3.3.3 Inversion de la matrice jacobienne

L'inversion de la matrice jacobienne permet la calcul de la cinématique inverse des robots sériels suivant l'équation (3.7).

Il est évident qui si le modèle représenté par l'équation (3.7) est simple au niveau conceptuel, l'inversion dans la pratique de la matrice jacobienne n'est pas triviale. D'une part la matrice jacobienne est rarement carrée[3]: dans ces conditions son inverse n'existe pas. D'autre part, le calcul d'une inverse généralisée génère souvent des singularités qui vont produire des solutions ne possédant pas de sens physique.

Dans tous les cas où l'inverse $J^{-1}$ n'existe pas, on cherche à formuler une matrice inverse généralisée qui fournit une solution "acceptable" et utile au problème. Le résultat dépend alors du choix de l'inverse généralisée qui produira une approximation de la solution selon des critères à définir. Il existe donc une infinité d'inverses généralisées, mais la plus utilisée en robotique et la pseudo-inverse de Moore-Penrose ([Doty 93] et références citées). La pseudo-inverse Moore-Penrose de la matrice A sera notée $A^+$ et abrégée simplement "pseudo-inverse"[4] par la suite. Une méthode souvent utilisée pour obtenir une pseudo-inverse est la décomposition en valeurs singulières de la matrice à inverser. On peut montrer [Golub 96] que toute matrice A peut s'écrire comme le produit de trois matrices U, $\Sigma$ et V ayant les propriétés suivantes: U et V sont orthogonale, $\Sigma$ est une matrice diagonale.

$$\underset{mxn}{\left[ A \right]} = \underset{mxn}{\left[ U \right]} \cdot \underset{nxn}{\left[ \Sigma \right]} \cdot \underset{nxn}{\left[ V \right]}^T, \qquad \left[ \Sigma \right] = \begin{bmatrix} \sigma_1 & & \\ & \dots & \\ & & \sigma_n \end{bmatrix} \tag{3.20}$$

L'inverse de A s'écrit donc:

$$\underset{nxm}{\left[ A \right]^{-1}} = \underset{nxn}{\left[ V \right]} \cdot \underset{nxn}{\left[ \Sigma \right]^{-1}} \cdot \underset{nxm}{\left[ U \right]}^T, \quad \left[ \Sigma \right]^{-1} = \begin{bmatrix} 1/\sigma_1 & & \\ & \dots & \\ & & 1/\sigma_n \end{bmatrix} \tag{3.21}$$

Il est évident que lorsque des $\sigma_i$ s'annulent, l'inverse $A^{-1}$ n'est plus définie. Il est alors courant d'utiliser la pseudo-inverse $A^+$ pour laquelle la matrice $\Sigma^+$ est définie comme suit:

3. La matrice jacobienne est carrée uniquement pour les manipulateurs possédant 3 degrés de liberté évoluant dans le plan, et pour les manipulateurs avec 6 degrés de liberté évoluant dans l'espace.
4. La matrice $A^+$ est souvent simplement appelée "pseudo-inverse" dans la littérature, ce qui porte à confusion car une pseudo-inverse vérifie moins de propriétés que la pseudo-inverse de Moore-Penrose. Toutefois dans la suite de ce rapport nous nous rallions à cet abus de langage.

$$\left[\,\Sigma\,\right]^{+} = \begin{bmatrix} \sigma^{+}_{1} & & \\ & \cdots & \\ & & \sigma^{+}_{n} \end{bmatrix} \qquad \text{avec} \qquad \begin{cases} \sigma^{+}_{i} = 1/\sigma_i, & \text{pour } \sigma_i \neq 0 \\ \sigma^{+}_{i} = 0, & \text{pour } \sigma_i = 0 \end{cases} \tag{3.22}$$

Il n'est pas commun de poser $0 = \infty$, mais on peut démontrer [Press 95] que cette assertion permet, dans la cas où la matrice A est singulière, d'obtenir une solution qui est optimale au niveau des moindres carrés (la solution minimise la norme de l'erreur commise).

La solution pour obtenir la cinématique inverse d'un robot à partir de sa matrice jacobienne J est donc formulée en fonction de la pseudo-inverse $J^{+}$. La matrice $J^{+}$ est calculée à partir de la décomposition en valeurs singulières de J. Les relations (3.23) résument le modèle utilisé pour la cinématique inverse.

$$\begin{gathered} \text{si} \qquad \mathbf{J} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^{T} \qquad \text{alors} \\ \dot{\mathbf{q}} = \mathbf{J}^{+} \cdot \dot{\mathbf{x}} \qquad \text{avec} \qquad \mathbf{J}^{+} = \mathbf{V} \cdot \Sigma^{+} \cdot \mathbf{U}^{T} \end{gathered} \tag{3.23}$$

## 3.4 Résolution de la cinématique de structures quelconques

La Section 3.2 et la Section 3.3 ont rappelé les bases existantes dans la littérature pour étudier les robots, essentiellement à structure sérielle. Quand aux méthodes pour les robots parallèles [Merlet 97], elles utilisent les propriétés spécifiques de ces structures et ne sont donc pas appropriées pour les autres types de robots. Cette section montre comment résoudre la cinématique des structures quelconques à l'aide d'extensions des outils mathématiques décrits précédemment.

Pour ce faire il est d'abord nécessaire d'avoir une méthode simple et cohérente pour décrire les structures bouclées. Ensuite la construction de la matrice jacobienne augmentée servira à exprimer toutes les contraintes que le robot subit ou contient. Finalement l'inversion de cette matrice va permettre de calculer un modèle cinématique complet pour la structure.

### 3.4.1 Description de la structure d'une chaîne articulée

La notation de Kleinfinger-Khalil décrite dans la Section 3.2.2.3 permet de placer des repères sur toute chaîne articulée et d'obtenir une cohérence dans la définition des paramètres. Il faut de plus choisir un mode de description de la structure des chaînes articulées qui permet à l'utilisateur de coder la structure du robot de manière simple et efficace. Ce même codage sera utilisé par le programme qui calculera la cinématique de ces structures.

Un arbre est une collection de noeuds, tous issus d'une même racine. Chaque noeud possède un seul parent. Dans un graphe, les relations entre les noeuds peuvent être quelconques. Le graphe permet donc des topologies bouclées.

Dans «CINEGEN» tout robot sera représenté sous forme d'un graphe. Si l'on isole de ce graphe les noeuds qui représentent des liens, alors le graphe prend la structure particulière d'un arbre. L'ajout de contraintes supplémentaires transforme cet arbre en graphe avec des boucles.

Le choix de coder toute chaîne articulée comme une structure en arbre avec des contraintes additionnelles pour les boucles permet de:

- conserver une grande simplicité au niveau de la description (un seul paramètre est utile pour identifier le parent de chaque lien),
- rester cohérent avec l'utilisation de la matrice jacobienne augmentée qui sera utilisée pour la résolution cinématique,
- avoir un algorithme unique pour traiter la cinématique de toutes les catégories de chaînes articulées.

La Figure 3-6 montre comment sont représentées des structures sérielles, arborescentes ou bouclées par des arbres et arbres avec boucles (graphes).

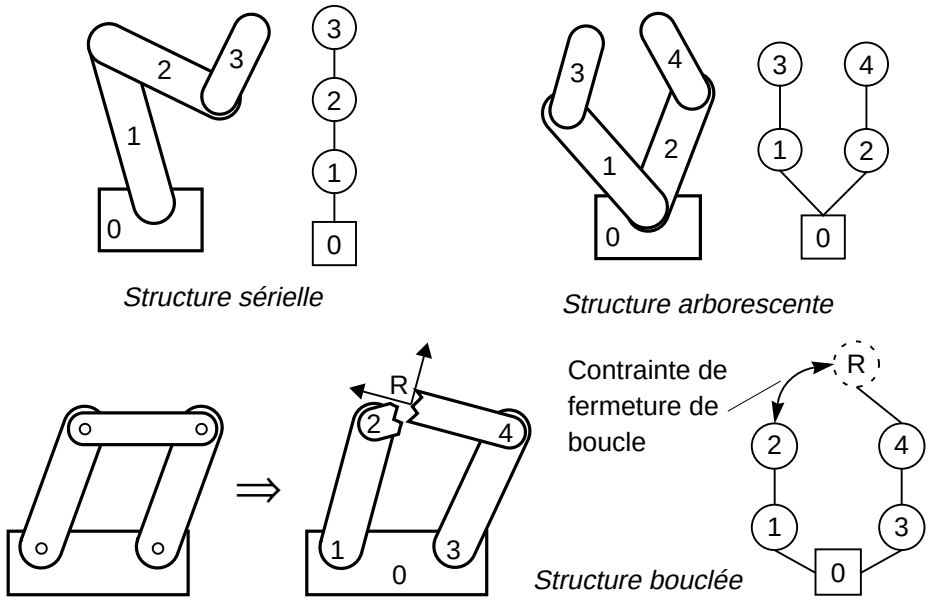



**Figure 3-6:** *Description des différents types de structures de chaînes articulées*

### 3.4.1.1 Structures sérielles et arborescentes

Une structure sérielle avec un seul organe terminal est simplement codée par un arbre à une seule branche dont la racine est la base du robot manipulateur.

Une structure comprenant plusieurs organes terminaux est codée par un arbre contenant autant de branches que la structure contient de chaînes. Les différents embranchements ne partent pas nécessairement de la racine mais peuvent avoir lieu depuis n'importe quel lien.

### 3.4.1.2 Structures bouclées

Une structure comportant des boucles est aussi représentée par un arbre: chaque boucle est divisée en deux branches et une contrainte est ajoutée entre

la terminaison de ces deux branches pour indiquer qu'elles forment une boucle.

#### *Division au niveau d'une articulation*

Certaines méthodes, [Kleinfinger 86] et [Dombre 88] entre autres, proposent de diviser la boucle au niveau d'une articulation. Cette méthode est surtout employée car elle permet un codage avec un formalisme unique: le formalisme de description d'un lien est aussi utilisé pour coder les terminaisons des deux sous-chaînes formant la boucle.

Pour ce faire, on ajoute une articulation virtuelle supplémentaire pour coder le corps de liaison qui ferme la boucle. Toutefois, la matrice de transformation associée à ce lien reste constante, comme si l'articulation était figée afin de ne pas introduire un degré de liberté supplémentaire.

#### *Division au niveau d'un corps*

Dans un but de clarté et d'efficacité, nous proposons de diviser une boucle au niveau d'un corps afin de ne pas introduire d'ambiguïté sur l'existence ou non d'une articulation supplémentaire.

La division d'une boucle s'effectue donc au niveau d'un corps quelconque du robot qui relie deux articulations appartenant à la boucle. Le corps est "cassé" en deux afin de décrire la structure comme un arbre, et la contrainte de fermeture de boucle spécifie que les deux morceaux doivent se joindre lors de la résolution cinématique. On obtient ainsi autant de corps que d'articulations. Dans l'exemple de la Figure 3-6, la structure bouclée physique possède quatre articulations, mais seulement trois corps les reliant puisque la structure est fermée. Le fait de scinder le corps supérieur en deux parties fait apparaître un quatrième corps virtuel dont il faut définir les dimensions à l'aide de repères supplémentaires.

Usuellement, on divise ce corps au niveau de l'une des articulations[5]. Ainsi il suffit de l'introduction d'un seul repère supplémentaire pour décrire la structure. On obtient donc un lien de taille nulle composé simplement d'une articulation (numéro 2 dans l'exemple) et un lien de la taille du corps original reliant les deux articulations. Ce dernier est formé symboliquement d'une articulation (numéro 4 dans l'exemple) et d'un repère (R) qui permettent de définir la transformation entre les deux articulations fermant la boucle.

Le choix du corps où la boucle va être divisée en deux est laissé à l'utilisateur. Il est toutefois conseillé d'équilibrer les deux branches formant la boucle afin de mieux répartir les calculs cinématiques.

### 3.4.2 Calcul de la matrice jacobienne augmentée

La matrice jacobienne ${}^{n}J_{n}$ de la Section 3.3.2 exprime les relations entres les vitesses cartésiennes du dernier lien d'un robot et les vitesses articulaires

5. Diviser un corps au niveau d'un articulation génère le même résultat que diviser la chaîne au niveau d'une articulation avec l'avantage de rester plus général (on peut décider de diviser le corps n'importe où). L'utilisation d'une notation différente basée sur des repères (cf. Section 5.3.2) permet cette généralité.

suivant l'équation (3.6), ceci pour un robot de type sériel. L'introduction de boucles cinématiques dans la structure du robot augmente le nombre de relations liant les différents corps du robot. Ces relations supplémentaires vont être traitées par une extension de la matrice jacobienne.

### 3.4.2.1 Contraintes dans un robot

Afin d'aborder le problème du calcul de la cinématique de robots à structure quelconque en fonction de consignes fournies par l'utilisateur, nous parlerons dans la suite de ce rapport de "contraintes". Ceci permet d'unifier le concept des relations reliant tous les corps formant un robot, et les relations des corps avec des influences externes. Ainsi un robot possédant des boucles cinématiques et dont l'organe terminal doit respecter des consignes utilisateur comportera deux types de contraintes:

1) les contraintes de fermeture des boucles afin que le robot conserve sa structure qui a été divisée pour sa représentation en arbre,
2) la contrainte de l'organe terminal qui doit constamment "suivre" des consignes (une structure peut posséder plusieurs contraintes de ce type).

Ces deux types de contraintes sont représentées par des arcs qui relient les différents noeuds de l'arbre représentant le robot (cf. Section 5.3.3 pour plus de détails). Elles transforment la structure arborescente en un graphe qui décrit le robot complet avec ses interactions internes et externes.

Les deux types de contraintes précitées (fermeture de boucle et suivi de consigne) vont êtres traités de la même manière en utilisant une matrice jacobienne augmentée. Cette matrice est formée d'autant de sous-matrices jacobiennes qu'il existe de contraintes. Pour un robot contenant *b* boucles cinématiques et *c* consignes (suivre un senseur), nous définissons la matrice jacobienne comme suit:

$$J_{augmenté} = \begin{bmatrix} J_{consigne\ 1} \\ \dots \\ J_{consigne\ c} \\ J_{fermeture\ boucle\ 1} \\ \dots \\ J_{fermeture\ boucle\ b} \end{bmatrix} \tag{3.24}$$

Les contraintes générées par la liaison entre un organe terminal du robot et une consigne de l'utilisateur est traitée comme la cinématique inverse d'un robot sériel dont la matrice jacobienne est calculée suivant la méthode de la Section 3.3. Les contraintes pour les fermetures des boucles cinématiques font l'objet de la section suivante.

### 3.4.2.2 Contraintes de fermeture de boucles

Les contraintes générées par une fermeture de boucle sont satisfaites de la manière suivante: les contributions des déplacements mesurés à la

terminaison de chaque sous-branche (droite et gauche) composant une boucle doivent être égales. Ceci équivaut à dire que les déplacements dus aux variations des articulations de la sous-branche gauche doivent annuler les déplacements dus aux variations des articulations de la sous-branche droite. La Figure 3-7 montre comment sont assignés les indices des corps et articulations pour l'embranchement et la coupure d'une boucle cinématique.

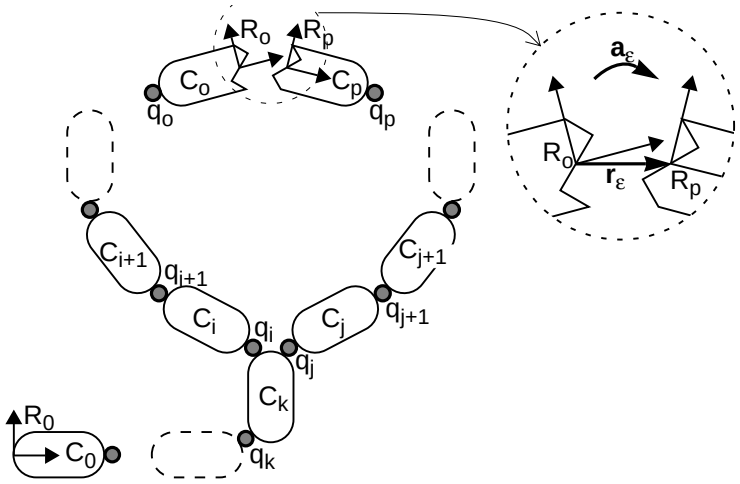



**Figure 3-7:** *Assignation des indices sur une boucle cinématique*

Cette équivalence des déplacements des deux sous-branches s'exprime donc comme suit:

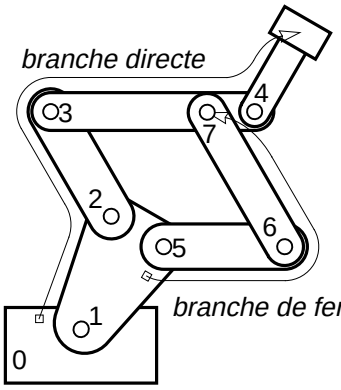
$$\begin{cases} \begin{bmatrix} \mathbf{d}_o \\ \underline{\delta}_o \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{gauche} \end{bmatrix} \cdot \begin{bmatrix} d\mathbf{q}_i \\ \cdots \\ d\mathbf{q}_o \end{bmatrix} \\ \begin{bmatrix} \mathbf{d}_p \\ \underline{\delta}_p \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{droite} \end{bmatrix} \cdot \begin{bmatrix} d\mathbf{q}_j \\ \cdots \\ d\mathbf{q}_p \end{bmatrix} \end{cases} \quad \textbf{avec} \quad \begin{bmatrix} \mathbf{d}_o \\ \underline{\delta}_o \end{bmatrix} = \begin{bmatrix} \mathbf{d}_p \\ \underline{\delta}_p \end{bmatrix} \qquad (3.25)$$

Si l'on compose les matrices et si l'on exprime les déplacements sous forme de vitesses, on obtient alors:

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{j}_i & \cdots & \mathbf{j}_o & -\mathbf{j}_j & \cdots & -\mathbf{j}_p \end{bmatrix} \cdot \begin{bmatrix} \dot{q}_i \\ \cdots \\ \dot{q}_o \\ \dot{q}_j \\ \cdots \\ \dot{q}_p \end{bmatrix} \qquad (3.26)$$

où $\mathbf{j}_k$ est le vecteur colonne de l'influence de l'articulation k (pour k=i, i+1, ... , o, j, j+1, p) suivant le Tableau 3-6.

L'équation (3.26) exprime la contrainte de fermeture d'une boucle cinématique. La matrice jacobienne de cette équation sera répétée pour chaque boucle de la structure afin de composer la matrice jacobienne augmentée finale de la définition (3.24). La Figure 3-8 montre un exemple de matrice jacobienne augmentée pour un robot manipulateur simplifié comportant une boucle. Il est clair que suivant la généralité de la matrice jacobienne augmentée de l'expression (3.24) le nombre de boucles acceptable dans une structure cinématique n'est limité que par le temps de calcul nécessaire à la résolution.




$$J = \begin{bmatrix} d_{x1} & d_{x2} & d_{x3} & d_{x4} & 0 & 0 & 0 \\ d_{y1} & d_{y2} & d_{y3} & d_{y4} & 0 & 0 & 0 \\ d_{\alpha 1} & d_{\alpha 2} & d_{\alpha 3} & d_{\alpha 4} & 0 & 0 & 0 \\ 0 & -d_{x2} & -d_{x3} & 0 & d_{x5} & d_{x6} & d_{x7} \\ 0 & -d_{y2} & -d_{y3} & 0 & d_{y5} & d_{y6} & d_{y7} \\ 0 & -d_{\alpha 2} & -d_{\alpha 3} & 0 & d_{\alpha 5} & d_{\alpha 6} & d_{\alpha 7} \end{bmatrix}$$

contrainte de l'organe terminal (suivit de consigne utilisateur)

contrainte de fermeture de boucle (structure propre du robot)

**Figure 3-8:** *Matrice jacobienne augmentée pour un manipulateur planaire hybride*†

†. Pour des raisons de simplification d'écriture, on ne considère pas dans cet exemple l'ajout nécessaire d'un repère pour que la boucle se ferme bien au niveau de l'articulation numéro 7. Seules les influences des articulations sont prises en compte.

### 3.4.3 Résolution des contraintes

La résolution de toutes les contraintes d'un robot (structure et consigne) va donc s'effectuer grâce à l'équation (3.7) dans laquelle la matrice jacobienne est remplacée par la matrice jacobienne augmentée de la définition (3.24) et le vecteur $\dot{\mathbf{x}}$ va aussi être augmenté pour respecter à la fois les contraintes internes et les multiples consignes:

$$\begin{bmatrix} \dot{q}_1 \\ \dots \\ \dot{q}_n \end{bmatrix} = \left[ J_{\text{augmenté}} \right]^{-1} \cdot \begin{bmatrix} \dot{x}_{1,1} \\ \dots \\ \dot{x}_{c,m} \\ \varepsilon_{1,1} \\ \dots \\ \varepsilon_{b,m} \end{bmatrix} \qquad (3.27)$$

où $\dot{\mathbf{x}}_{k=1\dots c} = \left[ \dot{x}_{k,m} \ \dots \ \dot{x}_{k,m} \right]^T$ sont les *c* vecteurs consignes,

et $\varepsilon_{l=1\dots b} = \left[ \varepsilon_{l,m} \ \dots \ \varepsilon_{l,m} \right]^T$ sont les *b* vecteurs erreurs de fermeture de boucle (cf. Section 3.4.3.2).

L'explication de l'obtention des vecteurs consignes et du pourquoi des vecteurs de fermeture de boucle est donnée dans les deux paragraphes suivant.

### 3.4.3.1 Obtention des vecteurs consignes

Dans le projet «CINEGEN», l'utilisateur définit les consignes de manière interactive par l'intermédiaire de périphériques de saisie que nous appellerons les *senseurs*. Le programme lit donc les valeurs transmises par les senseurs et construit pour chacun d'eux un vecteur à six composantes:

$\mathbf{v}_k=[v_{k,x}\ v_{k,y}\ v_{k,z}\ v_{k,\alpha}\ v_{k,\beta}\ v_{k,\gamma}]^T$,

$\mathbf{v}_k$ représente les déplacements imposés par le senseur k. Le vecteur $\mathbf{v}_k$ est donc composé des translations différentielles $\mathbf{d}_k$ et $\underline{\delta}_k$ des rotations différentielles: $\mathbf{v}_k = [\ \mathbf{d}_k \mid \underline{\delta}_k\ ]^T$, quelque soit le type de senseur (si il possède moins de 6 d.d.l., certaines composantes de $\mathbf{v}_k$ restent nulles).

Afin d'obtenir le mode de contrôle le plus adapté à la tâche à effectuer, l'utilisateur peut travailler dans différents repères (monde, point de vue et outil sont les plus courants).

Il faut donc transposer l'expression du vecteur différentiel du repère où il est mesuré dans le repère où l'on a calculé la matrice jacobienne. Selon la Figure 3-9, on appelle $R_W$ le repère de base, $R_S$ le repère dans lequel le senseur évolue et $R_E$ le repère de l'expression de la matrice jacobienne.

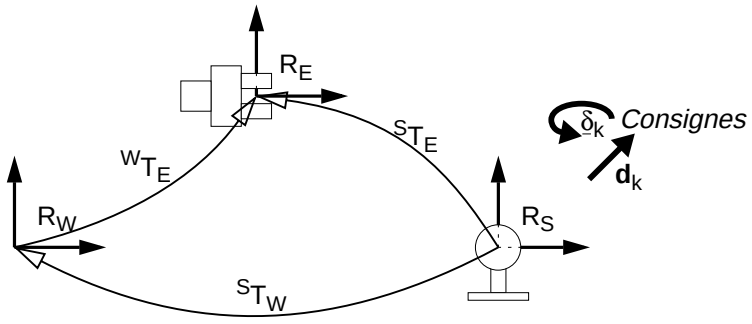



**Figure 3-9:** *Transformation des consignes fournies par un Senseur*

Les consignes fournies par le senseur qui sont relatives à $R_S$, sont exprimées dans le repère $R_E$ suivant la relation [Dombre 88]:

$$\begin{cases} {}^E\underline{\delta}_E = {}^S A_E{}^T \cdot {}^S\underline{\delta}_S \\ {}^E\mathbf{d}_E = {}^S A_E{}^T \cdot ({}^S\underline{\delta}_S \times {}^S r_E + {}^S\mathbf{d}_S) \end{cases} \tag{3.28}$$

où ${}^S A_E$ est la sous-matrice de rotation de la matrice homogène ${}^S T_E$, et ${}^S\mathbf{r}_E$ son vecteur translation.

Dans le cas présent, on obtient ${}^S T_E$ simplement par composition des deux matrices connues ${}^W T_E$ et ${}^S T_W$ : ${}^S T_E = {}^S T_W \cdot {}^W T_E$.

### 3.4.3.2 Traitement de la fermeture des boucles

Etant donné que la solution obtenue par la pseudo-inverse est une approximation, il est possible que la fermeture des boucles ne soit pas strictement respectée. En effet, imaginons un robot contenant une boucle cinématique et devant suivre une consigne dans une direction donnée. S'il n'est pas possible à la structure de se déplacer pour résoudre exactement la contrainte imposée par la consigne, l'algorithme va approximer une solution. L'erreur résultante va se répartir d'une part sur un suivi non parfait de la consigne, mais aussi d'autre part sur une fermeture partielle de la boucle cinématique. Etant donné que les contraintes pour les boucles sont aussi incrémentales (le déplacement d'une sous-branche droite doit être égal au déplacement de la sous-branche gauche d'une boucle), si cette approximation se répète souvent, la boucle va s'ouvrir petit à petit.

Pour résoudre ce problème, nous calculons le vecteur déplacement représentant l'erreur entre les deux sous-branches, et injectons ce vecteur dans la résolution de la matrice jacobienne augmentée. Ainsi lorsqu'une boucle a tendance à s'ouvrir, elle est immédiatement refermée sous l'influence des contraintes supplémentaires de déplacement (appliquées aux sous-branches formant une boucle) qu'occasionnent les vecteurs $\varepsilon_k$ .

Chaque vecteur $\varepsilon_k$ est calculé en fonction des matrices de transformations ${}^0T_o$ et ${}^0T_p$ liées aux repères $R_o$ et $R_p$ (cf. Figure 3-7) représentant les extrémités des sous-branches à joindre. De ces matrices homogènes 4x4 on extrait:

- deux vecteurs position $\mathbf{r}_o=[r_{xo}\ r_{yo}\ r_{zo}]^T$ et $\mathbf{r}_p=[r_{xp}\ r_{yp}\ r_{zp}]^T$ représentant les origines des repères $R_o$ et $R_p$ par rapport au repère de base,
- deux quaternions[6] $Q_o$ et $Q_p$ représentant les orientations des repères $R_o$ et $R_p$ par rapport au repère de base.

Les vecteurs $\mathbf{r}_o$ et $\mathbf{r}_p$ servent à déterminer le vecteur de translation qui sépare les deux extrémités de chaîne qui doivent se joindre:

$$\mathbf{r}_\varepsilon = \mathbf{r}_p - \mathbf{r}_o \tag{3.29}$$

Les quaternions $Q_o$ et $Q_p$ permettent de déterminer la rotation[7] entre les repères $R_o$ et $R_p$ sans ambiguïté. En effet, soustraire par exemple directement les angles d'Euler provenant des matrices ${}^0T_o$ et ${}^0T_p$ amène facilement à des problèmes d'erreur de $\pm\pi$ ou $\pm\pi/2$ puisque la représentation par des angles d'Euler n'est pas unique. On obtient donc un quaternion représentant l'erreur de rotation entre les repères terminaux de sous-chaînes formant une boucle:

$$Q_\varepsilon = Q_o / Q_p \tag{3.30}$$

Ce quaternion $Q_\varepsilon$ est finalement converti en trois angles d'Euler $\alpha_\varepsilon$, $\beta_\varepsilon$ et

6. Un quaternion, parfois appelé "paramètres d'Euler", est un quadruplet représentant une orientation de l'espace [Spring 86] [Funda 88]. L'avantage des quaternions par rapport aux autres représentations de rotation, est qu'ils ne présentent aucune singularité.
7. Le quaternion $q_r$ représentant la rotation qui existe entre deux quaternions $q_1$ et $q_2$ est défini par $q_r=q_1/q_2$.

$\gamma_\varepsilon$ afin de former un vecteur de rotation différentiel $\mathbf{a}_\varepsilon$: $\mathbf{a}_\varepsilon=[\alpha_\varepsilon\ \beta_\varepsilon\ \gamma_\varepsilon]^T$.

On obtient donc le vecteur erreur désiré qui va produire les variations nécessaires aux articulations pour que chaque chaîne cinématique se referme:

$$\varepsilon_{l=1\ldots b} = \left[\mathbf{r}_{l,\varepsilon} \quad \mathbf{a}_{k,\varepsilon}\right] \tag{3.31}$$

### 3.4.3.3 Cinématique inverse et directe

La méthode employée dans «CINEGEN» permet de résoudre aussi bien la cinématique inverse que directe de manipulateurs à structure quelconque.

Pour illustrer ce concept, reprenons le manipulateur hybride de la Figure 3-8 et imaginons qu'il puisse être contrôlé par un senseur fournissant des consignes à l'organe terminal.

En utilisant la matrice jacobienne augmentée complète définie dans cette même figure, on contrôle le manipulateur suivant son modèle inverse: les différentes articulations vont évoluer afin de satisfaire d'une part la consigne appliquée sur l'organe terminal et d'autre part la contrainte de fermeture de la boucle cinématique.

Pour contrôler ce même manipulateur suivant le modèle direct, il suffit de supprimer la consigne sur l'organe terminal ainsi que d'éliminer les trois premières lignes (6 pour un manipulateur spatial) de la matrice jacobienne augmentée. On contrôle alors la valeur d'une articulation quelconque à l'aide d'une interface adéquate, ce qui modifie les matrices de transformations de chaque lien et oriente le manipulateur en conséquence. Simultanément, l'inversion de la matrice jacobienne augmentée, alors réduite, continue de satisfaire la contrainte de fermeture de la boucle cinématique: le manipulateur évolue suivant son modèle direct.

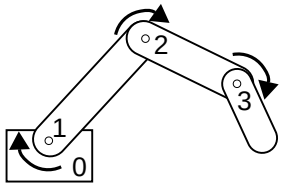
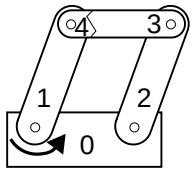
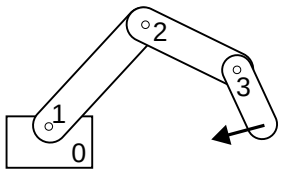
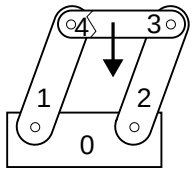
Le Tableau 3-7 illustre la résolution des modèles cinématiques, direct[8] et inverse, pour des manipulateurs sériels et parallèles. Comme l'exemple du paragraphe précédent le montre, ce mode de résolution s'étend évidemment à tout manipulateur hybride (série+boucles) par composition des différents modèles.

Le programme *VirtualRobot* qui est la concrétisation du projet «CINEGEN» permet la résolution complète de la cinématique (directe plus inverse) de robots manipulateurs à structure quelconque grâce à:

- une construction du jacobien flexible, reconfigurable dynamiquement en fonction des modifications des contraintes,

8. D'un point de vue rigoureux, la résolution du modèle direct n'est pas cinématique, mais géométrique (position de l'organe terminal en fonction des valeurs angulaires des articulations). Dans le cas d'une structure sérielle on utilise uniquement la composition des matrices. Toutefois, bien que l'on puisse en effet fournir des valeurs articulaires quelconques, on peut considérer que l'on fait évoluer le robot plutôt par incréments à partir d'une configuration donnée. Pour des robots avec des boucles, les consignes articulaires sont prises en compte, mais la fermeture des boucles est incrémentale. Le mode utilisé dans le cas général se place donc entre un modèle géométrique pur et un modèle cinématique.

**Tableau 3-7:** *Modes de résolution des contraintes des robots dans «CINEGEN»*

| *Consigne articulaire* / *Consigne cartésienne* | **Robots sériels** | **Robots Parallèles** |
|---|---|---|
| **Modèle Direct** |  $^0T_3 = {}^0T_1 \cdot {}^1T_2 \cdot {}^2T_3$ |  $\dot{\mathbf{q}} = J_{boucle} \cdot \underline{\varepsilon}$ |
| **Modèle Inverse** |  $\dot{\mathbf{q}} = J_{consigne} \cdot \dot{\mathbf{x}}$ |  $\dot{\mathbf{q}} = \begin{bmatrix} J_{consigne} \\ J_{boucle} \end{bmatrix} \cdot \begin{bmatrix} \dot{\mathbf{x}} \\ \underline{\varepsilon} \end{bmatrix}$ |

- un calcul en temps réel des transformations de coordonnées des robots en fonction des paramètres des liens,
- une interface avec des périphériques de saisie qui permettent le contrôle de six degrés de liberté afin de fournir des consignes aux organes terminaux,
- une interface avec des panneaux de contrôle permettant de modifier individuellement les paramètres de chaque articulation.

La réalisation des ces différentes particularités sera traitée en détail au Chapitre 5, “Implémentation de l’outil” dans la Section 5.3, et la Section 5.5, ainsi que dans le Chapitre 6, “Fonctionnalités et étude de cas” dans la Section 6.1.

## 3.5 Résumé

Ce chapitre concernant les fondements théoriques de la réalisation du projet «CINEGEN» commence par une description des outils mathématiques existant dans le domaine de la robotique (Section 3.1 à Section 3.3).

La Section 3.4 introduit le concept de cinématique généralisée mise au point pour le projet «CINEGEN». Grâce à ce concept basé sur l’utilisation d’une matrice jacobienne augmentée reconfigurable de manière dynamique, il est possible de contrôler les manipulateurs virtuels à structure quelconque suivant leur cinématique directe ou inverse au sein du même programme.

Le chapitre suivant détaille le format de fichier servant à la description des robots à structure quelconque suivant la méthode présentée dans le présent chapitre.

CHAPITRE

# 4

# FICHIER DE DESCRIPTION MAD

La description des robots à l'aide d'un fichier texte est le concept retenu dans le projet «CINEGEN». Il permet d'obtenir un programme générique qui peut simuler, sans recompilation, tout type de manipulateur.

Ce fichier doit répondre à certaines spécifications provenant de contraintes humaines et informatiques. Il est donc important de définir un format de fichier simple et souple à la fois. Ce chapitre décrit le format de fichier mis au point dans ce travail et traite un exemple concret pour l'illustrer.

## 4.1 Description des robots par un fichier MAD

Nous appellerons un fichier de description des robots, un fichier "MAD" pour **MA**nipulator **D**escription. Tout fichier qui sert à la description de robots dans le projet «CINEGEN» sera appelé "fichier de type MAD". De plus nous désignerons aussi par ce même nom le "langage MAD" sous-jacent à ce fichier de description: la syntaxe et la grammaire d'un fichier MAD.

### 4.1.1 Spécifications du fichier MAD

Le fichier de description des robots MAD est l'entrée principale du programme *VirtualRobot*. C'est à partir de cette description que le programme va construire un environnement de simulation contenant les robots que l'utilisateur désire simuler et manipuler. De ce fait le fichier MAD permet le stockage durable des descriptions des robots: il assure la persistance des données.

Le fichier MAD est la base de données commune à l'utilisateur et au programme. Pour cette raison il doit répondre à trois critères essentiels.

1) Simplicité, car c'est un homme qui le crée et le modifie.
2) Efficacité, puisque c'est la source de données pour le programme.
3) Flexibilité, afin d'être ouvert à des modifications ou extensions futures (comme liaison avec d'autres environnements de visualisation).

Le fichier MAD doit donc avant tout être facilement compréhensible par un créateur de structures robotiques. Ceci implique un fichier de description simple, afin d'être rapidement maîtrisable par une personne travaillant dans le domaine de la robotique, mais pas forcément experte dans la cinématique des robots. La clarté de la syntaxe va donc être déterminante pour l'utilisateur. La mise en page peut aussi influencer la lisibilité d'un tel fichier, il faut donc qu'elle soit suffisamment souple afin de satisfaire tout créateur de fichier MAD. On peut évidemment imaginer une interface graphique pour la création et l'édition du fichier MAD avec laquelle on pourrait éditer les paramètres dans différents champs de saisie. Un outil de ce genre sort toutefois du cadre du projet[1]. D'autre part, l'expérience montre qu'avec un peu d'habitude, il est souvent plus rapide de modifier un simple fichier texte que de passer par des multiples fenêtres pour régler chaque paramètre.

Même s'il est possible de construire des analyseurs de langage naturel, la tâche est laborieuse et les risques de fausses interprétations sont grands. Il est donc plus judicieux de mettre au point une syntaxe de fichier qui soit par la suite facilement analysable. Le point central est de définir une syntaxe et des règles de grammaire qui ne puissent pas amener à des ambiguïtés.

La structure d'un fichier MAD ressemble à celle d'un programme informatique procédural. Chaque définition est délimitée par un "bloc", qui peut contenir d'autres blocs. Un bloc de définition commence par un mot-clef du langage qui annonce le type de données qu'il contient. On obtient ainsi une composition de juxtapositions et imbrications de différents blocs qui représentent les différentes parties des robots. Cette description peut être assimilée à une représentation orientée-objet des robots[2].

Une qualité essentielle d'un fichier de définition de données est son aptitude à évoluer avec les nouvelles exigences d'un projet. Ceci est aisé étant donné la structure en blocs élémentaires du fichier MAD. Chaque élément de base est suffisamment petit pour qu'il n'ait pas besoin d'être modifié. Quand à l'ajout de nouvelles possibilités, elle se fait par l'ajout de nouveaux mots-clefs qui délimiteront de nouveaux blocs de données. Dans le projet «CINEGEN», l'extension de langage de description de robot MAD est aisée grâce à l'utilisation d'un générateur d'analyseur comme le décrit la Section 5.4.3 et la Section 5.5.1.1.

### 4.1.2 Format du fichier MAD

Un fichier de description de robot MAD peut contenir la définition de plusieurs manipulateurs. Dans ce fichier sont aussi déclarés les senseurs (périphériques de saisie) utilisés pour contrôler les robots.

Le fichier MAD fait également référence à tous les fichiers contenant les modèles géométriques utilisés pour représenter les différentes parties des robots. Ces descriptions des objets graphiques se trouvent dans des fichiers différents car les objets peuvent être créés par divers programmes de dessin 3D.

1. Une telle interface serait facile à développer et ne présente pas de difficultés techniques.
2. Contrairement à plusieurs types de descriptions de robots à caractère "séquentiel".

Le langage MAD doit permettre d'ajouter des commentaires dans le fichier afin de rendre plus compréhensible la description de robot. Ces commentaires suivent la même syntaxe que les commentaires d'un programme C++.

Cette section décrit les différents éléments d'un fichier MAD et leur utilisation. Une description complète de sa grammaire se trouve en Annexe A, "Grammaire MAD".

### 4.1.2.1 En-tête du fichier

Afin de conserver une pérennité des fichiers MAD, il est utile de spécifier tout au début du fichier une version du langage MAD utilisé. Ceci permettra aux futures versions de *VirtualRobot* qui utiliseraient des extensions spécifiques, de proposer un analyseur de fichier adapté afin de lire les anciennes versions de fichier MAD. Cette information est aussi utile pour l'utilisateur qui peut immédiatement voir quelle version du langage est utilisée.

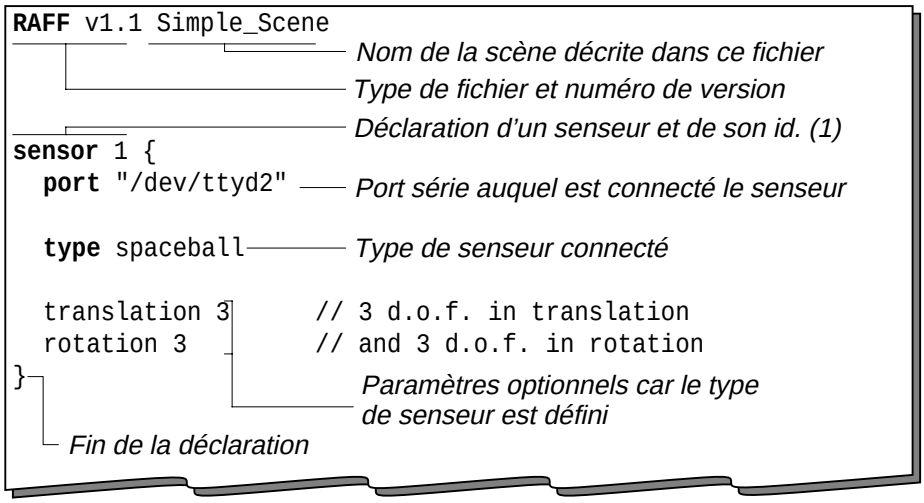



**Figure 4-1:** *Déclarations du début d'un fichier MAD*

### 4.1.2.2 Senseurs

L'interaction entre l'utilisateur et les robots simulés est une composante essentielle du projet «CINEGEN». Grâce à des périphériques de saisie 3D divers regroupés sous le nom de senseurs l'opérateur définit des consignes et manipule les robots. Il est donc nécessaire d'expliciter de quelle manière les senseurs vont contrôler la structure. Ceci est réalisé en ajoutant des contraintes entre un repère attaché au robot et certains senseurs comme l'expliquent la Section 3.4.2.1 et la Section 5.3.2.5.

Il faut déclarer les senseurs dès le début du fichier MAD afin de pouvoir les référencer par la suite lorsque l'on décrit les contraintes externes (suivi de senseur) des robots. La Figure 4-1 montre une déclaration typique de senseur au début d'un fichier de description.

Chaque senseur est identifié par un numéro qui doit être unique. Ensuite le fichier spécifie à quel port est branché le périphérique de saisie physique. Dans l'état actuel du programme, seuls des ports série RS-232 sont pris en

compte. L'extension à des senseurs pouvant envoyer des informations par d'autres canaux (le réseau par exemple) pourrait être utile, et ne pose pas de problèmes quand à la définition du fichier MAD. Toutefois, il faudrait alors écrire les routines adéquates pour lire ces informations.

On peut ensuite définir le type de senseur. Dans l'exemple fourni, le programme crée un objet senseur spécialisé ("spaceball") avec des caractéristiques précises. Dans un cas plus général, on peut définir le type d'informations que le senseur renvoie: nombre de degrés de liberté en translation et en rotation.

#### 4.1.2.3 Robots

Après avoir déclaré un ou plusieurs senseurs, le fichier MAD décrit les différents robots d'une scène à simuler. La Figure 4-2 montre la déclaration d'un nouveau robot à l'intérieur d'une scène décrite par un fichier MAD.

Nous continuerons à employer le terme "repère" pour désigner le concept mathématique, et utiliserons "référentiel" pour désigner l'objet repère utilisé dans le programme.

Chaque robot est identifié par un nom et contient une base, des liens, des *référentiels* et des contraintes. La description au niveau informatique des objets formant un robot est donnée dans la Section 5.3.2. Les paramètres des différents éléments d'un robot sont discutés dans cette section.

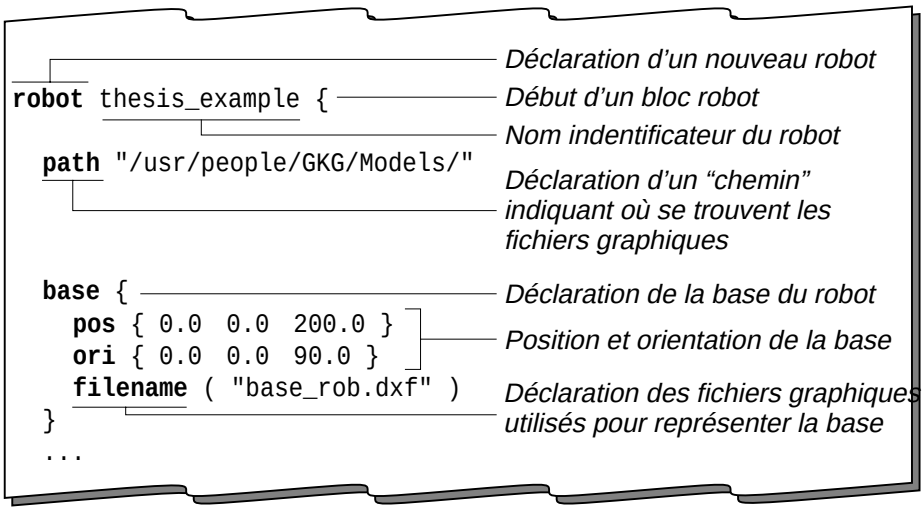



**Figure 4-2:** *Déclaration d'un robot dans un fichier MAD*

##### *Base*

Tout robot manipulateur doit nécessairement posséder une base dans le formalisme de *VirtualRobot*. Cette base n'est pas obligatoirement représentée par un objet graphique, mais elle est la source d'où émanent tous les autres liens du robot. La base est positionnée dans l'espace par trois paramètres de position et trois angles d'Euler comme le montre la Figure 4-2. Etant donné que chaque robot possède une base unique, il n'est pas nécessaire du lui spécifier un identificateur: par défaut toute base est référencée par le numéro 0 (zéro).

##### *Liens*

Les éléments essentiels d'un robot sont évidemment les liens qui le composent. Un lien est donc la combinaison d'une articulation et d'un corps

qui relie cette articulation au composant suivant. Les liens sont numérotés dans le fichier MAD afin des les identifier individuellement.

La géométrie du lien et sa posture par rapport au précédent est définie par les six paramètres de la notation Kleinfinger-Khalil. La plupart du temps on peut omettre les deux paramètres d'embranchement pour alléger l'écriture (cf. Annexe B, "Exemples de fichiers MAD de simulations réalisées" pour des fichiers avec les paramètres supplémentaires γ=`gamma` et ε=`epsil`).

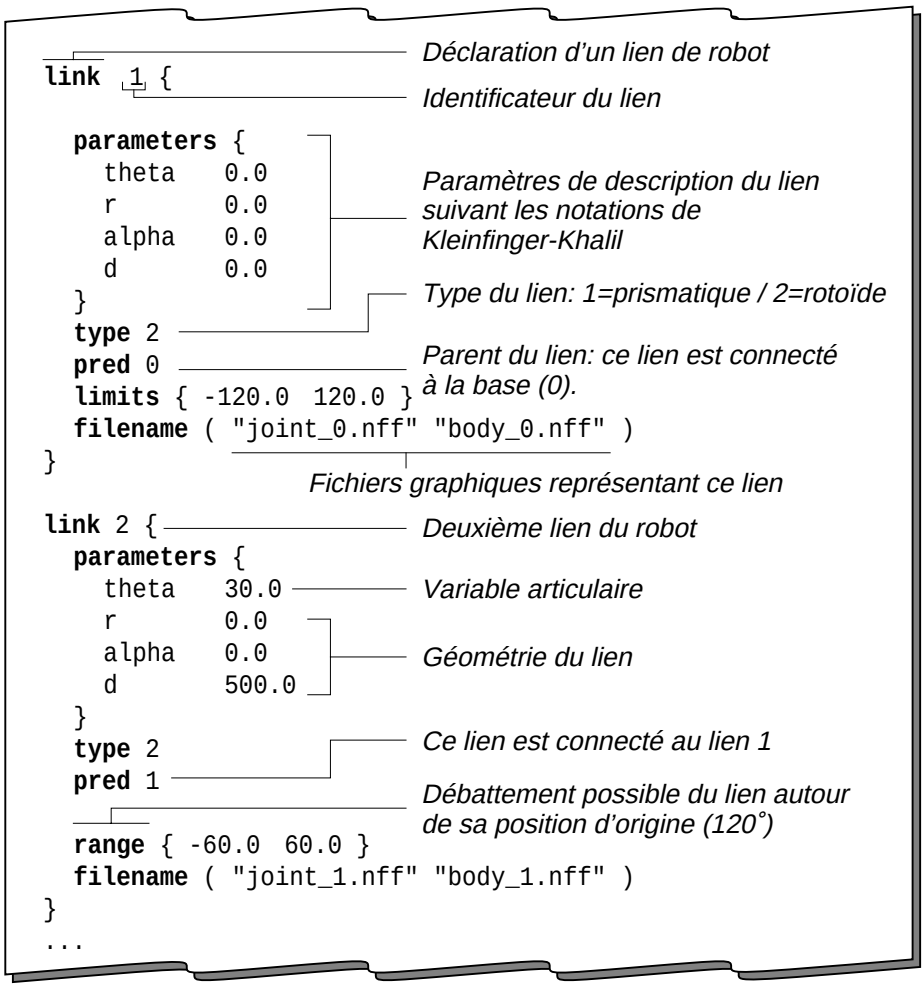



**Figure 4-3:** *Déclaration de liens dans un fichier MAD*

Il faut noter que les distances sont exprimées dans une unité de longueur arbitraire, et que les angles sont codés en degrés (beaucoup plus naturel pour l'utilisateur). Tous les paramètres d'un fichier MAD, quelles qu'en soient leurs utilisations (`parameters`, `range`, `limits`, `pos`, `ori`) suivent cette règle.

Le type d'articulation du lien est défini par un paramètre numérique. Dans l'état actuel de *VirtualRobot*, seuls des liens avec des articulations prismatiques (1) et rotoïdes (2) sont autorisés. Lorsque le lien est de type prismatique, le paramètre `r` devient la variable articulaire et sa valeur initiale est celle donnée dans le fichier. Par contre `theta` reste un paramètre dimensionnel fixe dans ce cas. Pour une articulation rotoïde la situation est

inversée: le paramètre articulaire devient `theta` alors que le paramètre `r` restera alors une constante de la géométrie pendant la simulation.

Le fichier MAD permet de définir des butées aux articulations afin de limiter la course des liens. Ces butées virtuelles peuvent être soit définies de manière absolue par rapport à l'axe de référence du lien (utilisation du mot-clef `limits`) soit de manière relative par rapport à la position initiale de l'articulation (utilisation du mot-clef `range`). L'utilisation de l'une ou l'autre des méthode est préférable suivant les cas et cette double possibilité offre une flexibilité maximale.

La hiérarchie arborescente du robot est décrite à l'aide du mot-clef `pred` qui désigne à quel lien précédent du robot l'articulation courante est attachée. Le fait d'ordonner les liens du robots suivant un arbre, comme expliqué dans la Section 3.4.1, permet donc d'utiliser un identificateur unique pour décrire entièrement la structure.

#### 4.1.2.4 Référentiels

Un type d'objet supplémentaire a été défini dans le projet «CINEGEN» pour aider à la description des structures de robot. Les possibilités qu'offre l'introduction de référentiels dans la structure sont présentées dans la Section 5.3.2.4. Nous retiendrons seulement ici que l'utilisation de référentiels permet une généralité totale dans la description des chaînes cinématiques puisque l'on n'est plus limité par les notations utilisées couramment en robotique pour décrire des liens. De plus l'utilisation de référentiels enlève toute ambiguïté pour décrire les structures bouclées complexes.

Un référentiel est défini dans un fichier MAD par trois coordonnées de position et trois angles d'Euler comme dans l'exemple de la Figure 4-4. Ces six paramètres placent le référentiel par rapport à un élément précédent, qui peut être soit un autre référentiel, soit un lien.

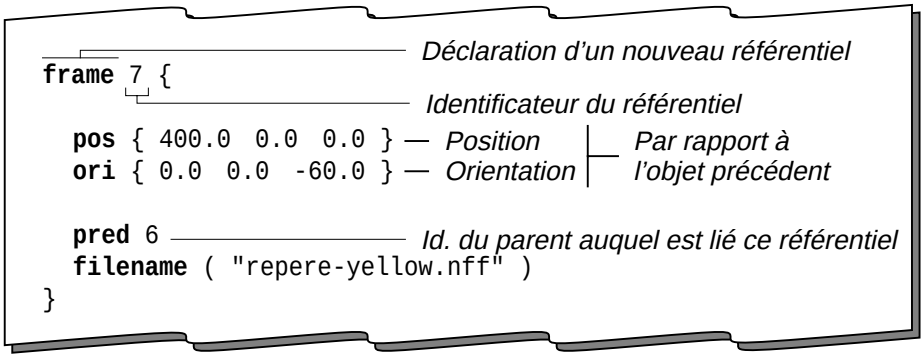



**Figure 4-4:** *Déclaration d'un référentiel dans un fichier MAD*

#### 4.1.2.5 Fichiers graphiques

Chaque lien du robot peut être représenté par un ou plusieurs objets graphiques lors de la visualisation de la scène. Comme on peut le remarquer dans tous les extraits de fichier MAD des exemples précédents, les autres éléments d'un robot peuvent aussi posséder une représentation graphique.

Indépendamment de leur nature, une base, un lien ou un référentiel peuvent être visualisés par le fichier graphique qui leur est associé.

Cette visualisation d'objets, qui n'ont pas forcément de réalité physique (comme un référentiel), est parfois très utile pour mieux appréhender une structure complexe ou le comportement des contraintes de fermeture. Il est aussi possible d'ajouter des éléments graphiques à la scène sans forcément qu'ils aient une relation avec un lien donné.

Le mot-clef `path` (cf. Figure 4-2) – au début de la déclaration d'un robot – permet de localiser le répertoire où sont stockés les fichiers graphiques. Ceci évite de chaque fois répéter la localisation complète du fichier graphique. Il est possible d'attacher de multiples objets graphiques à un même élément du robot. Pour différentier l'aspect "liste" à taille non définie, on utilise dans la syntaxe des parenthèses plutôt que les accolades normalement utilisées pour délimiter les blocs à structure fixe[3].

Afin de positionner correctement les objets graphiques par rapport aux éléments du robot, il a été choisi par convention que leur orientation et position soient alignées sur les repères d'origine de ces éléments. La Figure 4-5 montre schématiquement cette méthode. Toutefois s'il est impossible de positionner l'objet graphique de cette manière (récupération d'un format externe par exemple), l'utilisation d'un référentiel supplémentaire permet alors de centrer correctement l'élément du robot et l'objet graphique.

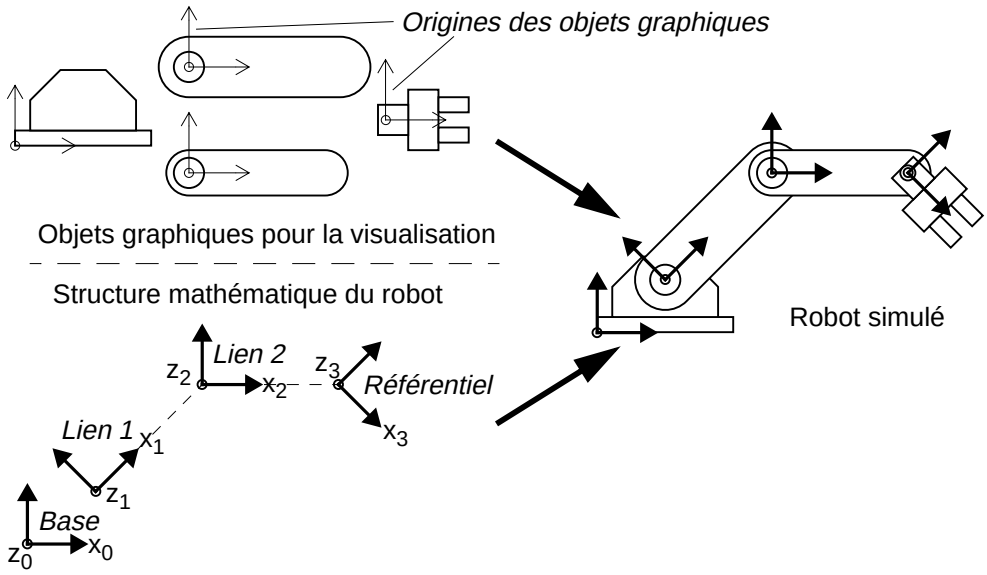



**Figure 4-5:** *Positionnement des objets graphiques*

### 4.1.2.6 Définition des contraintes

La Section 3.4.2.1 explique les deux types de contraintes que l'on trouve dans un robot défini dans «CINEGEN». Ces deux types de contraintes se déclarent

3. Cette règle est violée pour la déclaration d'un `robot` puisque l'ensemble de ses composant – dont le nombre n'est pas fixé à l'avance – est délimité par des accolades (robot { ... })! Cette contradiction est historique et perdure pour des raisons de compatibilité...

de la même manière, comme on le remarque dans l'exemple de la Figure 4-6, à l'aide du mot-clef `constraint`. Les deux éléments entre lesquels une contrainte est créée sont déclarés dans le bloc de la définition de contrainte. Le Tableau 4-1 décrit les cinq possibilités différentes de création des contraintes entre les divers types d'éléments existant dans un robot (chaque élément devient un noeud du graphe). L'ordre des deux noeuds entre lesquels la contrainte est créée est indifférent.

**Tableau 4-1:** *Types de contraintes dans le graphe du robot*

| noeud A | noeud B | Type de contrainte généré | Action dans le graphe |
|---|---|---|---|
| lien | lien | Fermeture de boucle | Création d'un arc (de type contrainte de fermeture) entre les deux noeuds A et B |
| référentiel | référentiel | | |
| lien | référentiel | | |
| lien | senseur | Suivi de senseur | Création d'un "traqueur"† T et d'un arc (type suivi) entre A et T |
| référentiel | senseur | | |

†. Un *traqueur* est un objet qui effectue la liaison entre un senseur virtuel et le senseur physique. Voir la Section 5.3.2.5 pour une définition complète, ainsi que la Figure 5-11.

Pour les contraintes entre des liens et référentiels, il suffit de spécifier les identificateurs des éléments désirés. Pour une contrainte avec un senseur, il est possible de spécifier une liste de senseurs potentiels. Ainsi il est possible de contrôler l'élément du robot avec plusieurs senseurs différents. Ceci est effectué en modifiant la contrainte entre l'élément concerné et chaque senseur successivement. Le cas spécifique du senseur n° 0 correspond à une contrainte nulle: l'élément contraint n'est plus obligé de suivre aucun senseur, on n'effectue donc plus de cinématique inverse (pour cet élément).

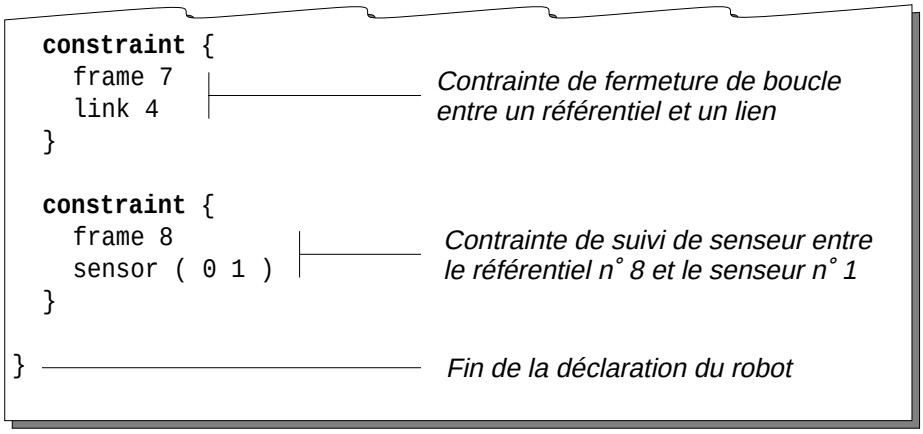



**Figure 4-6:** *Déclaration des contraintes dans un fichier MAD*

## 4.2 Exemple de création d'un fichier MAD

Cette section propose un exemple concret de la création d'un fichier MAD en vue de la simulation et de la manipulation d'un robot. La première étape consiste à former le graphe correspondant à la structure désirée. L'étape

suivante comprend l'assignation de repères sur chaque lien suivant la notation de Kleinfinger-Khalil ainsi que la détermination des paramètres géométriques. La dernière étape est l'écriture proprement dite du fichier MAD

### 4.2.1 Analyse de la structure

Afin de proposer un exemple simple mais suffisamment complet pour comprendre toutes les fonctionnalités de la description des robots avec le fichier MAD, nous reprendrons l'exemple du robot planaire hybride de la Figure 3-8. Cette structure articulée possède 7 articulations rotoïdes et une boucle cinématique, elle conserve donc 4 degrés de liberté ce qui en fait une structure redondante.

La boucle de cette structure doit avant tout être "ouverte" pour permettre la description sous forme d'arbre comme l'indique la Section 3.4.1. Cette ouverture de la boucle est schématisée dans la Figure 4-7. Afin de conserver un bon équilibre, la structure est éclatée au niveau de l'articulation n°7. Le corps virtuel (puisqu'il ne correspond pas à un élément physique) lié à cette articulation devra correspondre en position et orientation avec un référentiel supplémentaire (n°8 sur la figure) attaché au corps n°3. Nous obtenons ainsi deux chaînes cinématiques:

- la chaîne principale composée des liens 0, 1, 2, 3, et 4,
- la chaîne secondaire composée des liens 5, 6, et 7; l'articulation 5 est référencée par rapport au corps 1.

Un second référentiel (n° 9 sur la figure) attaché sur l'organe terminal est utilisé afin d'appliquer les consignes utilisateur à cet endroit précis (et non au niveau de l'articulation n°4).

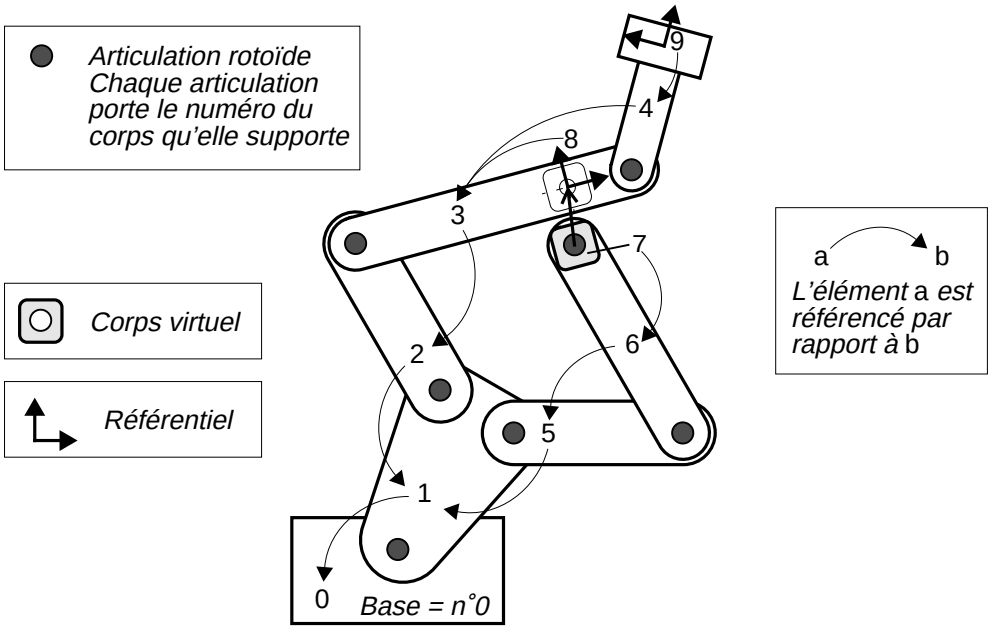



**Figure 4-7:** *Robot hybride planaire "éclaté"*

### 4.2.2 Description géométrique

Une fois la structure analysée, il faut écrire ses caractéristiques géométriques à l'aide de la notation Kleinfinger-Khalil (cf. Section 3.2.2.3) et des référentiels supplémentaires introduits ($R_8$ et $R_9$).

Chaque articulation i supporte un corps $C_i$. L'ensemble articulation-corps constitue un lien de robot noté $L_i$. La base est désignée par $B_0$. Les axes $z_i$ (définis par le vecteur $\mathbf{z}_i$) de chaque *repère* décrivant un lien $L_i$ sont concourants avec l'axe de l'articulation correspondante i. Dans le cas de la structure étudiée, tous les axes $z_i$ sont parallèles (perpendiculaire au plan de la feuille). Quand aux axes $x_i$, ils sont définis par la perpendiculaire commune aux axes $z_i$ et $z_{i+1}$. Les distances entre les axes $z_{i-1}$ et $z_i$ le long de $x_i$ sont notées $D_i$. Les variables articulaires associées aux articulations i sont désignées par $\theta_i$.

Cette structure comporte un embranchement au niveau du corps 1. Cela implique de définir un axe supplémentaire $x_1$' dont l'orientation est donnée par le paramètre $\gamma_5$ par rapport à $x_1$. La Figure 4-8 représente l'assignation de tous les référentiels ainsi que la définition des paramètres principaux.

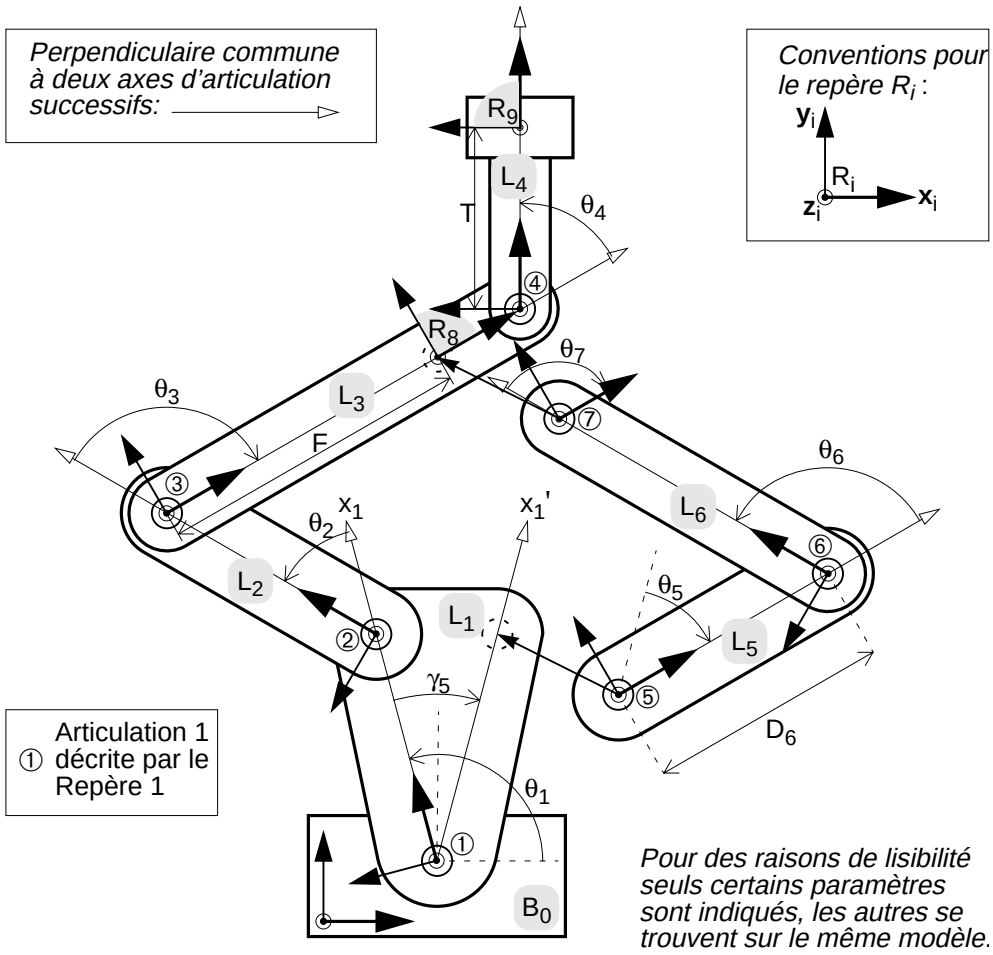



**Figure 4-8:** *Référentiels et paramètres du robot hybride planaire*

Deux distances supplémentaires sont nécessaires pour positionner les référentiels par rapport aux corps qui les supportent: F pour positionner le référentiel $R_8$ avec lequel l'articulation 7 devra coïncider; T pour positionner "l'outil" du manipulateur par rapport à la dernière articulation 4. L'ensemble de ces caractéristiques permet de définir le Tableau 4-2 regroupant les paramètres Kleinfinger-Khalil de cette structure (le référentiel de la base et celui du premier lien sont considérés comme ayant la même origine pour simplifier les paramètres).

**Tableau 4-2:** *Paramètres Kleinfinger-Khalil pour le robot hybride planaire*

| art. i† | pred.‡ | $\theta_j$ | $r_i$ | $\alpha_i$ | $d_i$ | $\gamma_i$ | $\varepsilon_i$ |
|---|---|---|---|---|---|---|---|
| 1 | 0 | $\theta_1$ | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | $\theta_2$ | 0 | 0 | $D_2$ | 0 | 0 |
| 3 | 2 | $\theta_3$ | 0 | 0 | $D_3$ | 0 | 0 |
| 4 | 3 | $\theta_4$ | 0 | 0 | $D_4$ | 0 | 0 |
| 5 | 1 | $\theta_5$ | 0 | 0 | $D_5$ | $\gamma_5$ | 0 |
| 6 | 5 | $\theta_6$ | 0 | 0 | $D_6$ | 0 | 0 |
| 7 | 6 | $\theta_7$ | 0 | 0 | $D_7$ | 0 | 0 |

†. numéro de l'articulation courante
‡. numéro du lien auquel l'articulation considérée est fixée

Le Tableau 4-3 fournit les caractéristiques des référentiels utilisés.

**Tableau 4-3:** *Paramètres des référentiels pour le robot hybride planaire*

| réf. | pred. | position | | | orientation† | | |
|---|---|---|---|---|---|---|---|
| | | x | z | y | $\alpha$ | $\beta$ | $\gamma$ |
| 8 | 3 | F | 0 | 0 | 0 | 0 | 0 |
| 9 | 4 | T | 0 | 0 | 0 | 0 | 0 |

†. Angles d'Euler.

### 4.2.3 Ecriture du fichier MAD

La dernière étape consiste à écrire le fichier MAD correspondant à la structure décrite. Pour cela il est nécessaire de fournir des valeurs numériques aux différents paramètres. Les paramètres $D_i$ doivent être définis suivant la taille de la structure que l'on désire. Il faut de plus assigner des valeurs aux paramètres $\theta_i$ afin de donner une posture initiale à la structure (celle de la Figure 4-8 par exemple). Le Tableau 4-4 montre les valeurs choisies pour la structure hybride.

Les valeurs fournies sont arrondies: la construction géométrique d'une structure correspondant à ces valeurs ne respectera pas exactement la condition de fermeture de la boucle. Ceci n'est pas grave, car l'algorithme de résolution de contraintes va immédiatement refermer la boucle en ajustant les valeurs des articulations.

Ce Tableau 4-4 définit aussi les limites des butées de chaque articulation. Dans notre cas toutes ces limites sont définies par rapport à l'angle initial de

l'articulation, c'est-à-dire de manière relative. Ces limites sont choisies pour cet exemple particulier de manière empirique simplement afin de limiter les mouvements inconsidérés de la structure.

**Tableau 4-4:** *Paramètres dimensionnels du robot hybride planaire*

| lien | $\theta^{\dagger}$ | d | min | max |
|---|---|---|---|---|
| 1 | 0.0 | 0 | -160.0 | 160.0 |
| 2 | 51.0 | 500.0 | -120.0 | 120.0 |
| 3 | -120.0 | 400.0 | -50.0 | 110.0 |
| 4 | 60.0 | 600.0 | -120.0 | 120.0 |
| 5 | -51.0 | 500.0 | -120.0 | 120.0 |
| 6 | 120.0 | 400.0 | -110.0 | 50.0 |
| 7 | -60.0 | 492.0 | -150.0 | 150.0 |
| **autres paramètres:** | | F=492.0 | T=200.0 | $\gamma_5$=-18° |

†. en degrés

La Figure 4-9 présente le fichier MAD complet décrivant la structure hybride présentée. On peut remarquer que 120 lignes de texte très simple suffisent pour obtenir une simulation complète dans un environnement virtuel de ce robot.

Il faut aussi définir au minimum un senseur afin de pouvoir manipuler la structure créée suivant sa cinématique inverse. Ce senseur est déclaré au début du fichier MAD. Nous utilisons pour cet exemple un senseur de type Magellan de Logitech (se comportant exactement comme une SpaceBall, d'où l'identificateur "spaceball"). Quand à la cinématique directe, elle s'effectue simplement en ouvrant un panneau de contrôle pour le lien désiré (cf. Section 6.1.2.1).

La fin du fichier est particulièrement importante puisqu'elle décrit les deux contraintes que l'on rencontre dans ce robot:

- la contrainte de fermeture de boucle; le lien 7 et le référentiel 8 doivent se superposer,
- la contrainte de suivi du senseur principal; le référentiel doit soit suivre le senseur 0 (senseur nul: la structure ne bouge pas) soit suivre le senseur 1 défini comme une SpaceBall.

Finalement, on remarque aussi dans ce fichier la déclaration des objets graphiques utilisés pour représenter la structure. Certains modèles graphiques sont utilisés pour représenter des éléments particuliers (base, lien 1, référentiel 9 représentant le point de contrôle de l'outil). Toutefois, la plupart des liens utilisent la fonctionnalité de représentation automatique des éléments (mot-clef `auto`, cf Section 6.3.5), ce qui permet de mettre au point très rapidement une simulation.

La Figure 4-10 représente un exemple de copie d'écran de la structure simulée avec *VirtualRobot* à partir du fichier de description présenté.

```
RAFF thesis_example

sensor 0 { }
sensor 1 {
  port "/dev/ttyd2"
  type spaceball
}

robot hybride {
  path "/usr/projects/Models/"

  base { filename ( "base.nff" ) }

  link 1 {
    parameters {
      theta 0.0
      r     0.0
      alpha 0.0
      d     0.0
    }
    type 2
    pred 0
    range { -160.0 160.0 }
    filename ( "link_1.dxf" )
  }

  link 2 {
    parameters {
      theta 51.0
      r     0.0
      alpha 0.0
      d     500.0
    }
    type 2
    pred 1
    range { -120.0 120.0 }
    filename ( auto )
  }

  link 3 {
    parameters {
      theta -120.0
      r     0.0
      alpha 0.0
      d     400.0
    }
    type 2
    pred 2
    range { -50.0 110.0 }
    filename ( auto )
  }

  link 4 {
    parameters {
      theta 60.0
      r     0.0
      alpha 0.0
      d     600.0
    }
    type 2
    pred 3
    range { -120.0 120.0 }
    filename ( auto )
  }

  link 5 {
    parameters {
      theta -51.0
      r     0.0
      alpha 0.0
      d     500.0
      gamma -18.0
      epsil 0.0
    }
    type 2
    pred 1
    range { -120.0 120.0 }
    filename ( auto )
  }
  link 6 {
    parameters {
      theta 120.0
      r     0.0
      alpha 0.0
      d     400.0
    }
    type 2
    pred 5
    range { -110.0 50.0 }
    filename ( auto )
  }
  link 7 {
    parameters {
      theta -60.0
      r     0.0
      alpha 0.0
      d     492.0
    }
    type 2
    pred 6
    range { -150.0 150.0 }
    filename ( auto )
  }

  frame 8 {
    pos { 492.0 0.0 0.0 }
    ori { 0.0 0.0 60.0 }
    pred 3
    filename ( )
  }

  frame 9 {
    pos { 200.0 0.0 0.0 }
    ori { 0.0 0.0 0.0 }
    pred 4
    filename ( "repere-R.nff" )
  }

  constraint {
    link 7
    frame 8
  }

  constraint {
    frame 9
    sensor ( 0 1 )
  }

} // end of robot hybride
```

**Figure 4-9:** *Fichier MAD du robot hybride planaire*

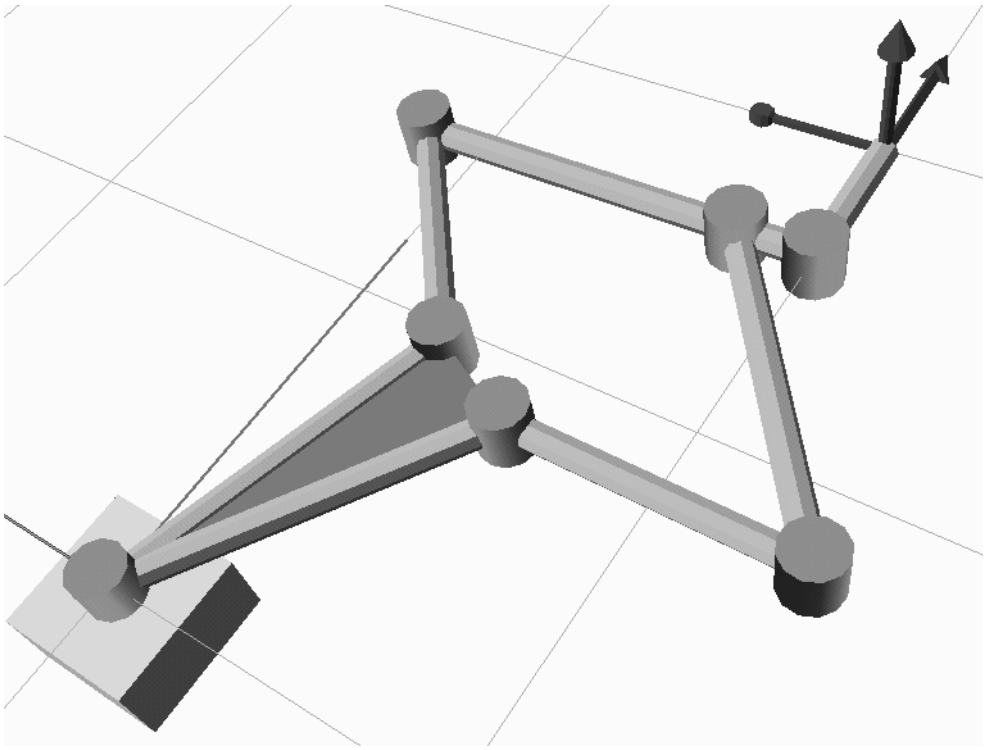


**Figure 4-10:** *Exemple de simulation du robot hybride planaire*

## 4.3 Résumé

Ce chapitre détaille le format de fichier MAD qui permet la description des robots manipulateurs à structure quelconque. Le fichier contient la géométrie et la structure du (ou des) robot(s) à simuler. Le programme *VirtualRobot* construit automatiquement un environnement virtuel correspondant à cette description. Cet environnement permet de manipuler le robot virtuel suivant sa cinématique inverse en utilisant les senseurs déclarés dans ce même fichier MAD ou alors sa cinématique directe grâce à des panneaux de contrôle (cf. Section 6.1.2).

La deuxième partie de ce chapitre présente un exemple complet pour la création d'une simulation d'un robot planaire hybride: en commençant par sa description schématique jusqu'à aboutir à l'écriture du fichier MAD correspondant.

Le chapitre suivant concerne la réalisation de l'implémentation du programme informatique permettant de concrétiser les concepts théoriques décrits dans ce chapitre ainsi que le précédant (Chapitre 3, "Concepts théoriques").

CHAPITRE

# 5

# IMPLÉMENTATION DE L'OUTIL

Une partie importante du projet «CINEGEN» consiste à développer un outil informatique qui montre la validité de l'approche choisie pour la simulation en temps réel de cinématique de robots manipulateurs. Le terme "outil informatique" est utilisé ici pour parler d'un ensemble plus complet qu'un simple "programme informatique". En effet l'outil développé s'appuie sur une architecture bien spécifique, qui regroupe aussi bien une partie matérielle que logicielle et fait appel à d'autres programmes ou bibliothèques existants.

Le programme *VirtualRobot* qui est la partie centrale de cet outil, comporte ses propres particularités dues à l'utilisation d'environnements virtuels qui nécessitent la visualisation d'une scène tridimensionnelle à une fréquence suffisamment élevée (25Hz). Il faut combiner ces particularités avec les besoins de la simulation de structures articulées qui impliquent des algorithmes mathématiques imposant des contraintes de temps de calcul. Ce chapitre se propose de décrire le type de plateforme logicielle et matérielle retenue pour le développement ainsi que les outils employés pour la réalisation du projet. La partie centrale porte sur l'architecture du programme, la relation entre les différents modules et les méthodes utilisées pour les réaliser.

## 5.1 Spécificité d'une application VR

Un programme qui met en oeuvre une interface basée sur la réalité virtuelle possède certaines caractéristiques particulières qui sont propres à ce type de d'environnement, ceci quelles que soient les bibliothèques graphiques utilisées.

En premier lieu, toute application VR est basée sur une couche logicielle permettant de réaliser des animations graphiques tridimensionnelles en temps réel. Cette bibliothèque regroupe des fonctionnalités de haut niveau pour manipuler et afficher des objets dans un espace tridimensionnel. Elle est construite sur une couche graphique de plus bas niveau qui contient tous les appels graphiques de base nécessaires à la visualisation.

La mise en oeuvre d'un environnement virtuel peut s'effectuer suivant diverses variantes, toutefois certains points clefs sont toujours traités:

- nécessité d'obtenir une simulation qui fonctionne en temps réel pour l'opérateur, c'est-à-dire à une fréquence supérieure à 25Hz environ,
- une boucle de simulation qui gère les événements, les actions et l'affichage,
- l'organisation des données graphiques dans un graphe de scène[1] structuré.

### 5.1.1 Temps réel

La caractéristique première d'une application VR est de pouvoir fournir des images du monde virtuel à une cadence suffisamment élevée. Le temps réel pour une simulation graphique implique d'avoir une fréquence de rafraîchissement supérieure à 25Hz – "fréquence de fusion" de l'oeil humain – afin d'assurer une visualisation sans saccade. Toutefois, la mise à jour des objets virtuels en fonction des actions de l'utilisateur peut-être légèrement plus lente sans dégrader de manière significative les capacités de l'opérateur à effectuer une tâche donnée. Par exemple une expérience réalisée pour mesurer les performances humaines à attraper une balle virtuelle en fonction de la fréquence de rafraîchissement [Richard 96] montre qu'il n'existe pas de différence significative entre 28Hz et 14Hz. Toutefois le domaine des facteurs humains est encore très étudié, et nous retiendrons simplement qu'une simulation fonctionnant à une fréquence proche de 25Hz procurera des résultats satisfaisants[2].

Atteindre ces performances implique évidemment des contraintes matérielles et logicielles. D'autant plus qu'à part l'aspect purement visualisation, il faut tenir compte de toutes les autres tâches que le système aura à effectuer. Aucune tâche de l'application ne doit ralentir excessivement ou perturber de manière importante la cadence de visualisation.

Plus précisément, toute action, tout calcul qui aura une répercussion sensible sur la scène visualisée doit pouvoir être effectué dans un laps de temps suffisamment court pour que ce changement s'effectue avec fluidité. On peut imaginer découpler totalement le processus de visualisation des autres calculs importants en espérant obtenir un rafraîchissement graphique suffisant. Dans cette situation, si le processus qui modifie par exemple la position d'un objet envoie des nouvelles positions à une cadence insuffisante au processus de visualisation, on obtiendra finalement un mouvement haché[3].

1. Certaines bibliothèques graphiques bas niveau qui ne fournissent pas de graphe de scène peuvent tout de même être utilisées si le développeur programme cette fonctionnalité.
2. 25Hz sont suffisants pour la partie visualisation. Les variations de forces sont discernables jusqu'à 20-30Hz [Burdea 96], ce qui implique un contrôleur fonctionnant à 300Hz environ. Par contre les sensations tactiles (du doigt humain) sont perceptibles jusqu'à 5 ou 10 KHz [Shimoga 92]!
3. Sauf si le processus de visualisation effectue une interpolation. Toutefois interpoler par exemple les positions d'un objet n'est pas envisageable si l'utilisateur est inclus dans le processus de simulation car cela entraînerait un retard pur.

Cette contrainte imposant que tout changement élémentaire dans la scène à visualiser doit être effectué en temps réel limite évidemment les possibilités de traitement par rapport à une simulation dans laquelle tout est pré-calculé. Il est important de composer avec cette contrainte fondamentale des environnements virtuels.

### 5.1.2 La boucle de simulation

Tout programme impliquant un environnement virtuel comporte normalement une boucle de simulation qui est répétée continuellement jusqu'à la fin du programme. Même si certaines bibliothèques comme Open Inventor [Wernecke 93] se comportent un peu différemment au niveau utilisateur, elles comprennent toujours une boucle de simulation interne.

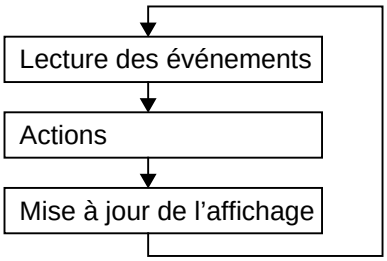



**Figure 5-1:** *Boucle de simulation*

La Figure 5-1 représente les trois phases principales d'une boucle de simulation typique. La première étape consiste à lire tous les événements; c'est-à-dire les différents senseurs (comme la souris) mais aussi des événements qui proviennent de programmes externes ou du réseau. Une fois ces événements lus et analysés, le programme effectue les actions nécessaires correspondantes (comme déplacer un objet selon le mouvement de la souris). C'est pendant cette période que vont aussi être effectuées toutes les tâches définies par le programmeur pour obtenir le comportement adéquat de l'application (dans le cas de *VirtualRobot* ce sont essentiellement les calculs cinématiques). Finalement, la dernière phase d'une boucle de simulation consiste à la mise à jour de tous les objets graphiques et à leur affichage correct suivant les paramètres en cours. Cette partie est plus ou moins prise en charge automatiquement suivant le niveau de la bibliothèque graphique.

Il est donc clair que le programmeur d'une application réalité virtuelle intervient essentiellement dans la deuxième phase, et qu'il doit réussir à réduire le temps de calcul de cette étape au maximum afin de ne pas perturber la simulation.

### 5.1.3 Le "graphe de scène"

Pour effectuer le rendu graphique, la boucle de simulation utilise l'ensemble des modèles tridimensionnels et leurs propriétés qui sont stockés classiquement dans un "graphe de scène". Le *graphe de scène* définit l'ensemble de la structure des différents objets (concrets et abstraits) contenus dans une scène graphique.

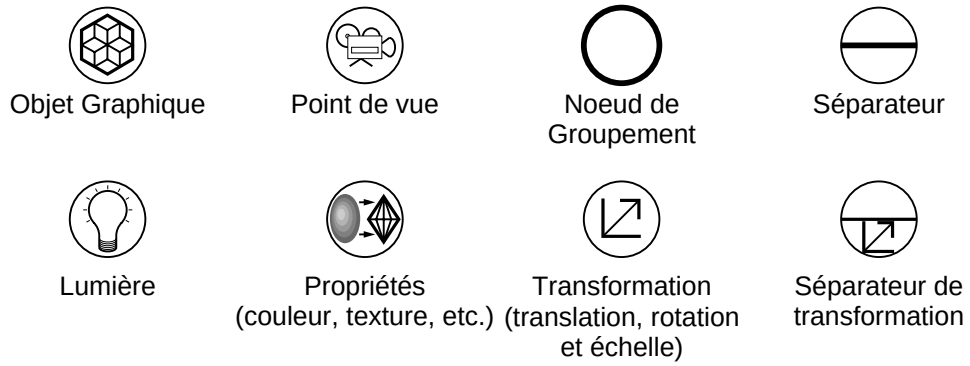



**Figure 5-2:** *Noeuds principaux utilisés dans un graphe de scène*†

†. Ces types de noeuds correspondent à ceux définis par WorldToolKit [Sense8 98], de légères variations de définition existent dans d'autres bibliothèques graphiques comme Open Inventor [Wernecke 93], mais les fonctionnalités restent les mêmes.

Le graphe de scène est une collection ordonnée de noeuds. Ces noeuds, de types divers, servent à décrire les objets et leurs relations dans une scène graphique. Les différentes bibliothèques graphiques peuvent proposer certains noeuds spécifiques, mais les principaux se retrouvent dans toutes, ils sont représentés à la Figure 5-2.

Un type de noeud indispensable est celui qui contient une description de la géométrie d'un objet graphique. Ensuite viennent les noeuds qui permettent de positionner les différents éléments dans l'espace 3D et d'appliquer des transformations (translations, rotations, variation d'échelle) sur les géométries ou de grouper des ensembles. Enfin certains noeuds, comme des lumières, caméras ou propriétés de surface, servent à définir le rendu final de la scène.

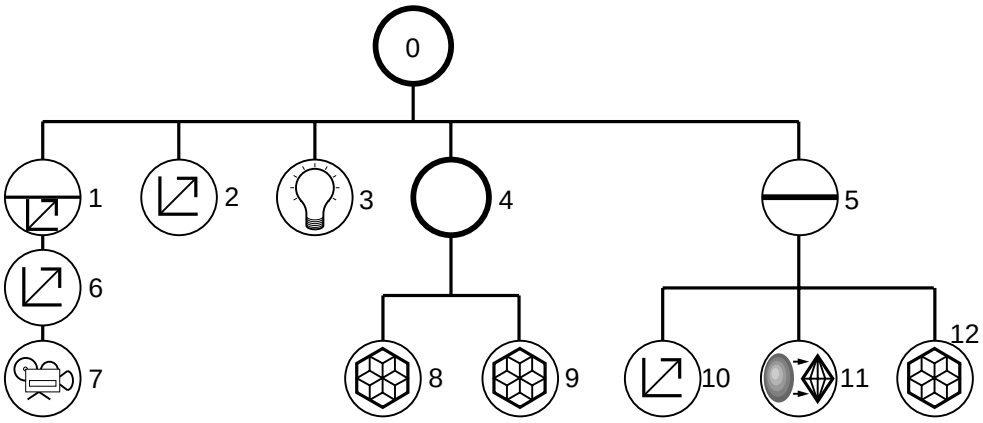



**Figure 5-3:** *Graphe de scène typique*

Avant chaque nouvel affichage de la scène, le programme va parcourir le graphe de scène afin de positionner tous les éléments graphiques dans la scène et de les dessiner avec les bonnes propriétés. Classiquement, un graphe de scène se "traverse" de haut en bas et de gauche à droite. Par exemple, dans la Figure 5-3 le noeud caméra (7) est soumis à une transformation locale (6), alors que la transformation (2) s'applique à tous les noeuds à sa droite: (3), (4), (8) et (9). Toutefois, la géométrie (12) ne subit ni la transformation (2), ni

la lumière (3) car elle est isolée du reste de la scène par le séparateur (5). Elle n'est alors sensible qu'à la transformation (10) et aux propriétés (11).

## 5.2 Plateforme de développement

Une "plateforme" est la combinaison de l'équipement matériel informatique et du système d'exploitation

Il existe une multitude d'environnements de développement de programmes. Chacun possède ses caractéristiques propres qui le prédisposent à certaines applications, tout en respectant plus ou moins bien les contraintes externes.

### 5.2.1 Besoins

Le projet «CINEGEN» génère des besoins informatiques bien particuliers. La partie concernant la résolution de cinématiques de robots nécessite une puissance de calcul suffisante pour effectuer de nombreuses opérations matricielles dont l'inversion d'une matrice de taille moyenne à chaque cycle. L'aspect environnement virtuel requiert des capacités graphiques temps réel. Ceci ne peut être envisagé qu'à l'aide de matériel informatique dédié. Enfin le caractère interface homme-machine suppose la possibilité de pouvoir connecter de nombreux périphériques de saisie à la station de travail.

La constante de temps d'un contrôleur de robot est de l'ordre de 10ms à 0.5ms.

Il faut noter que dans le cas d'une simulation graphique, le "temps réel" possède une constante de temps d'environ 40ms afin d'avoir une fréquence de rafraîchissement de 25Hz. Ce temps réel particulier est d'une part très exigeant au niveau du rendu graphique, et d'autre part relativement généreux au niveau du nombre de calculs cinématiques à faire, comparé à un contrôleur de robot. Toutefois, «CINEGEN» doit être capable de gérer plusieurs robots à la fois lors d'une simulation multi-robots.

### 5.2.2 Station de travail

*VirtualRobot* est développé sur plateforme Silicon Graphics. Ce choix est d'abord historique, mais les raisons qui le justifiaient alors restent valables malgré l'expansion des performances des ordinateurs et des composants graphiques ces dernières années.

Les développements antérieurs[4] [Flückiger 94] [Piguet 94] [Natonek 95] qui ont initié «CINEGEN» datent de 1993-1994 et seul Silicon Graphics pouvait alors proposer des stations de travail permettant la simulation de mondes virtuels en temps réel. En effet ces stations étaient équipées de cartes graphiques spécifiques permettant de gérer toutes les transformations de coordonnées ainsi que les projections nécessaires à un rendu d'objets tridimensionnels. Cette particularité laisse un maximum de temps au processeur principal pour s'occuper d'autres tâches.

Aujourd'hui des "compatibles PC" haut de gamme – sur base de processeurs Intel ou équivalents – peuvent êtres équipés de cartes graphiques à haute performance ce qui leur permet de rivaliser avec le bas de gamme des

4. Un environnement de téléopération basé réalité virtuelle pour un robot manipulateur spécifique à cinq degrés de liberté.

stations Silicon Graphics. Dans un souci de commercialisation de *VirtualRobot* il serait donc envisageable d'utiliser des plateformes PC pour un développement futur.

Il faut noter que les capacités de calcul des processeurs augmentent considérablement, et parallèlement les performances des cartes graphiques aussi. Donc l'application gagne en rapidité sur les deux plans. D'autre part comme le temps réel d'une simulation graphique n'est pas aussi rigide que celui d'un contrôleur de robot, l'application gagne simplement en fluidité et en temps de réponse avec l'amélioration des performances: trois ans auparavant nous nous contentions d'un rafraîchissement à 10Hz sur une station de moyenne gamme, alors que nous pouvons obtenir aujourd'hui 20Hz sur une station bas de gamme.

### 5.2.3 Système d'exploitation

Il existe plus d'une douzaine de systèmes d'explotation reconnus comme sérieux; ils présentent chacun des caractéristiques variées. Pour le projet «CINEGEN» le choix est simplement dicté par l'utilisation des stations de travail Silicon Graphics: Unix, ou plus précisément une version spécifique d'Unix système V appelée IRIX [Silicon Graphics Inc. 98].

Unix est sans doute un des meilleurs choix pour le développement d'applications dans un cadre de recherche. Ceci de par la taille de la communauté scientifique utilisant ce système d'exploitation, et donc du grand nombre de bibliothèques et de programmes distribués, mais aussi par la puissance et les outils qu'offre ce système d'exploitation ainsi que par sa fiabilité.

La maîtrise du système Unix requiert un apprentissage conséquent, qui est toutefois récompensé par l'éventail des possibilités qu'il ouvre ainsi que par l'efficacité formidable qu'il offre (par rapport à des système d'exploitation dit plus "conviviaux"). L'investissement en temps vaut donc certainement la peine, en tout cas dans un milieu universitaire.

### 5.2.4 Langage de programmation

Le choix du langage de programmation peut devenir un casse-tête si l'on ne s'arrête pas à des arguments très pragmatiques. En effet les argumentations sur la "pureté" d'un langage plutôt qu'un autre sont tout à fait valables, mais il faut mesurer leur utilité dans un cadre comme celui de «CINEGEN».

*VirtualRobot* est totalement écrit en C++ pour les raisons principales citées ci-dessous.

- Langage compilé très performant au niveau de la vitesse d'exécution.
- Permet d'utiliser toutes les bibliothèques externes imposées par l'application (principalement la bibliothèque graphique 3D, indispensable, écrite en C).

- Large choix de bibliothèques non-commerciales et commerciales disponibles, performantes et fiables (ceci est dû à l'ancienneté du langage).
- Approche objet permettant un meilleur découplage des différentes fonctionnalités et donc une meilleure compréhension et fiabilité.
- Mécanismes objets comme abstraction de données, classes, héritage, polymorphisme, liaison dynamique, etc. [Eckel 95] [Stroustrup 97] (non disponibles dans le langage C qui satisfait par ailleurs les trois premiers points).

## 5.3 Architecture logicielle

L'élaboration d'un programme de moyenne envergure nécessite une structure méthodique et cohérente. Cette structure conditionne la facilité de mise en oeuvre, les performances et la fiabilité du programme, mais aussi ses possibilités d'extensions futures et son ouverture vers d'autres programmes.

Afin de bien comprendre comment le programme est construit et se déroule, il faut tout d'abord étudier son architecture au niveau global. Ce chapitre montre quels sont les différents modules constituant le programme *VirtualRobot* ainsi que la structure adoptée pour le stockage et la manipulation des données principales. Cette analyse permet de mettre en évidence les outils qui sont nécessaires pour la réalisation des différents modules.

Parallèlement à cette analyse interne du programme, il faut traiter l'interface graphique classique, qui est un élément important de tout programme se voulant "interactif". L'interface graphique, si elle n'intervient pas dans le cœur du programme, doit toutefois s'intégrer de manière efficace avec le reste des fonctionnalités.

### 5.3.1 Organisation des modules

Le programme *VirtualRobot* est orienté objet et comprend une boucle principale d'événements à caractère séquentiel, appelant différentes méthodes: le cœur du programme traite les données entrées et fournit des résultats à caractère essentiellement graphique en retour. La Figure 5-4 montre les principaux modules et leurs relations ainsi que les différentes entrées et sorties.

#### 5.3.1.1 "Parser"

Ce premier module *parser* (analyseur de texte) lit et analyse un fichier de description de robots (MAD, cf. Section 4.1) afin de construire les structures cinématiques qui y sont décrites. Comme la plupart des parsers, ce module commence par découper le texte du fichier en *token*s (unité syntaxique élémentaire), puis vérifie si l'agencement des ces tokens satisfait la grammaire qui régit un fichier de type MAD. Chaque fois qu'une règle de grammaire est validée, une action est alors effectuée. Cette action peut avoir comme paramètres les différentes valeurs des tokens qui forment cette règle.

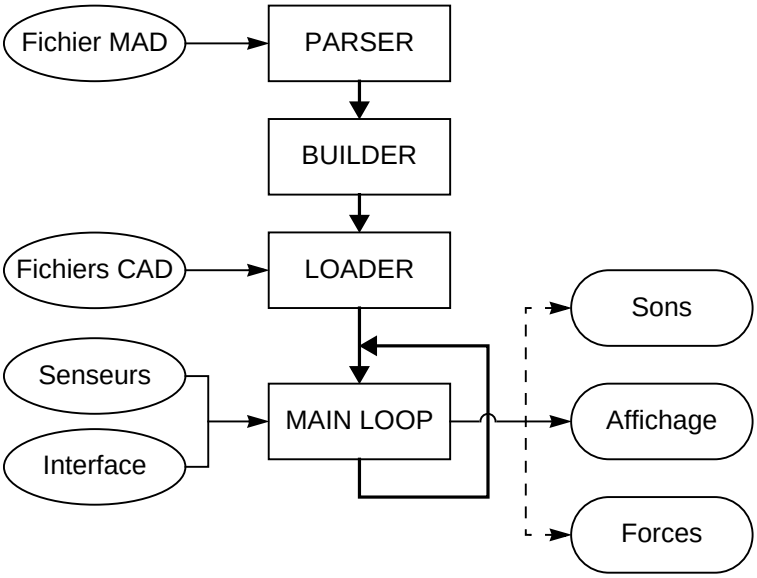



**Figure 5-4:** *Organigramme du programme*

Par contre, si certaines suites de tokens ne vérifient aucune règle connue du parser, alors une procédure d'erreur est appelée. Cette procédure peut simplement afficher un message d'erreur puis rendre le contrôle au parser, arrêter définitivement le programme ou si elle possède une gestion plus évoluée essayer de récupérer des erreurs courantes sur la base d'a priori.

#### 5.3.1.2 "Builder"

Après avoir analysé le fichier de description, la deuxième étape consiste à construire une représentation de la structure cinématique, manipulable par le programme, avec tous ses paramètres en mémoire.

Deux approches sont envisageables pour effectuer la liaison entre le parser et le builder:

1) le parser construit un arbre provisoire de toutes les règles de grammaire trouvées, puis passe cet arbre (contenant toutes les valeurs des tokens valides) au builder. Ainsi, si l'arbre n'est pas valide, aucune action de construction n'est entreprise.

2) le parser appelle les fonctions nécessaires du builder à chaque fois qu'une règle de grammaire est unifiée. Dans ce cas, si une sous-règle échoue, il faudra annuler les actions précédentes.

*VirtualRobot* utilise la seconde méthode car la profondeur d'imbrication des règles est faible et la liaison entre une règle et une action à effectuer est normalement directe. Donc la tâche du builder ne se fait pas réellement après l'analyse du fichier, mais en parallèle puisqu'à chaque règle de grammaire validée une action de construction est appelée[5]. Toutefois ces routines de

5. Les routines du builder peuvent être appelées par la suite dans la boucle principale du programme, dans le cas où l'on change dynamiquement la structure cinématique d'un robot.

construction appartiennent toutes à un module bien séparé du parser: ce sont des méthodes des différents objets construits.

#### 5.3.1.3 “Loader”

Une fois que les robots sont construits en mémoire, et vérifiés (c’est-à-dire que non seulement les règles sont valides, mais que les paramètres aussi sont cohérents), le loader s’occupe de charger tous les fichiers graphiques représentant les différents composants de la scène. Ce sont essentiellement les différentes parties du robot, mais aussi d’autres objets graphiques qui peuvent être utilisés pour représenter des entités abstraites comme un référentiel.

Dans le cas où l’utilisateur spécifie que la représentation d’un élément de robot doit se faire automatiquement, le loader s’occupe de construire dynamiquement des objets graphiques qui vont représenter cet élément de robot.

Le loader sera donc essentiellement composé d’appels de routines de la bibliothèque graphique. Il faut toutefois stocker tous les objets graphiques de manière indépendante dans le graphe de scène et construire tous les noeuds supplémentaires nécessaires à l’organisation de ces objets.

#### 5.3.1.4 “Main Loop”

La boucle principale est véritablement le cœur du programme: cette partie correspond à la boucle de simulation d’un environnement virtuel. C’est-à-dire qu’elle regroupe la lecture de tous les événements et effectue les actions nécessaires avant de réactualiser l’affichage. Toutefois, cette boucle de simulation est beaucoup plus importante que dans un programme classique de réalité virtuelle. En effet, elle doit effectuer tous les calculs nécessaires aux transformations cinématiques de chaque robot. Il faut en effet à chaque boucle analyser les contraintes dans la structure cinématique, construire la matrice jacobienne augmentée correspondant et l’inverser afin d’obtenir la nouvelle posture de chaque lien en fonction des consignes données par l’utilisateur.

### 5.3.2 Structure d’un robot

Le stockage des données en mémoire par le programme revêt une grande importance car il conditionne la facilité et la rapidité avec lesquelles elles vont pouvoir être exploitées. D’autre part cette structure doit être facilement compréhensible par le programmeur pour des modifications futures et surtout être extensible et ouverte. Dans notre cas, la structure est très proche des différents constituants d’un robot manipulateur, et en même temps présente un découpage en objets présentant une flexibilité maximum au niveau de la programmation.

Etant donné la structure arborescente choisie pour la description des robots (cf. Section 3.4.1) il serait tentant d’utiliser le graphe de scène pour stocker les différents objets composants un robot. Dans une approche orientée-objet, chaque `lien` d’un robot pourrait simplement hériter d’un noeud graphique auquel on ajouterait les paramètres et fonctionnalités nécessaires. Toutefois ceci est impossible car la structure en arbre ouvert des

graphes de scènes ne permet pas de représenter les propriétés des robots contenant des boucles. De plus, utiliser un graphe de scène existant rendrait le cœur de l'application complètement dépendante de la bibliothèque choisie. Finalement cette option serait relativement restrictive car étendre les fonctionnalités d'un graphe de scène existant serait compliqué.

Le choix d'une structure de données propre à *VirtualRobot* s'est donc imposé. Il serait en fait plus judicieux de parler *des* structures de données du programme car, en plus de la structure fondamentale du robot, il contient des catégories d'objets utiles pour le reste du fonctionnement du programme:

- tous les objets relatifs à l'application graphique, comme les fenêtres ou les éléments de l'interface (boutons, etc.); nous n'en parleront pas ici,
- la liste des robots contenus dans la scène graphique courante,
- la liste des senseurs qui vont permettre d'interagir avec les robots.

Nous verrons dans la Section 5.5 comment ces objets sont implémentés, il faut juste retenir ici que ces différentes catégories ont besoin de se passer des messages pour fonctionner ensemble. L'objet `senseur` est relativement simple dans sa structure puisqu'il permet simplement d'interfacer un périphérique physique avec les données du programme. Par contre l'objet `robot` est évidemment plus complexe et comprend plusieurs autres objets comme le montre l'exemple de la Figure 5-5.

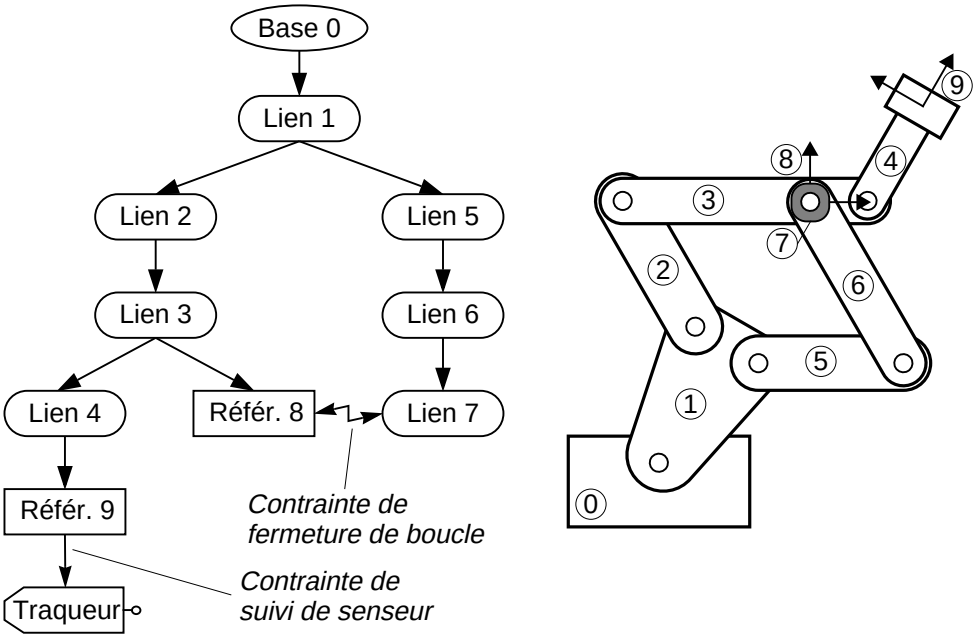



**Figure 5-5:** *Exemple de structure d'un robot*

La structure de données représentant un robot en mémoire est directement le reflet de la structure théorique d'un robot (cf. Section 3.4.1): un graphe. Les objets composants ce graphe sont la représentation informatique des objets d'un fichier MAD (cf. Section 4.1): une base, des liens, des référentiels et des traqueurs.

#### 5.3.2.1 Gestion du robot par un “graphe”

Un graphe orienté est formé avec des arêtes unidirectionnelles qui relient un noeud source à un noeud destination.

Le graphe représentant un robot est une collection de noeuds et d’arrêtes. Dans notre cas, les noeuds peuvent êtres de types différents, de même que les arêtes peuvent avoir différentes significations. Le graphe est orienté ce qui permet de conserver la hiérarchie. Cette hiérarchie, ou ordre, permet de traverser le graphe de manière très rapide, et aussi de retrouver les ascendants de n’importe quel noeud.

Les noeuds stockent les différents objets constituant un robot. L’utilisation d’un noeud générique dont sont dérivés tous les autres types de noeuds permet l’ajout très simple d’un nouveau type si ce besoin apparaît.

Les arêtes représentent les relations entre les noeuds. Leur type est simplement codé par un nombre qui permet de différencier des liaisons entre liens ou les différentes contraintes.

#### 5.3.2.2 L’objet Base

L’objet `base` représente la base d’un robot, c’est-à-dire l’élément unique auquel est attaché tout le reste de la structure. On pourrait imaginer d’utiliser simplement un lien normal pour représenter la base, pour rester plus homogène. Toutefois, l’objet `base`, qu’il soit la représentation d’un élément réel ou non, possède suffisamment de caractéristiques uniques pour le différencier des autres liens:

- il est unique,
- il ne possède pas de parent,
- il est commode de le positionner dans le monde non pas par les paramètres Kleinfinger-Khalil, mais par des coordonnées de position et d’orientation.

D’autre part, la différenciation de l’objet `base` par rapport aux liens permettra facilement d’étendre ses fonctionnalités par la suite. Par exemple, il pourra devenir l’élément de liaison avec un objet “robot mobile”. Ainsi le lien entre un simulateur de robot mobile et *VirtualRobot* sera direct. L’objet `base` héritera simplement des propriétés d’un robot mobile et pourra ainsi se mouvoir dans l’espace, en entraînant toute la structure du robot manipulateur qui lui est attachée. Cette application est particulièrement intéressante dans le cadre de la collaboration avec la NASA qui utilise des plateformes mobiles supportant des bras manipulateurs.

#### 5.3.2.3 L’objet Lien

L’objet `lien` est le constituant essentiel d’un robot. Il contient les paramètres Kleinfinger-Khalil du lien ainsi que des matrices de transformation. Le stockage des matrices de transformation entre le lien courant et son prédécesseur ainsi que par rapport au référentiel monde permet de gagner du temps lors de calculs partiels de la chaîne cinématique. Ceux-ci peuvent s’avérer nécessaires par exemple si l’on décide de mettre à jour seulement une sous-chaîne partielle terminale. Si le stockage permanent de la matrice de transformation monde-lien augmente la taille de l’objet il permet néanmoins

de limiter les calculs. Même si ce lien est "visité" par plusieurs chemins différents la matrice n'est calculée qu'une seule fois.

#### 5.3.2.4 L'objet Référentiel

L'objet `référentiel` a été introduit d'abord par commodité pour l'utilisateur qui doit décrire une nouvelle structure. Il définit un changement de repère entre son père et ses descendants. Cela évite de chercher parfois des paramètres Kleinfinger-Khalil compliqués. Par exemple pour une structure bouclée ternaire comme le robot Delta, il suffit de décrire trois chaînes cinématiques identiques, puis de les positionner correctement par rapport à la base grâce à des référentiels. Sans l'usage de l'objet `référentiel`, il faudrait utiliser les deux paramètres optionnels pour les branches de Kleinfinger-Khalil, et la description des trois chaînes ne serait alors pas homogène.

L'objet `référentiel` peut aussi être utilisé pour attacher un objet graphique à une partie spécifique d'un robot. Dans ce cas l'origine de l'objet ne doit pas coïncider nécessairement avec l'origine du lien, mais peut être délocalisée à l'envi par l'utilisateur.

L'utilisation la plus intéressante de l'objet `référentiel` est le contrôle total qu'il offre pour lier un senseur à un robot, c'est-à-dire la manière par laquelle le senseur va donner des consignes au robot. En effet, attacher un senseur à un lien est très limitatif puisque qu'il va agir au point d'origine de cette articulation. Pour contrer ce problème, lorsque le bras d'un robot se termine par un outil, il est possible d'ajouter un `référentiel` qui positionnera l'outil par rapport au dernier lien. Il suffit alors d'attacher le senseur à ce référentiel terminal, et le robot sera vraiment contrôlé au niveau de son outil. Cela permet aussi dans le cas d'un changement d'outil de changer uniquement les paramètres de ce référentiel outil, et de ne pas intervenir sur les paramètres des liens.

#### 5.3.2.5 Traqueur

Cet objet spécial n'est pas défini dans les fichiers MAD, mais est créé automatiquement lorsqu'une contrainte entre un élément du robot et un senseur est rencontrée. L'objet `traqueur` va contenir une liste de senseurs potentiels qui peuvent contrôler cet élément du robot. Son utilité est de séparer complètement la structure de données d'un robot de celle des senseurs (la même liste de senseurs du monde peut être utilisée par plusieurs robots). Ainsi toutes les contraintes internes (fermeture de boucles) et externes (suivre un senseur) sont contenues dans le graphe du robot[6]. En outre, ceci permet de retrouver beaucoup plus facilement les contraintes dans le graphe pour l'algorithme de cinématique inverse.

6. Ce découplage ne fonctionnerait pas si l'on autorisait certains objets à pointer directement sur un senseur (afin d'éviter une classe supplémentaire). D'autre part cela alourdirait la structure de plusieurs objets et obligerait donc tous les objets en question de connaître la classe senseur (couplage fort).

### 5.3.3 Interface graphique conventionnelle

Une application basée sur un environnement virtuel utilise classiquement uniquement son interface de type tridimensionnelle (visualisation sur un écran d'objets tridimensionnels) pour interagir avec l'utilisateur puisque l'on essaie d'immerger celui-ci dans une scène virtuelle. Toutefois, si l'on veut lui fournir une palette variée d'outils et de nombreux modes de travail différents, il devient difficile de lui proposer tous ces choix directement dans l'environnement virtuel. On peut par exemple créer des menus tridimensionnels ou des fenêtres flottant dans l'espace, mais l'expérience montre que souvent une interface graphique classique 2D est plus efficace. Ceci provient de l'entraînement des utilisateurs face à des interfaces classiques, mais aussi du trop grand nombre de degrés de liberté à gérer dans un monde tridimensionnel. Dans le cas de «CINEGEN» qui est une application plus orientée vers l'ingénierie, une interface 2D classique à base de boutons, sélecteurs et ascenseurs est proposée en plus de l'environnement virtuel. Il faut avouer qu'une telle interface est un outil efficace non seulement pour l'utilisateur, mais augmente la productivité du développeur qui peut tester beaucoup plus rapidement certains concepts.

L'interface graphique conventionnelle 2D prend donc une place importante dans le projet «CINEGEN» et il est essentiel qu'elle puisse communiquer efficacement avec le programme *VirtualRobot*.

## 5.4 Bibliothèques utilisées

Un projet informatique de moyenne envergure comme l'est *VirtualRobot* s'appuie forcément sur des bases logicielles existantes. Il s'agit alors de faire un choix entre la réécriture de modules bien spécifiques à l'application, et la réutilisation de modules existants.

Si le problème est relativement simple, il est souvent moins onéreux en temps d'écrire les fonctions dont on a besoin. Elles pourront être optimisées pour la tâche à effectuer et totalement contrôlées par les programmeurs. Arrive un niveau de complexité où il devient intéressant d'exploiter des bibliothèques créées pour des besoins plus généraux. Il faut alors un investissement en temps (éventuellement en argent) pour maîtriser cette bibliothèque ou faire fonctionner un nouvel outil. Cet investissement apporte le bénéfice d'outils possédant plus de fonctionnalités et plus fiables. Le choix correct d'une bibliothèque est fondamental pour obtenir toutes les fonctionnalités nécessaires ainsi qu'une fiabilité suffisante. La documentation accompagnant l'outil devrait toujours être aussi un critère déterminant pour ce choix.

L'utilisation d'une bibliothèque existante peut aussi améliorer la portabilité d'un programme. C'est ainsi que le choix d'une bibliothèque graphique qui existe et fonctionne sur de multiples plateformes avec la même interface, permettra le portage du code écrit plus facilement. En effet, moins le programmeur écrira de code spécifique à une machine – grâce à l'utilisation d'un maximum de bibliothèques portables – plus son application elle-même

deviendra facilement portable. Cela peut être un critère de qualité et un avantage certain pour la diffusion d'une application.

*VirtualRobot* s'appuie largement sur des bibliothèques existantes afin d'obtenir le meilleur de plusieurs outils reconnus pour leurs performances. La bibliothèque la plus importante dans ce cas présent est bien sur la bibliothèque graphique qui va s'occuper de la gestion des objets 3D et de l'affichage. L'interface graphique conventionnelle se base aussi évidemment sur une bibliothèque dédiée. De plus, d'autres fonctionnalités, comme le parser ou les calculs matriciels, sont également remplies par des outils et bibliothèques appropriées.

### 5.4.1 Bibliothèque 3D: WorldToolKit

Une bibliothèque de bas niveau contient des fonctions basiques. Une bibliothèque de haut niveau d'abstraction possède une *API* évoluée.

Toute application mettant en oeuvre de la visualisation graphique 3D s'appuie sur une bibliothèque graphique de plus ou moins haut niveau d'abstraction. Utiliser des routines très proches de la couche matérielle réalisant les calculs et le rendu graphique permet d'optimiser au maximum les performances. A l'autre extrême, utiliser une bibliothèque de haut niveau d'abstraction réduit drastiquement la taille du code à écrire pour le développeur. Si de par sa généralité, une bibliothèque de haut niveau n'atteint pas forcément les performances d'une bibliothèque de bas niveau, son utilisation dans la pratique pour un programme de large envergure peut finalement permettre des performances égales voire meilleures. En effet pour des applications complexes, une mauvaise mise en oeuvre de multiples routines de bas niveau peut être catastrophique, alors que la bibliothèque de haut niveau peut optimiser certaines tâches en regroupant des mêmes types d'appels (puisqu'elle possède une vue d'ensemble).

OpenGL ou Direct3D sont des exemples de bibliothèques 3D bas niveau alors qu'Open Inventor ou WorldToolKit sont des bibliothèques 3D de haut niveau (construites sur OpenGL).

Pour créer un système de réalité virtuelle, il est nécessaire d'utiliser une bibliothèque graphique de haut niveau d'abstraction étant donné la complexité des scènes à représenter. Soit la bibliothèque graphique choisie possède déjà une *API* puissante, soit les développeurs doivent créer leur propre API de haut niveau d'abstraction afin d'encapsuler des ensembles de fonctions élémentaires et proposer une gestion des données évoluée.

Le concept de réalité virtuelle implique une interaction forte avec l'utilisateur. Ce problème ne s'arrête donc pas uniquement à la visualisation graphique de modèles 3D, mais nécessite une gestion des périphériques externes d'entrée et sortie. En outre, un environnement de programmation pour la réalité virtuelle doit non seulement gérer des objets graphiques, mais leur organisation et leur persistance. Il faut pouvoir assigner des comportements aux objets, permettre la gestion de "chemins" dans l'espace ou encore gérer de multiples points de vues et fenêtres. La génération d'un environnement sonore peut aussi être inclus dans un bon environnement de programmation. Finalement, des routines de haut niveau pour gérer du texte dans l'espace sont bienvenues pour ajouter des informations supplémentaires à la scène.

"WorldToolKit is a cross-platform software development system for building high-performance, real time, integrated 3D applications for scientific and commercial use." [Sense8 98]

*VirtualRobot* est basé sur une bibliothèque 3D commerciale de haut niveau: WorldToolKit de Sense8 [Sense8 98] [wwwSEN]. Cette bibliothèque orientée objet écrite en C (elle ne permet donc pas l'héritage ou la liaison dynamique), contient plus de 1000 fonctions de haut niveau pour configurer, interagir et contrôler des simulations graphiques en temps réel. Plus de 20 classes composent WorldToolKit. Elles incluent entre autres un univers (qui gère la simulation et contient tous les autres objets), des objets géométriques, des points de vue, des senseurs, des chemins, des lumières et des objets texte. WorldToolKit est portable sur des plateformes SGI, Sun, HP, Dec, Intel et PowerPC et est optimisé pour utiliser les fonctions graphiques spécifiques à chaque plateforme. De plus WorldToolKit supporte une grande variété de périphériques d'entrée et sortie propres aux environnements virtuels.

La bibliothèque graphique de haut niveau WorldToolKit répond aux exigences de l'application *VirtualRobot* pour les raisons suivantes:

- elle régit la scène graphique par un graphe de scène,
- elle contient un "*wrapper*" C++ qui permet une bonne intégration avec *VirtualRobot*,
- elle autorise tous les modes de rendu utilisés (transparence, textures, etc.),
- elle gère l'affichage stéréo avec les lunettes CrystalEyes,
- elle supporte les divers périphériques 3D de saisie utilisés,
- elle assure des performances entièrement satisfaisantes sur la plateforme utilisée.

Toutefois, WorldToolKit comporte aussi des désavantages comme son prix ou quelques limitations techniques. Cependant à l'heure actuelle, peu de concurrents sont vraiment envisageables. Open Inventor, un des meilleurs candidats, fonctionne aussi sur plateforme PC (Windows NT) et est totalement écrit en C++, mais est beaucoup plus pauvre quant à ses fonctionnalités annexes: aucune gestion des périphériques 3D et gestion du fenêtrage de moins haut niveau

Il existe encore la possibilité de recourir à des bibliothèques graphiques développées par des groupes de recherche. Deux possibilités auraient été envisageables pour «CINEGEN»:

- libPTK (librairie Performer ToolKit) développée à l'ISR (EPFL) qui est une bibliothèque graphique extrêmement performante, formée de nombreux modules, construite sur Performer [wwwVIR].
- VEVI (Virtual Environment Vehicle Interface), développé à l'IMG (NASA Ames), outil de visualisation orienté réseau [Hine 95] [Piguet 95].

Cependant, ces environnements sont en perpétuelle évolution et leur documentation n'est encore que partielle. Ces raisons réduisent fortement l'intérêt de baser *VirtualRobot* sur ces bibliothèques pour des raisons d'efficacité et de fiabilité.

### 5.4.2 Interface 2D: Tcl-Tk et Tix

L'interface graphique utilisateur, appelée *GUI* (Graphical User Interface) par la suite, est un élément essentiel d'un programme, et c'est pourquoi «CINEGEN», bien que basé sur les environnements virtuels ne néglige pas l'interface 2D. Si l'interface paraît évidente à un utilisateur, c'est certainement qu'elle est bien conçue, et justement le temps de développement est en conséquence: il est très difficile et coûteux de créer une bonne GUI [Crampes 97].

De nombreuses bibliothèques existent pour mettre en oeuvre une GUI. De plus, des outils d'aide à la création de GUI sont proposés. Les bibliothèques les plus courantes dans le monde Unix sont X11, Xt (X Toolkit Intrinsics) et Motif [Young 95], alors que les MFC (Microsoft Foundation Classes) ou OWL (Borland Object Windows Library) sont le plus souvent utilisées pour le monde windows. Xforms [wwwXFO] ou RapidApp [wwwRAP] entre autres, sont des outils pour générer plus facilement des GUI basées sur X11 et Motif. La GUI est certainement la partie la plus sensible au portage entre plateformes étant donné qu'elle repose sur tout un ensemble de routines du système d'exploitation et du gestionnaire de fenêtres. Pour répondre à ce problème, certaines bibliothèques, comme Java (ou des dizaines d'autres), proposent une compatibilité inter-plateforme. Ces bibliothèques utilisent alors les ressources graphiques de chaque plateforme spécifique pour offrir une même interface unifiée.

Etant donné le large choix de GUI disponibles, il est difficile d'évaluer quelle sera la meilleure solution pour un problème donné. Le Tableau C-1 en Annexe C, "Comparaison des GUI pour *VirtualRobot*" montre les critères de choix qui ont été retenus dans le cas de «CINEGEN».

Le choix s'est porté sur une bibliothèque largement utilisée dans des projets importants, qui est gratuite et portable sur de nombreuses plateformes: Tcl/Tk. Cette bibliothèque possède de plus les avantages suivants:

- très grande facilité de mise en oeuvre et rapidité dans la création et l'ajustement de GUI,
- interface C qui permet de l'intégrer relativement aisément avec des programmes C et C++,
- nombreuses extensions disponibles (dessin de graphiques, intégration www, etc.) dont Tix pour améliorer la partie GUI.

#### *Tcl/Tk*

*Tcl/Tk* [wwwTCL] est un système de programmation développé par John Ousterhout à l'University of California, Berkeley. Ce système est facile à utiliser et possède une interface graphique très pratique. *Tcl* (**T**ool **C**ommand **L**anguage) est un langage de programmation interprété supportant la plupart des fonctionnalités des langages procéduraux classiques. *Tk* (**T**ool**K**it) apporte des fonctionnalités graphiques à Tcl, et ressemble à la plupart des autres GUI, mais reste très simple à utiliser. En fait Tcl/Tk est un système de programmation car c'est à la fois un langage interprété et une bibliothèque facilement extensible (en C) et intégrable dans un programme C ou C++. De

très nombreux projets utilisent Tcl/Tk comme outil pour créer leur GUI et/ou lier différents modules. Le lecteur se reportera donc à la bibliographie pour plus de détails [Johnson 96] et [Welch 97].

Un “widget”, littéralement “truc” est un composant graphique utilisé pour créer une GUI.

C’est essentiellement la partie Tk dont va se servir *VirtualRobot* pour créer sa GUI. Tk fournit un ensemble de “*widget*” (boutons, ascenseurs, fenêtres, cadres, champs de saisie, boîte à onglets, etc.) très facile à mettre en oeuvre. Si Tk n’offre pas la souplesse totale de disposition et de contrainte entre les éléments graphiques d’un système comme Motif, il s’affranchit de multiples paramètres fastidieux à régler et permet en un tour de main de créer une interface fonctionnelle en assemblant simplement les widgets de base.

#### *Tix*

*Tix* [wwwTIX] est l’une des nombreuses extensions apportées à Tcl/Tk, qui améliore la partie GUI et apporte un concept “objet” à Tk [Harrison 97]. Tix (**T**k **I**nterface E**x**tension) propose un ensemble de “*mega-widget*” utilisables directement, ainsi que les méthodes pour en construire de nouveaux. Un “mega-widget” est un nouvel objet graphique interactif composé de plusieurs widgets Tk ou Tix (un mega-widget peut être formé d’autres mega-widgets). Ce concept de mega-widget qui deviennent des objets graphiques à part entière et instanciables à volonté, est la base de l’interfaçage entre *VirtualRobot* et Tcl/Tk-Tix.

### 5.4.3 Parser: ANTLR

Comme de très nombreux programmes, *VirtualRobot* se sert d’un “mini-langage” pour définir une description de données. Le fichier MAD est utilisé dans ce but. Il est donc nécessaire d’avoir un analyseur syntaxique/grammatical qui va examiner le fichier de description. Cet analyseur est souvent écrit à la main lorsqu’il s’agit d’un petit projet ou que les performances sont cruciales. Cette approche devient vite fastidieuse lorsque le nombre de règles augmente. De plus, toute modification de la grammaire nécessite de changer les routines d’analyse, augmentant chaque fois le risque d’introduction d’erreurs.

C’est pourquoi l’emploi d’un générateur d’analyseur (“parser generator”) augmente grandement la vitesse de développement et de modification, ainsi que la fiabilité du code généré. Un générateur d’analyseur lit une grammaire de définition d’un langage (et éventuellement un vocabulaire) et génère un analyseur qui reconnaîtra les phrases de ce langage.

Dans le monde Unix, les outils Lex et Yacc et leurs versions GNU (Flex et Bison), sont largement employés pour générer des analyseurs. Malheureusement il s’agit de deux outils différents – Flex (Lex) pour la syntaxe et Bison (Yacc) pour la grammaire – qu’il faut combiner pour obtenir un analyseur complet. D’autre part, ces outils ne génèrent que du code C[7] (pas

7. Il existe un analyseur dérivé de Flex, Flex++ qui génère un code C++ (utilisé dans les premières versions de *VirtualRobot* comme analyseur syntaxique), mais il n’existe malheureusement à ce jour pas de Bison++.

de C++). De plus, si le code produit est performant, il est totalement illisible pour un humain, donc difficile à déverminer et impossible à modifier.

"ANTLR constructs human-readable recursive-descent parsers in C or C++ from pred-LL(k) grammars, namely LL(k) grammars, for k>1 that support predicates." [Parr 97]

Un générateur d'analyseur – puissant mais moins connu – est utilisé dans le cas de *VirtualRobot*. Il s'agit de ANTLR qui est l'acronyme de **AN**other **T**ool for **L**anguage **R**ecognition [Parr 97] [wwwANT]. ANTLR est un module d'un outil plus complet: PCCTS (Purdue Compiler Construction Tool Set). PCCTS comprend ANTLR ainsi qu'un générateur d'analyse d'arbres: SORCERER (dont nous ne parlerons pas).

ANTLR utilise une méthode "top-down", désignée LL(k), d'analyse syntaxique mise au point par son auteur. Classiquement, les analyseurs "bottom-up" LR ou LALR sont plus robustes que les analyseurs LL de par leur stratégie qui utilise plus d'informations sur le contexte. Toutefois un analyseur LL(k) peut être plus puissant par l'utilisation de prédicats pour analyser des langages sensibles au contexte, et par l'utilisation d'une profondeur d'analyse arbitraire.

Les fonctionnalités qui ont été décisives pour le choix de ANTLR comme générateur d'analyseur pour *VirtualRobot* sont les suivantes:

- intégration de l'analyse lexicale et syntaxique dans le même outil et le même fichier de définitions de vocabulaire/grammaire.
- intégration des actions à effectuer en fonction des règles unifiées directement dans le fichier de définitions du langage.
- passage de paramètres entre les règles de grammaires facilitant la communication d'attributs entres les différents objets.
- génération d'un analyseur en C++ structuré en classes correspondant aux règles de grammaire, et donc compréhensible par un humain.

### 5.4.4 Mathématique: Lapack.h++

Le projet «CINEGEN» implique une base mathématique; il est donc nécessaire de disposer d'outils adéquats pour traiter le problème de manière informatique. La majorité des calculs relève de l'algèbre linéaire de base. Toutefois le cœur de l'algorithme de résolution de contraintes est basé sur l'inversion de matrices de tailles moyennes suivant des méthodes évoluées. Plusieurs outils dans ce domaines existent (LINPACK, EISPACK), dont certains font partie du domaine public. Pour des raisons d'efficacité, de stabilité et d'expériences précédentes, le choix s'est porté sur une bibliothèque commerciale: Lapack.h++.

"LAPACK.h++ lets you translate your linear algebra problems from mathematics to C++ code through an intuitive interface that's efficient and easy to use."

Lapack.h++ est une bibliothèque orientée objet, distribuée par Rogue Wave software [Rogue Wave 96b] [Rogue Wave 96a] [wwwROG], qui fournit un ensemble de fonctionnalités pour traiter les matrices spéciales (triangulaire, par bande, etc.), les problèmes sous-déterminés et sur-déterminés d'équations linéaires ainsi que les décompositions en valeurs singulières (*SVD*). Les algorithmes utilisés par Lapack.h++ sont repris de la bibliothèque LAPACK, écrite en Fortran, appartenant au domaine public. Ces algorithmes sont donc largement utilisés et testés par la communauté scientifique. En fait, chaque objet Lapack.h++ encapsule une partie de la

bibliothèque LAPACK, ce qui procure à l'utilisateur une interface intuitive et facile à mettre en oeuvre. Cette utilisation à un plus au niveau d'abstraction permet de s'affranchir des problèmes d'implémentation et de gestion du stockage. D'autre part, cette encapsulation permet à Rogue Wave de porter ses bibliothèques sur de nombreuses plateformes et ainsi de fournir à l'utilisateur une solution cross-plateforme efficace et complètement transparente.

Il faut encore noter que Lapack.h++ est construite au-dessus de Math.h++ qui constitue la bibliothèque de base pour le traitement mathématique de Rogue Wave. Cette bibliothèque propose un ensemble de classes génériques pour les vecteurs et matrices, ainsi que les transformations de base statistique et de Fourier.

L'utilisation des bibliothèques Math.h++ et Lapack.h++ pour le projet «CINEGEN» assure des routines de traitement des matrices optimales et fiables, tout en permettant une intégration facile et élégante avec le code C++ de *VirtualRobot*.

## 5.4.5 Autres outils

En plus des bibliothèques fondamentales qui ont été passées en revue, *VirtualRobot* utilise aussi quelques outils qui n'ont pas un impact majeur sur la structure du programme, mais qui valent la peine d'être cités.

### 5.4.5.1 Manipulation de donnée: LEDA

Tout programme informatique organise les données en ensembles structurés afin d'y accéder suivant différentes méthodes et de les manipuler. Par exemple, les listes d'objets sont un des moyens de stockage souvent utilisés. Il est alors nécessaire de disposer de méthodes pour gérer les éléments de la liste. Ce type de besoin est si courant en programmation, qu'une bibliothèque "standard" a été développée autour de C++ pour répondre à ce problème: STL pour "Standard Template Library". Cette bibliothèque regroupe une collection d'objets de stockage ainsi que des méthodes unifiées d'accès aux éléments. STL devrait être incluse dans un environnement de programmation C++ complet et, de par sa conformité à la norme "ANSI-C++", être portable entre différentes plateformes. Toutefois la version de la norme ANSI pour les STL ne sera finalisée qu'à la fin 1998 ce qui limite encore nettement le développement de bibliothèques STL par les fournisseurs de logiciels. Par exemple, Silicon Graphics ne proposait pas encore une STL satisfaisante au début du projet. D'autre part l'aspect totalement générique d'une STL la rend d'usage ardue pour des types d'objets plus compliqués comme les graphes. C'est pourquoi *VirtualRobot* n'utilise pas STL mais se base sur une autre bibliothèque pour la manipulation de donné: LEDA (Library of Efficient Data Types and Algorithms) [Mehlhorn 98] [wwwLED].

Alors que de nombreuses bibliothèques existent pour le traitement statistique ou l'analyse numérique, rien n'est proposé pour le domaine de la géométrie et de la combinatoire informatique: le domaine qui traite des objets tels que graphes, séquences, dictionnaires, arbres, plus court chemin, flux,

lignes, segments ou surfaces convexes. LEDA répond à ce besoin et fournit de nombreuses classes d'objets génériques permettant de manipuler ces types de données. LEDA est implémenté en C++ avec une documentation de qualité et est distribué gratuitement pour les institutions de recherche.

"LEDA is a library of efficient data types and algorithms in combinatorial and geometric computing." [wwwLED]

*VirtualRobot* n'utilise qu'une toute petite partie des potentialités de LEDA. Cependant l'utilisation de cette bibliothèque évite de re-implémenter de nombreux types de base, et de bénéficier d'une bibliothèque reconnue comme efficace et largement testée. Les principaux atouts de LEDA dans le contexte de *VirtualRobot* sont les suivants:

- réalisée en C++, tous ses types et algorithmes sont précompilés d'où une réduction du temps de compilation,
- propose beaucoup de types de données paramétrisés,
- son type de données "graphe" est pratique d'utilisation et possède de nombreuses méthodes d'accès aux objets et d'itérations comme: "pour tous les noeuds v du graphe G fait {...}".

#### 5.4.5.2 Documentation: Cocoon

"Cocoon organizes the information provided in C or C++ header files into a liberally cross-linked hypertext documents that hopefully affords all the most important navigational aids". [wwwCOC]

Un problème majeur lors de la mise en oeuvre d'un projet informatique est la réalisation de la documentation. Le programme lui-même doit être généreusement commenté si on veut avoir une chance de le réutiliser par la suite. De même une documentation technique (manuel de référence) est nécessaire pour la compréhension du programme et pour son extension par de tierces personnes (nous ne parlerons pas ici du manuel d'utilisation qui peut se réaliser de manière plus conventionnelle). Pour de gros projets des outils considérables sont indispensables tel ROOT [wwwROO] développé par le CERN pour la maintenance de ses propres projets. La taille de «CINEGEN» ne justifie pas l'utilisation d'un tel outil. Par contre, un utilitaire relativement simple, Cocoon [wwwCOC], permet d'aider à réaliser la documentation de *VirtualRobot*.

Cocoon traite tous les fichiers en-têtes d'un projet réalisé en C++ afin de créer un réseau de pages au format HTML qui documentent les modules, les classes et les fonctions globales. Quelques conventions de mise en page dans les fichiers d'en-tête permettent à Cocoon d'en extraire les commentaires qui vont servir à créer la documentation. Une fois ces en-têtes traités, Cocoon fournit un ensemble de pages hypertext contenant de nombreux liens entre les mots clefs ainsi qu'une organisation hiérarchique et un index global. Chaque classe rencontrée possède une introduction, puis chaque méthode est documentée avec ses arguments. Les liens sont aussi créés en fonction de l'héritage des différentes classes. Evidement Cocoon ne fournit que l'information que le programmeur a bien voulu écrire dans son code, mais il organise cette information de manière efficace. D'autre part, le fait de documenter les classes directement dans les fichiers d'en-tête permet au programmeur d'insérer des commentaires sans changer de fichier ou d'application, et d'apporter les corrections nécessaires directement lorsque certaines méthodes sont modifiées.

### 5.4.6 Conclusion

*VirtualRobot* utilise diverses bibliothèques provenant du domaine public ou du commerce. La maîtrise de ces différents outils garantit une fiabilité des fonctionnalités de base. Le Tableau 5-1 résume l'usage de ces outils. Afin de garantir une éventuelle portabilité du programme sur d'autres plateformes, les bibliothèques ont été choisies en conséquence.

**Tableau 5-1:** *Bibliothèques externes utilisées pour VirtualRobot*

| Bibliothèque | Usage | Provenance | Portabilité |
|---|---|---|---|
| WorldToolKit | Environnement Virtuel | Sense8 | Unix, Windows NT |
| Tcl/Tk + TIX | interface utilisateur | domaine public | Unix, Windows† |
| ANTLR | générateur d'analyseur | domaine public | Unix, Windows |
| Lapack.h++ | traitement matriciel | RogueWave | Unix, Windows |
| LEDA | gestion des données | domaine public | Unix, Windows |
| Cocoon | Documentation | domaine public | Unix, Windows |

†. La portabilité Windows inclut normalement Windows 95 et Windows NT.

## 5.5 Implémentation des modules

La Section 5.3 détermine quels sont les différents modules de *VirtualRobot* et la Section 5.4 présente les bibliothèques utilisées. Cette section décrit comment sont implémentés les modules composant *VirtualRobot*, c'est-à-dire la réalisation de l'outil proposé dans ce travail. Nous n'allons pas expliquer ici les détails du code nécessaire à cette programmation (le lecteur se rapportera pour cela à l'Annexe D, "Documentation de *VirtualRobot*"), mais nous décrivons les algorithmes utilisés pour les parties principales du programme.

### 5.5.1 Construction de la structure de données

La majorité des données utilisées par le programme *VirtualRobot* pour créer un environnement virtuel contenant les robots provient de fichiers de description. Le fichier MAD décrit la structure des robots et leurs interactions avec les senseurs. Des fichiers contenant les modèles graphiques d'objets permettent la représentation visuelle des objets. *VirtualRobot* analyse ces fichiers afin créer la structure de données interne du programme.

#### 5.5.1.1 Analyse du fichier MAD: "parser"

L'analyse du fichier MAD est essentiellement prise en charge par PCCTS. Les données devant être définies par le développeur pour la génération automatique d'un parser de fichier MAD sont de deux types.

- La syntaxe du fichier *VirtualRobot*. Cela correspond à la définition des tokens (les mots clefs à syntaxe fixe ainsi que les types de variables) pouvant être rencontrés dans un fichier MAD. La syntaxe des variables

est décrite à l'aide de l'assemblage voulu de caractères génériques. Les variables rencontrées dans un fichier MAD peuvent être de type chaîne de caractères, nombre entier ou nombre réel.

- La grammaire du fichier MAD. Chaque objet du programme *VirtualRobot* à identifier est représenté par une règle grammaticale nommée expression. Toute expression est découpée en sous-expressions, ceci jusqu'au niveau du token élémentaire. La réussite de l'unification d'une expression est associée à une action de création de l'objet correspondant. Le principe d'appel des actions est pratiquement identique pour toutes les expressions rencontrées dans un fichier MAD. La Figure 5-6 montre ce schéma générique d'analyse d'une expression.

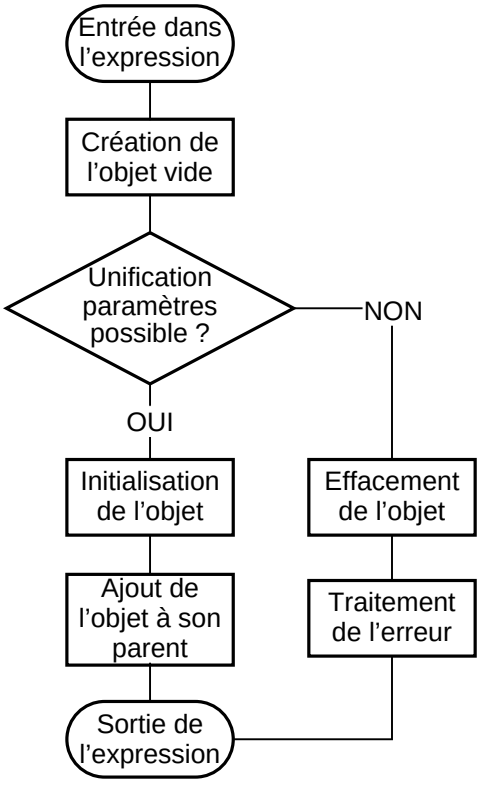



**Figure 5-6:** *Actions relatives à l'analyse d'une expression d'un fichier MAD*

L'analyse top-down effectuée par PCCTS permet la création des objets du monde en allant du plus général au plus petit sous-objet. Ainsi le programme commence par créer l'objet `monde` qui rassemblera tous les autres objets. Puis chaque fois que la description d'un robot est identifiée, une instance de `robot` est créée, initialisée avec les bons paramètres, puis ajoutée à l'objet global `monde`. De même chaque nouveau composant d'un robot rencontré est ajouté au graphe du robot auquel il appartient. PCCTS autorise les passages de paramètres entre les expressions et sous-expressions, ce qui permet à un sous-objet de connaître son "parent". Dans le cas de l'analyse d'un robot R, si le parser identifie un `lien`, l'objet `robot` sera passé comme argument à la règle qui va analyser le lien. A ce moment, le lien pourra stocker un pointeur sur ce robot R qui est son "parent". Il faut donc créer

l'objet correspondant à l'expression traitée dès l'entrée dans la procédure d'unification de cette expression. En effet, une instance de cet objet sera nécessaire pour pouvoir lui attacher les sous-objets. Si ensuite une quelconque règle échoue, il faudra détruire cet objet qui n'est pas valide. Ceci oblige aussi à disposer de fonctions pour initialiser tous les paramètres nécessaires après la création.

Ces fonctions seront aussi utiles lors de modifications dynamiques éventuelles qui ont lieu si des événements externes le demandent, par exemple lorsqu'un ascenseur de l'interface graphique modifie la position initiale de la base d'un robot.

#### 5.5.1.2 Construction dynamique des objets: "builder"

Chaque robot est évidemment construit dynamiquement lors de l'exécution, puisque le programme ne connaît rien du monde avant de lire un fichier MAD. Cette étape du programme n'est pas un module proprement dit, mais un ensemble de routines attachées aux différents objets. Chaque objet – `base`, `lien` ou `référentiel` – contient donc les fonctionnalités qui vont permettre de construire pièce par pièce la structure d'un robot.

Lorsqu'il rencontre un fichier MAD qui débute par les règles correctes, le programme commence par construire un objet `monde`. L'objet `monde` permet de référencer tous les autres objets qui appartiennent à une simulation. Cet objet hérite d'un autre objet `scène`, comme le montre la Figure 5-7, qui prend en charge la partie environnement virtuel avec toutes ses initialisations et la gestion des fenêtres. Ensuite les objets `robot` ou `senseur` vont être enregistrés auprès du `monde`. Ceci est possible grâce au passage de paramètres effectués entre les règles du parser. Ainsi le `monde` contient deux listes principales qui référencent tous les senseurs et robots créés dynamiquement.

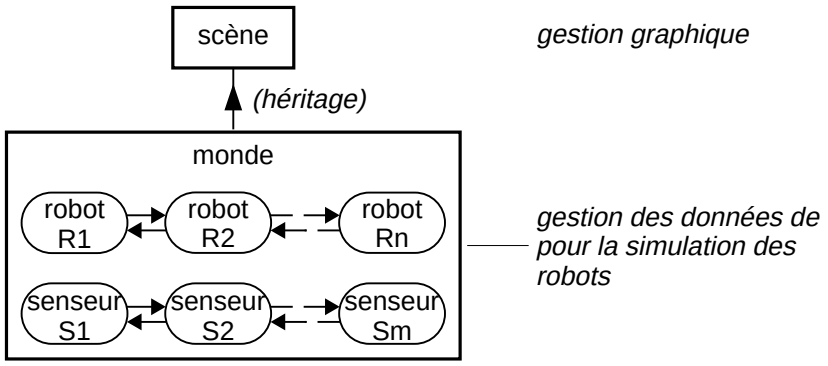



**Figure 5-7:** *Composition des objets formant une scène*

Lorsqu'un nouveau `robot` est créé, il est simplement initialisé avec quelques valeurs et il contient alors un graphe vide. Puis, au fur et à mesure que ses différents composants sont identifiés à la lecture du fichier MAD, ils sont insérés dans le graphe du robot (géré par l'objet `graphe`).

Cette structure entièrement top-down serait suffisante pour une application classique, puisque tout objet peut être atteint si l'on part de l'objet de base: le `monde`. Toutefois ces relations sont insuffisantes pour une

application incluant de la réalité virtuelle: l'utilisateur peut vouloir accéder directement à un objet tout en bas de la hiérarchie. Ce cas arrive par exemple si l'utilisateur sélectionne un objet graphique. Cet objet graphique devra être alors capable de déterminer quel est son parent hiérarchique. Ceci est possible grâce aux références que contient chaque objet sur son parent.

La Figure 5-8 montre les relations entre les objets composant un robot. Tout objet constituant d'un `robot` – à savoir l'objet `base` et les objets `lien`, `référentiel` et `senseur` – hérite d'un même objet générique `noeud`. Ainsi l'objet `graphe` organisant la structure d'un robot est composé de `noeud`s génériques. Ceci permet aussi de disposer de méthodes générales pour la manipulation des objets (positionnement, demande d'information, affichage, etc.)

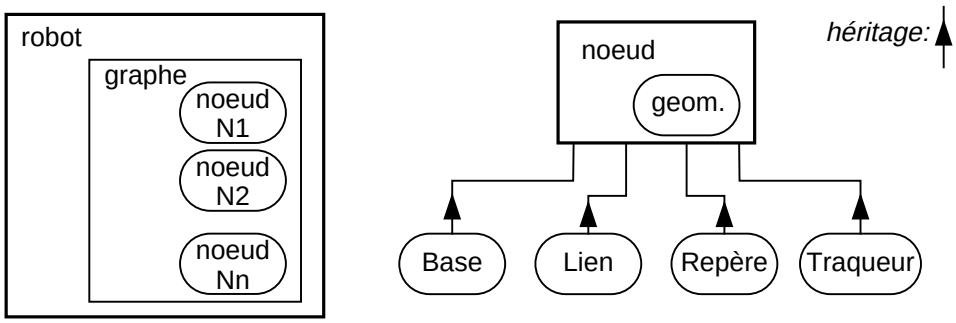



**Figure 5-8:** *Composition des objets formant un robot*

## 5.5.2 Construction du graphe de scène

Afin de définir un environnement virtuel qui permette de visualiser la scène par représentation tridimensionnelle d'objets graphiques, il faut créer le graphe de scène et gérer tous les appels graphiques de manière adéquate.

### 5.5.2.1 Classes utilisées

La construction de la scène graphique est étroitement liée à la bibliothèque graphique utilisée. Toutefois, afin de garantir une indépendance maximale vis-à-vis de cette bibliothèque, *VirtualRobot* regroupe tous les appels graphiques émanant des objets composant un robot dans une seule classe: `Geometries`. Ainsi chaque objet qui a besoin de propriétés graphiques contiendra un objet `geometrie`.

D'autre part, toutes les fonctionnalités connexes à la scène graphique, comme le fenêtrage ou les entrées sorties, sont aussi définies dans des classes séparées. Celles-ci sont brièvement décrites ci-dessous.

- `Scènes`: cette classe hérite[8] de la classe WtUniverse [Sense8 98] qui définit la classe mère d'une scène graphique. Cette classe s'occupe de toutes les initialisations d'une scène graphique, y compris l'ouverture d'une fenêtre principale. De plus cette classe transpose les fonctionnalités pour lancer ou arrêter une simulation ("main loop" de

8. Au sens orienté-objet du C++: héritage des caractéristiques (attributs et méthodes) de la classe mère.

l'application graphique). La classe `Mondes` (section précédente et Figure 5-7) hérite elle même de cette classe de base.

- `Windows`: cette classe hérite de la classe WtWindow et permet à *VirtualRobot* de faire tous les appels relatifs à la gestion de fenêtres à travers une classe générique.
- `Senseurs`: cette classe hérite de la classe WtSensor et permet la création, l'initialisation, la configuration, la lecture et la destruction des objets `senseurs`. En regroupant des appels propres à WorldToolKit, cette classe simplifie grandement la gestion des senseurs. D'autres classes spécifiques (comme "Mouse" ou "SpaceBall") vont hériter de cette classe et spécialiser un `senseur` tout en garantissant au reste du programme une même interface pour n'importe quel senseur.

#### 5.5.2.2 Structure et construction du graphe de scène: "loader"

Lors d'une simulation effectuée par *VirtualRobot*, les éléments principaux composant la scène graphique sont les robots que le programme simule. Le positionnement de tous les constituants des robots est entièrement pris en charge par *VirtualRobot* (ils sont organisés par les graphes des robots et non le graphe de scène comme expliqué à la Section 5.3.2). Chaque position et orientation des objets composant un robot sont mises à jour indépendamment par le programme, en fonction de la résolution de toutes les contraintes, à chaque nouvelle boucle de simulation. Ceci implique un graphe de scène essentiellement horizontal regroupant tous les objets graphiques au même niveau, chacun possédant son propre noeud de transformation[9].

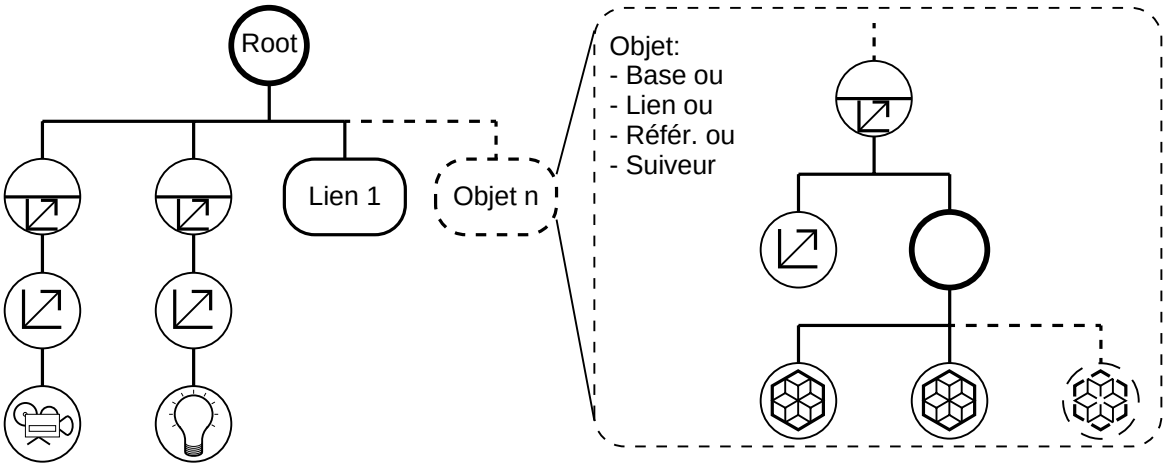



**Figure 5-9:** *Composition du graphe de scène de VirtualRobot*

Comme le montre la Figure 5-8, la classe `Noeuds` est composée entre autre d'un objet de la classe `Geometries`. Chaque objet `geometrie` va contenir des éléments du graphe de scène. C'est donc l'objet `geometrie` qui

9. Cette situation contraste avec un environnement virtuel classique où le moteur graphique met à jour les positions des objets de la scène en fonction des modifications des noeuds de transformation du graphe de scène.

permet la représentation graphique des objets composant un robot dans l'environnement virtuel. Ainsi chaque fois qu'un nouveau noeud de robot est créé, plusieurs noeuds graphiques sont insérés dans le graphe de scène.

La Figure 5-9 montre cette organisation. Un séparateur[10] est d'abord créé pour isoler les géométries représentant l'objet courant des autres transformations. Puis un noeud de positionnement est inséré sous ce séparateur. Ensuite un noeud de groupement va permettre d'associer tous les objets graphiques qui décrivent l'objet courant (le fichier MAD permet d'associer plusieurs fichiers graphiques à un composant de robot). Tous les objets graphiques sont supposés avoir leur origine positionnée sur l'origine du repère qui référence l'objet du robot par rapport à son parent (cf Section 4.1.2.5).

### 5.5.3 Boucle principale: "main loop"

Dans cette partie du programme sont concentrés tous les appels aux méthodes des différents objets afin de réaliser la résolution des contraintes et la mise à jour de l'environnement virtuel. Ces fonctions sont critiques au niveau temps de calcul car la fluidité de la simulation va dépendre de leur rapidité d'exécution.

La Figure 5-10 montre les principales fonctions appelées à chaque boucle de simulation. Une première phase consiste à analyser tous les événements graphiques qui influencent le programme. Ces événements peuvent être générés par l'interface graphique (boutons, ascenseurs, etc.) et/ou la souris et le clavier indépendamment de l'interface graphique. De la même manière, il est nécessaire de lire l'état de tous les senseurs enregistrés pour cette simulation. La résolution des contraintes de chaque robot et finalement la mise à jour des éléments de la scène en fonction de tous ces événements terminent cette boucle.

```
MainLoop:
  Interface_Events
  Mouse_Events
  Key_Events
  for all sensor in SensorsList
    sensor.Read
  end loop
  for all robot in RobotsList
    if (activeSensors > 0) then
      robot.InverseKinematics
      robot.Position
    end if
  end loop
```

```
InverseKinematics (robot):
begin
  createCartesianSpeeds
  createJacobian
  inverseJacobian
  calcJointSpeeds
  updateJoints
  calcNewErrors
end
```

**Figure 5-10:** *Algorithme de la boucle principale (pseudo-code)*

10. Ce séparateur est en fait un séparateur de transformation. Il isole ses fils des autres transformations, mais leur laisse subir d'autres noeuds comme les lumières par exemple.

### 5.5.3.1 Lecture des senseurs

Dans un environnement Unix, les événements graphiques sont pris en charge par X windows.

La lecture des senseurs se différencie de celle des autres événements clavier et souris, car elle n'est pas prise en charge par le gestionnaire d'interface graphique. Il s'agit donc de développer les routines nécessaires pour lire les valeurs retournées par chaque senseur. Cette tâche doit être adaptée aux senseurs utilisés et au type de connexion. La plupart du temps les senseurs sont branchés à l'ordinateur par un port série (RS232), mais l'information peut aussi provenir d'une autre machine par le réseau, ou éventuellement par un autre processus. D'ou l'intérêt de la classe `Senseurs` qui permet de rendre ce problème transparent du point de vue des robots. Quelle que soit la nature de l'information que l'on désire comme consigne pour manipuler le robot, elle est gérée de manière identique au niveau de la classe `Robot`.

L'utilisation de la bibliothèque graphique WorldToolKit simplifie grandement la gestion des senseurs car les routines de gestion de nombreux périphériques commerciaux sont fournies. Il suffit dans ces cas de composer avec les routines de base pour créer les fonctions génériques de la classe `Senseurs`.

Etant donné que la résolution des contraintes s'effectue dans l'espace des vitesses, il faut convertir les données provenant des senseurs en incréments à appliquer au robot. Dans la version actuelle la résolution des contraintes se fait uniquement en fonction de l'incrément courant, donc un senseur de type absolu sera aussi utilisé comme un capteur incrémental.

### 5.5.3.2 Construction de la matrice jacobienne augmentée

La matrice jacobienne étant une représentation instantanée de l'état du robot, elle doit donc être recalculée après chaque modification des valeurs des articulations du robot. Dans le cas de *VirtualRobot*, la matrice jacobienne augmentée sera donc calculée à chaque boucle puisque les postures des robots sont mises à jour avant chaque nouvel affichage.

La matrice jacobienne augmentée $J_a$ est définie dans la Section 3.4.2. Le nombre de colonnes de $J_a$ est égal au nombre de degrés de liberté du robot, et le nombre de lignes de $J_a$ est égal au nombre de contraintes (nombre de boucles + nombre de senseurs actifs) multiplié par six (nombre de degrés de liberté de l'espace cartésien tridimensionnel). Il s'agit de "remplir" chaque case de cette matrice en fonction des postures courantes de chaque lien. Ceci est réalisé en parcourant le graphe du robot et en convertissant les matrices de transformation de chaque lien en valeurs d'influence pour la matrice jacobienne.

#### *Analyse des contraintes*

Avant de construire la matrice jacobienne augmentée du robot, le programme analyse toutes les contraintes actives dans ce robot. Pour cela il traverse le graphe du robot et identifie tous les noeuds (`lien` ou `référentiel`) possédant une contrainte (soit fermeture d'une boucle, soit suivi d'un senseur). Cette identification permet de construire deux listes stockant des pointeurs sur tous les noeuds concernés. La première liste contient les pointeurs sur les `lien` ou `référentiel` (n° 9 dans l'exemple) juste avant

un `traqueur` référençant un `senseur` actif[11]. La deuxième liste contient une structure de 3 noeuds correspondant chaque fois aux deux noeuds terminaux reliés par une contrainte de fermeture de boucle (n° 7 et n° 8 dans l'exemple), plus le noeud dont sont issus ces deux sous-branches (n° 1).

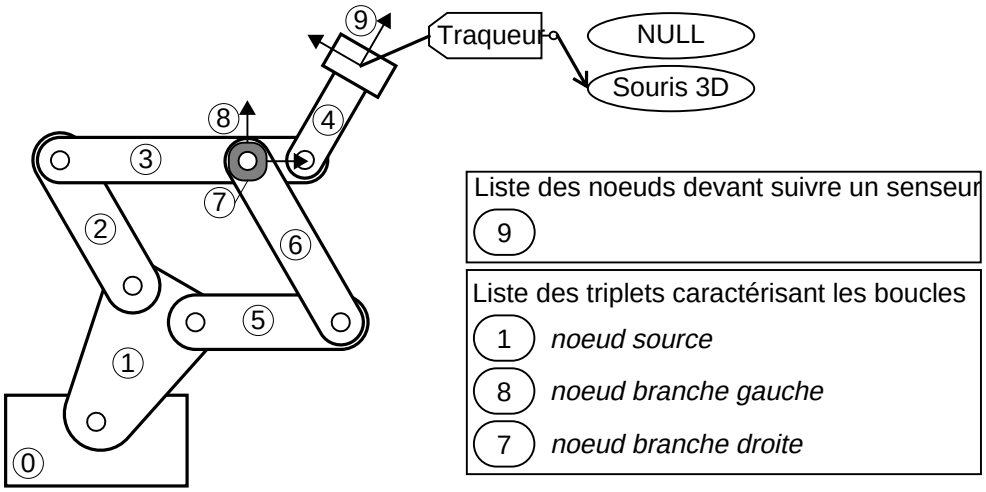



**Figure 5-11:** *Exemple de contraintes actives d'une structure*

Cette redondance dans le stockage des contraintes dans des listes supplémentaires (elles apparaissent déjà dans le graphe) permet d'éviter de traverser le graphe complet à chaque nouvelle construction du jacobien. Il suffit d'appeler la méthode d'analyse des contraintes après chaque modification dynamique de la structure du robot, ou des senseurs.

### *Remplissage de la matrice jacobienne augmentée*

L'algorithme de construction de la matrice jacobienne augmentée se fait en deux phases: d'abord il construit le sous-jacobien pour chaque branche qui se termine par un senseur actif, puis il construit les sous-jacobiens correspondant aux boucles cinématiques.

La première phase utilise la liste des noeuds ayant un senseur actif et se base sur la relation de récurrence (3.19). Considérons une branche dont le dernier noeud possède la matrice de transformation ${}^{n}T_{E}$. L'algorithme remonte la branche pour calculer successivement les matrices ${}^{k}T_{E}$ représentant l'influence de l'articulation k sur le dernier noeud E:

$${}^{k}T_{E} = {}^{k}T_{k+1} \cdot \ldots \cdot {}^{n-1}T_{n} \cdot {}^{n}T_{E}$$

Chaque ${}^{k}T_{E}$ calculé en fonction de la matrice de transformation locale de l'articulation et de ${}^{k+1}T_{E}$ provenant de l'étape précédente, est utilisé pour construire 6 éléments (en colonne) du jacobien (cf. Section 3.4.2).

Dans la deuxième phase, l'algorithme est le même que précédemment, mais est effectué pour chaque sous-branche formant une boucle. Le processus commence donc en fonction des noeuds terminaux stockés dans la liste des

11. Un traqueur peut pointer sur une liste de senseurs potentiels. Si un senseur est référencé par un traqueur, il est considéré comme "actif". Le senseur nul (utilisé lorsque l'on ne désire pas de suivi de consigne senseur) n'est jamais actif.

triplets caractérisant une boucle, et s'arrête au moment où le noeud commun aux sous branches traitées est atteint. Donc chaque sous-branche est traitée successivement et contribue à former la sous-matrice jacobienne. La seule spécificité étant que les influences de la branche droite (arbitraire) sont inversées avant d'être insérées dans la matrice jacobienne augmentée suivant la relation (3.26) décrite dans la Section 3.4.2.2. Il faut en effet que l'influence de la branche droite compense exactement l'influence de la branche gauche afin d'assurer la fermeture de la boucle cinématique.

#### 5.5.3.3 Résolution des contraintes

L'inversion de la matrice jacobienne augmentée définie dans la Section 3.4.2 est la clef de la résolution des contraintes d'un robot. Cette inversion définie dans la Section 3.4.3 conditionne la validité des solutions obtenues. L'utilisation d'une décomposition en valeurs singulières (SVD) comme le montre la Section 3.3.3 permet l'obtention de solutions dans le cas général avec une approximation au sens des moindres carrés.

La décomposition en valeurs singulières de la matrice jacobienne augmentée est effectuée à l'aide de la bibliothèque mathématique Lapack.h++. Les fonctionnalités de cette bibliothèque permettent de contrôler précisément comment s'effectue cette décomposition.

Les étapes suivantes sont répétées pour chaque robot de la scène afin de résoudre toutes les contraintes.

1) La matrice jacobienne augmentée $J_a$ est construite suivant la méthode de la Section 5.5.3.2.
2) $J_a$ est décomposée en trois matrices suivant une SVD: $J_a = U \cdot \Sigma \cdot V^T$.
3) On calcule la pseudo-inverse $J_a^+$ de $J_a$: $J_a^+ = V \cdot \Sigma^+ \cdot U^T$, où $\Sigma+$ est définie dans la section Section 3.3.3.
4) On obtient les variations articulaires: $\dot{\mathbf{q}} = J_a^+ \cdot \dot{\mathbf{x}}_a$, où $\dot{\mathbf{x}}_a$ est le vecteur augmenté des déplacements cartésiens défini dans la Section 3.4.3.

Si une singularité dans la matrice jacobienne apparaît, elle va perdre au moins un rang, et des valeurs proches de 0 vont surgir dans les éléments de $\Sigma$. Etant donné que la méthode actuelle est basée sur une approximation par les moindres carrés, il suffit donc d'inverser chaque élément de $\Sigma$. Lorsque $\Sigma$ comporte un élément proche de 0, alors on choisit 0 pour son inverse (cf. Section 3.3.3).

### 5.5.4 Intégration de la GUI

L'interface graphique utilisateur joue un rôle capital pour l'interactivité du programme, pour sa simplicité d'utilisation et son efficacité. Le choix de Tcl/Tk associé à Tix pour réaliser la GUI de *VirtualRobot* apporte une facilité de développement de nouveaux éléments d'interface, mais pose quelques problèmes quand à son fonctionnement avec le cœur du programme qui est basé sur une boucle de simulation. En effet, Tcl/Tk est conçu pour créer des

interfaces graphiques et réaliser la "glu" entre les différents modules d'un programme. Le déroulement de l'ensemble du programme est alors contrôlé par Tc/Tk. Par exemple une interface avec des boutons correspondant à différents modules d'un programme, vont lancer les modules adéquats lorsqu'on clique dessus. Ces modules peuvent êtres lancés en appelant le nom d'un programme complet (exécutable), ou appeler directement des procédures C ou C++. Tcl prend donc en charge tous les événements clavier, souris et autres dans une "boucle événementielle" qui tourne en continu, et appelle des commandes en conséquence: c'est le but premier d'un "Tool Command Language".

Dans le cas de *VirtualRobot*, le déroulement du programme est contrôlé par la boucle de simulation de l'environnement virtuel. Pour des raisons de performance et de régulation des priorités, il n'est pas envisageable de transférer ce contrôle à Tcl/Tk. C'est pourquoi il a fallu trouver un autre moyen pour intégrer Tcl/Tk dans la boucle de simulation. Deux approches ont été implémentées: une première utilisant des "*thread*s" (cf. Section E.1 de l'Annexe E) a finalement été rejetée au profit d'une solution plus simple qui satisfait toutefois les exigences du programme. Cette dernière méthode est décrite dans cette section.

### *Implémentation par gestion externe de la boucle Tcl*

La solution retenue pour l'intégration de Tcl/Tk dans *VirtualRobot* est d'éclater la boucle de gestion d'événements de Tcl et de la prendre en charge au sein de la boucle de simulation. Tcl fournit en effet les procédures permettant de contrôler manuellement la gestion des événements.

La solution adoptée, illustrée à la Figure 5-12, est de vider la queue des événements Tcl à chaque passage dans la boucle de simulation. Une lecture forcée des événements (la procédure analysant les événements retourne un code même si aucun événement n'est survenu) permet de ne pas bloquer la boucle principale de *VirtualRobot* (attente d'un événement potentiel). D'autre part le fait de vider toute la queue de ses événements garantit à Tcl de pouvoir traiter suffisamment d'événements à la suite pour présenter des réactions rapides. Les problèmes techniques liées à cette méthode sont détaillés dans la Section E.2. Cette méthode donne entière satisfaction aussi bien pour l'actualisation du monde virtuel que pour le comportement de l'interface utilisateur,

### *Intégration de l'interface Tix avec les objets de VirtualRobot*

Une fois la gestion des événements de l'interface graphique intégrée dans la boucle de simulation de l'environnement virtuel, il reste à définir une méthode efficace pour créer les éléments de l'interface graphique (widgets) qui leur permettent de communiquer avec les autres objets. Etant donné que *VirtualRobot* est conçu suivant une méthode orientée-objet, il est logique de fournir à n'importe quel objet les fonctionnalités pour créer sa propre interface. Par exemple, chaque `lien` du robot peut avoir un panneau d'interface de contrôle associé.

L'utilisation de Tix comme couche au-dessus de Tcl/Tk permet une approche respectant l'esprit Tcl/Tk pour la création et la gestion d'interface

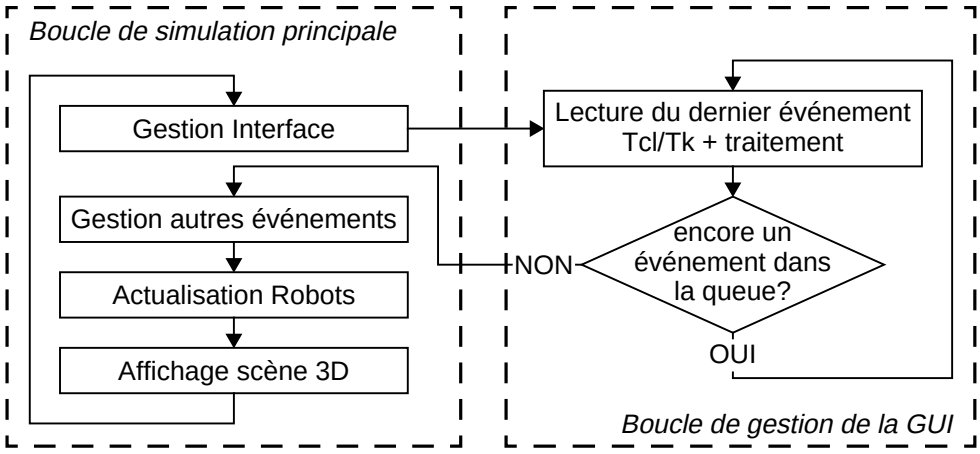



**Figure 5-12:** *Insertion de la gestion Tcl/Tk dans la boucle de simulation*

tout en renforçant l'aspect objet de l'interfaçage avec *VirtualRobot*: chaque objet C++ pourra hériter d'un mega-widget. Ce mega-widget est décrit en Tcl/Tk-Tix et contient tous les widgets nécessaires à la formation d'un panneau d'interface pour l'objet C++[12]. Cette technique est mise en oeuvre par l'intermédiaire de deux nouvelles classes créées dans le cadre de ce projet:

- Tix-Interprète est une classe dont une seule instance sera réalisée. Elle se charge de l'ouverture d'un interpréteur Tcl/Tk-Tix et de ses canaux de liaison avec C++. Par un appel à une méthode globale de cette classe, n'importe quel autre objet peut retrouver le pointeur sur l'interpréteur Tix et ainsi lui envoyer des commandes.
- Tix-Interface est une classe dont vont hériter tous les objets de *VirtualRobot* qui veulent posséder une interface graphique Tix. Cette classe contient les méthodes nécessaires à la création d'un mega-widget, à l'assignation de paramètres, à la liaison automatique de variables entre C++ et Tcl. Cette classe effectue aussi la gestion des *call-back* avec le passage de paramètres.

La Figure 5-13 montre le schéma d'utilisation de ces deux classes pour interfacer Tcl/Tk-Tix avec un programme C++, tout en conservant un concept orienté-objet. En plus d'une intégration parfaite dans le programme C++, cette approche permet de découpler complètement deux travaux: a) la création de l'interface; b) la programmation de la simulation. En effet, on peut créer, étendre et même tester le mega-widget Tix indépendamment du programme C++ qui va en hériter. Comme Tcl/Tk-Tix est un langage interprété, aucune compilation n'est nécessaire si l'on apporte une modification au mega-widget. Ceci réduit fortement le temps de développement de l'interface graphique (qui nécessite constamment de petits ajustements).

12. Une autre solution serait de créer une série d'objets C++ qui vont chaque fois encapsuler un widget Tk. Chacun de ces objets disposerait des méthodes adéquates pour sa création, destruction, changement de paramètres, affichage, insertion dans un autre objet. Cette approche reviendrait simplement à disposer d'une interface C++ au lieu de Tcl pour créer ce widget, ce qui n'est pas forcément une simplification pour la mise en oeuvre!

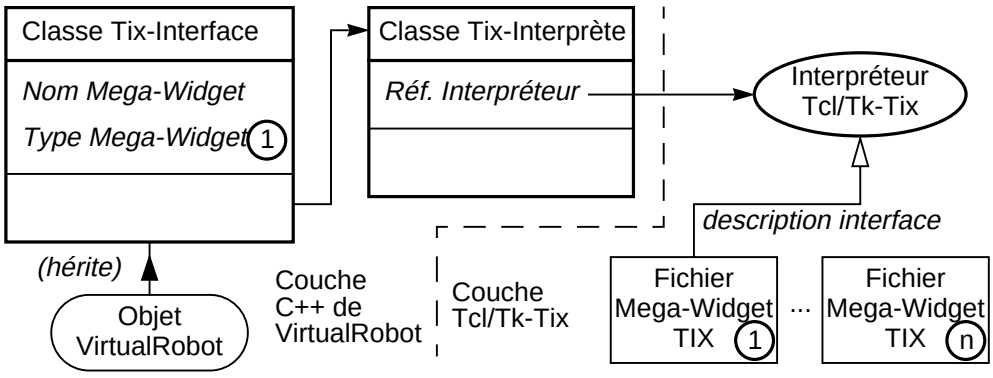



**Figure 5-13:** *Interfaçage de Tcl/Tk-Tix avec VirtualRobot*

## 5.6 Résumé

Ce chapitre a montré comment est mis en oeuvre de manière informatique le projet «CINEGEN» pour aboutir au programme *VirtualRobot* qui permet la simulation en temps réel de la cinématique de robots manipulateurs à structure quelconque. Il a décrit successivement les points suivants.

- La plateforme informatique nécessaire à cette réalisation; elle dépend essentiellement du temps réel à garantir dans une application basée réalité virtuelle.
- L'architecture théorique du programme et l'organisation des structures de données.
- Les outils et bibliothèques sur lesquels se base *VirtualRobot* pour gagner en efficacité et fiabilité.
- L'implémentation des algorithmes nécessaires à la construction des robots et à la résolution des contraintes.
- L'intégration d'une GUI basée sur Tcl/Tk-Tix dans l'architecture orientée-objet de *VirtualRobot*.

Le chapitre suivant décrira en détails les possibilités qu'offre *VirtualRobot* et analysera les résultats obtenus lors de simulations de structures de types variés.

CHAPITRE

# 6

# FONCTIONNALITÉS ET ÉTUDE DE CAS

Une interface utilisateur nécessite un échange bidirectionnel d'informations. Les méthodes essentielles pour fournir des consignes à la machine reposent sur le geste et la voix. Le geste doit être compris dans un sens général: cela peut être un mouvement (agissant directement sur un système mécanique ou bien analysé par un système sans contact) de la main, d'un doigt ou de toute autre partie du corps, voire une pression sur un objet. L'utilisation de la voix est intéressante dans les cas où l'on peut définir des actions avec une sémantique simple ("allume le projecteur" par exemple), mais reste limitée dans son emploi par sa faible bande passante (la définition d'une courbe tridimensionnelle risque d'être fastidieuse vocalement).

Les retours d'information peuvent s'adresser à tous les systèmes sensoriels de l'homme. Toutefois l'utilisation la plus fréquente concerne la vision, le toucher (tactile et force) et l'ouïe[1].

La vision est sans aucun doute le moyen le plus utilisé dans tous les systèmes de simulation et de réalité virtuelle pour procurer des informations à l'utilisateur. Cela provient de l'état de la technologie qui est avancée dans ce domaine (écran de qualité, simulations 3D temps réel, système de vision stéréoscopique, etc.), mais aussi de la quantité énorme d'informations que le cerveau humain arrive à extraire et à traiter grâce à son système de vision. De plus, l'homme moderne est habitué à gérer de l'information provenant d'écrans d'ordinateurs. Ceci facilite son apprentissage face à des systèmes de réalité virtuelle qui ne ressemblent pas à la réalité (soit par manque de performance, soit par volonté délibérée d'épurer l'information).

1. Le goût et l'odorat sont des sens plus difficiles à exciter artificiellement. Le goût n'est pas très utile dans une simulation de robots car l'homme adulte n'a pas l'habitude de porter à sa bouche des pièces mécaniques. L'odorat dans la réalité sert souvent de signal d'alarme (odeur d'un moteur qui chauffe trop) mais peut être remplacé dans l'environnement virtuel par d'autres type d'informations (voyant rouge d'alarme). Il existe toutefois des systèmes d'éveil par odeur de brûlé.

Les systèmes à retour d'efforts sont encore peu développés à cause de la complexité de mise en oeuvre de tels mécanismes, mais ils semblent promis à un avenir brillant. En effet l'homme effectue un très grand nombre de tâches à l'aide de ses mains (outil terminal du bras et du reste du corps), et si la vision joue un rôle important dans la coordination de ses mouvements, la sensation de toucher et de force est aussi fondamentale. Afin de rendre l'homme performant pour une tâche effectuée dans un environnement virtuel, il semble donc naturel de lui fournir le même type de sensation d'efforts.

Les stimulations de l'ouïe commencent aussi à être utilisées grâce aux progrès effectués dans la création d'environnements sonores tridimensionnels artificiels. A part l'augmentation de la sensation de réalité grâce au bruitage, l'utilisation de sons dans un environnement virtuel permet d'aider à la localisation d'événements (qui ne sont pas forcément discernables sur la représentation graphique en fonction du point de vue à cet instant).

## 6.1 Définition de consignes

Afin d'offrir un maximum de flexibilité et une adaptation optimum à la tâche, le programme *VirtualRobot* propose des moyens variés pour saisir des données. L'interface basée sur les environnements virtuels privilégie les périphériques de saisie utilisés dans ce type d'application (Magellan, souris à ultrasons et autres périphériques du commerce). Toutefois, il est aussi possible comme annoncé dans la Section 5.3.3 d'utiliser une interface conventionnelle 2D. D'autre part, un périphérique dédié à la robotique, incluant un retour d'efforts, a été développé durant ce travail de thèse.

### 6.1.1 Périphériques 3D du marché

L'intérêt premier d'une interface basée sur la réalité virtuelle pour étudier et piloter des robots est l'utilisation de périphériques de saisie et de visualisation permettant d'appréhender l'espace tridimensionnel de manière intuitive.

Les périphériques de saisie (senseurs) sont utilisés dans *VirtualRobot* pour contrôler un `lien` ou un `référentiel` de la structure d'un robot. Ils permettent de générer une consigne que va devoir suivre l'objet concerné. La consigne est générée dans l'espace cartésien de la tâche et peut-être exprimée dans différents repères. Le pilotage de l'organe terminal d'une structure articulée consiste à contrôler le robot suivant sa cinématique inverse.

Les périphériques de saisie développés pour les applications de réalité virtuelle sont nombreux. Certains sont des prototypes de laboratoire de recherche, mais beaucoup sont déjà des produits commercialisés.

L'utilisation de senseurs disponibles sur le marché garantit une simplicité de mise en oeuvre et la possibilité d'obtenir une configuration identique pour différents postes de travail, ou de reproduire le même environnement d'application dans d'autres laboratoires de recherche.

Une présentation de la panoplie des senseurs existants sort du cadre ce travail et le lecteur intéressé pourra se reporter aux références [Burdea 93],

[Pimentel 95] [VRNews 97]. Nous présentons simplement dans cette section deux périphériques à six degrés de liberté. Le propos n'est pas de décrire chaque périphérique en lui-même mais plutôt leurs modes d'actions qui diffèrent par le type d'information qu'ils fournissent: soit des valeurs incrémentales, soit des positions/orientations absolues.

### 6.1.1.1 Senseur incrémental

Ce type de senseur se comporte comme un joystick qui possède six degrés de liberté (3 en translation et 3 en rotation) au lieu de deux seulement. La Figure 6-1 présente le périphérique Magellan utilisé pour le projet «CINEGEN». L'utilisateur peut manipuler la partie préhensible (en forme de cylindre) montée sur des ressorts dans toutes les directions. L'amplitude des mouvements est d'environ 1.5mm en translation et 4° en rotation. Les capteurs mesurant les déplacements sont basés sur un système optique (diodes et photorécepteurs).

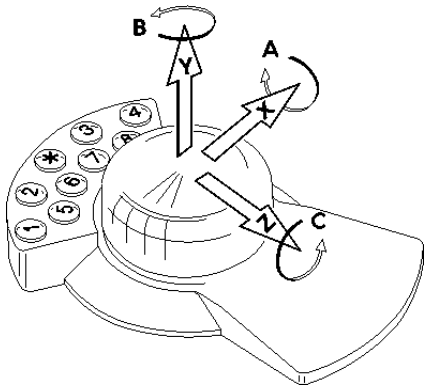



**Figure 6-1:** *Le périphérique de saisie incrémental Magellan*

La consigne générée par ce senseur est fonction du déplacement ou de la rotation qui lui est appliquée. Dans certains autres systèmes où ces déplacements sont de très faible amplitude, ce sont la force et/ou couple appliqués qui fournissent la consigne[2]. L'objet contrôlé se déplace dans la direction où l'on "pousse" le senseur à une vitesse proportionnelle à l'amplitude du mouvement ou à la force appliquée. Il tourne à une vitesse proportionnelle à la rotation ou au couple généré par l'utilisateur.

Ce type de senseurs est le plus approprié pour le contrôle des robots dans le cas du programme *VirtualRobot* puisque la résolution des contraintes s'effectue dans l'espace des vitesses. Ainsi un ensemble de valeurs de déplacement généré par le senseur sera directement utilisé, à un facteur d'échelle près, comme vecteur consigne pour la résolution des contraintes (cf. Section 3.4.3.1).

### 6.1.1.2 Senseur absolu

Les senseurs de type absolu sont composés d'un élément qui est positionné dans l'espace par rapport à un élément de base fournissant une référence fixe.

2. Par exemple la SpaceBall de Spacetex IMC Corp..

Les moyens de positionnement peuvent être mécanique, magnétique, optique ou à ultrasons (pour les cas les plus courants). La souris 3D représentée sur la Figure 6-2 est un périphérique de ce type. Le positionnement de la souris proprement dite est effectué par la mesure de 9 (3x3) distances entre trois haut-parleurs à ultrasons placés sur le triangle de référence et trois micros placés sur la souris. Ce senseur renvoie les informations de position et orientation absolue dans l'espace.

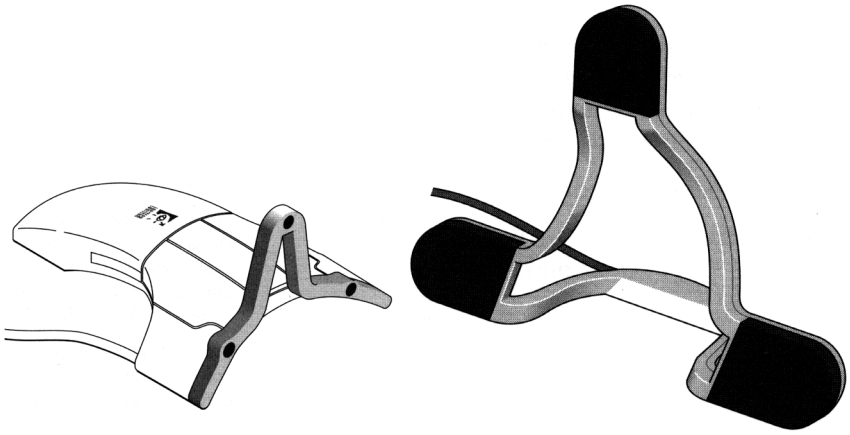


**Figure 6-2:** *Le périphérique de saisie absolu "Souris 3D"*

Ce type de senseur est idéal pour contrôler des robots dont on connaît le modèle géométrique inverse. Cet outil de pointage intuitif avait été utilisé dans un environnement de téléopération d'un robot particulier qui avait permis la réalisation de la première téléopération transatlantique. Ce projet réalisé au sein du Département de Microtechnique [Flückiger 94] est d'ailleurs, avec le travail de Piguet [Piguet 94], la source d'inspiration principale pour le travail de thèse présenté ici.

Dans le cas de *VirtualRobot* il est nécessaire d'utiliser une transformation intermédiaire pour fournir non pas des postures absolues au programme de résolution de contrainte, mais des valeurs incrémentales.

### 6.1.2 Interface classique

Nous entendons par "interface classique" l'interaction grâce à une souris conventionnelle (voire des touches du clavier) avec des éléments graphiques plans usuels.

Deux panneaux de contrôle – constitués d'ascenseurs, boutons et champs de saisie – permettent de fournir des consignes pour réaliser le contrôle de la cinématique directe ou inverse du robot. Ils servent essentiellement dans deux cas de figure:

- on ne dispose pas, ou ne désire pas utiliser un périphérique 3D pour une application ou tâche particulière,
- on essaie d'obtenir un contrôle plus fin du déplacement (au détriment d'un contrôle naturel) et/ou l'on veut découpler totalement chaque degré de liberté.

### 6.1.2.1 Contrôle des “liens”

Un panneau de contrôle spécifique a été créé pour maîtriser chaque paramètre Kleinfinger-Khalil d’un lien. L’utilisateur peut “ouvrir” un tel panneau simplement en cliquant sur le lien désiré (évidemment plusieurs de ces panneaux peuvent être ouverts simultanément).

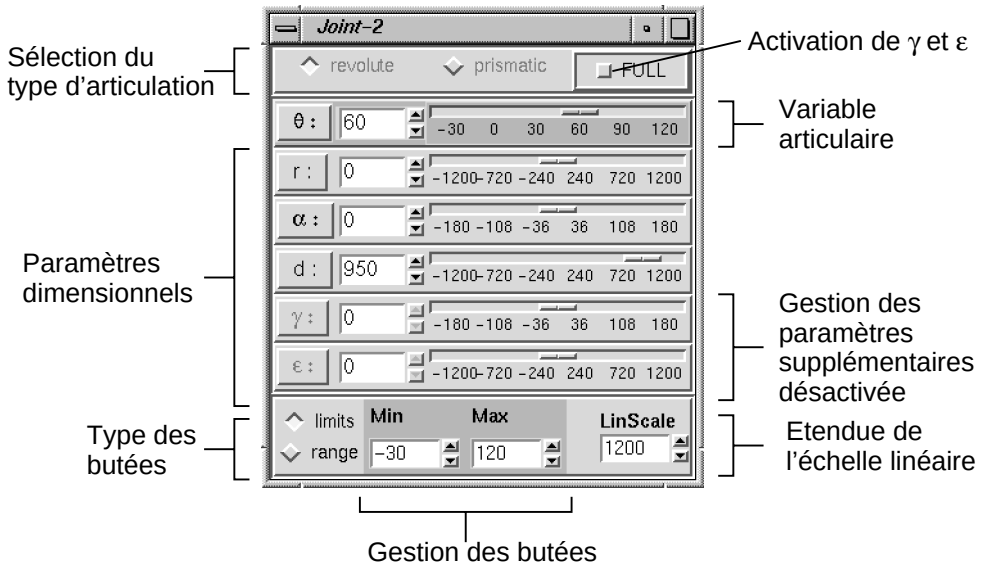



**Figure 6-3:** *Panneau de contrôle des liens du robot*†

†. Un exemple détaillé d’un sous élément de contrôle d’une variable est donné dans la Figure 7-2 sous le terme “mega-widget” du glossaire.

La Figure 6-3 montre un exemple de ce type de panneau de contrôle composé de trois parties principales:

1) La partie supérieure autorise la modification éventuelle du type d’articulation du lien (prismatique ou rotoïde). Un bouton permet aussi d’activer ou désactiver la possibilité de contrôle des paramètres supplémentaires Kleinfinger-Khalil pour les branches.

2) La partie centrale de contrôle proprement dite comprend 6 *mega-widget* permettant de changer individuellement les valeurs de chaque paramètre Kleinfinger-Khalil. Ces mega-widget offrent trois possibilités de contrôle: champ de saisie, boutons d’incrément/décrément et ascenseur. Cette redondance donne à l’utilisateur une flexibilité maximale.

3) La partie inférieure rend possible le contrôle des butées de l’articulation en sélectionnant son mode (absolu ou relatif) et ses valeurs (inférieure et supérieure).

Suivant le type de l’articulation, prismatique ou rotoïde, le mega-widget correspondant (soit contrôle de r, soit contrôle de θ) se met automatiquement en évidence, par changement de couleur, afin de différencier cette variable articulaire des autres paramètres dimensionnels. Le contrôle du robot suivant sa cinématique directe s’effectue par la modification de cette variable articulaire.

#### 6.1.2.2 Contrôle de la cinématique inverse

*VirtualRobot* offre un panneau de contrôle qui se comporte comme un senseur 3D et permet donc de fournir des consignes en vue d'effectuer la cinématique inverse. Ce senseur, un peu particulier puisqu'il agit grâce à une interface graphique conventionnelle, se déclare de la même façon que les autres dans un fichier MAD (cf. Section 4.1). Il peut donc servir à "piloter" soit un référentiel soit un lien. La Figure 6-4 montre l'apparence de ce senseur que l'on gère avec la souris conventionnelle.

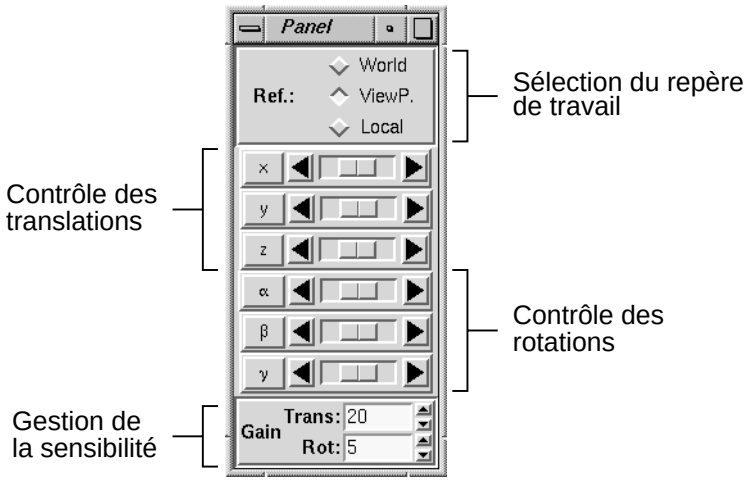



**Figure 6-4:** *Panneau de contrôle pour la cinématique inverse*

Ce panneau de contrôle offre à nouveau un mode d'interaction redondant. L'utilisateur peut soit déplacer un point du robot par incréments en cliquant sur le bouton correspondant à la direction choisie, soit déplacer ce point en continu grâce au petit ascenseur qui module la vitesse de déplacement. Les trois axes de translation ainsi que les trois axes de rotation sont contrôlés indépendamment dans deux directions.

Le référentiel dans lequel les consignes sont exprimées est choisi à l'aide de la partie supérieure du panneau. La partie inférieure permet de régler la sensibilité du senseur, c'est-à-dire le facteur d'échelle entre un incrément élémentaire et la valeur consigne correspondante fournie à l'algorithme de résolution de contrainte.

### 6.1.3 "Syntaxeur" dédié à la robotique

Les canaux standards de transmission de l'information (senseur 3D et affichage) utilisés dans le programme *VirtualRobot* sont essentiellement unidirectionnels: les senseurs fournissent une consigne au programme et la visualisation permet l'appréciation de la tâche. Pour compléter ce mode d'interaction, un nouveau type de périphérique de saisie muni d'un retour d'efforts a été conçu lors de ce travail de thèse: un syntaxeur.

Le terme *syntaxeur* exprime l'idée que l'on peut "s'adresser" (pour donner des consignes) à un robot avec un langage particulier. Ce terme est utilisé dans [Coiffet 93] pour décrire la partie maître d'un système de

téléopération dans le cas où elle n'est pas le reflet de la structure du robot esclave, mais un périphérique de commande générique permettant une variété de modes de pilotage.

L'utilisation de systèmes à retour d'efforts dans les applications de téléopération ou réalité virtuelle n'est évidemment pas nouvelle. Toutefois le concept du périphérique – dédié à la définition de tâches robotiques – développé est original grâce à son mode de contrôle général et son découplage des consignes de translation de celles de rotation. Cette section décrit le concept de ce nouveau "syntaxeur".

#### 6.1.3.1 Motivations

La Section 6.1.1 a présenté deux types de senseurs différents: incrémental versus absolu. Chacun possède des avantages et des inconvénients en fonction de l'utilisation envisagée.

Pour définir un mouvement dans l'espace, il semble naturel de déplacer sa main suivant la trajectoire désirée. Cette idée est centrale dans le concept de réalité virtuelle avec immersion totale: ce sont les mouvements des différentes parties du corps qui sont prises (sans transformation) pour consignes du système à contrôler.

L'ensemble bras-main permet aisément d'effectuer des mouvements dans l'espace pour atteindre une position donnée. Toutefois, à cause de sa cinématique, la chaîne bras-main est beaucoup plus limitée que la plupart des poignets de robots pour les rotations. En effet l'amplitude des rotations au niveau de l'organe terminal que peuvent effectuer usuellement les robots est bien supérieure à celle du poignet humain. Ceci amène à des difficultés de contrôle en rotation. Appelons gain le rapport entre les angles de la consigne envoyée au robot et l'angle correspondant du poignet humain. Si le gain est unitaire, on ne peut générer que des petites rotations[3]. Si l'on augmente trop le gain, on perd alors en précision.

Ces considérations ont amené à concevoir un syntaxeur dont le mode de travail est mixte: les translations sont définies de manière absolue et les rotations de manière incrémentale. Ceci permet de tirer le meilleur parti de l'ensemble bras-main pour la génération de consignes pour un robot.

De plus, le syntaxeur est conçu pour être motorisé afin de se comporter comme un système à retour d'efforts. Le fait de pouvoir générer des forces et couples augmente fortement le contrôle intuitif du robot virtuel. Ces forces généralisées permettent d'appréhender la structure mécanique du robot. On peut par exemple sentir lorsqu'on arrive en butée avec certaines articulations. Il est aussi possible de percevoir directement les interactions du robot piloté avec son environnement. En plus de ce type de forces qui ont un équivalent réel, il est évidemment possible de générer avec le syntaxeur des forces et couples permettant de fournir des informations sur d'autres types de

3. Une possibilité pour obtenir de plus grandes rotations est d'utiliser le principe de la souris que l'on déplace sur son tapis: lorsque l'on arrive au bord du tapis et que l'on veut prolonger le mouvement du curseur dans la même direction, on soulève la souris pour la repositionner sur le bord opposé. Ceci implique toutefois d'effectuer le mouvement en plusieurs étapes et d'inhiber l'effet du senseur lorsque le poignet se replace.

paramètres. On peut par exemple imaginer de fournir une plus grande résistance à l'approche d'une singularité pour dissuader l'utilisateur d'amener le robot dans une position dangereuse (pour le robot).

### 6.1.3.2 Conception du syntaxeur

Le point clef du syntaxeur développé réside dans le découplage des translations absolues et des rotations incrémentales.

- La structure permettant le déplacement en translation est basée sur le concept du robot DELTA [Clavel 91] développé au sein du Département de microtechnique.
- Les rotations sont réalisées par une sorte de petit joystick à trois degrés de liberté en rotation autour d'un point fixe. Ce joystick est fixé sur la nacelle[4] qui se déplace en translation.

Par la combinaison des ces deux structures on obtient les six degrés de liberté nécessaires pour piloter des robots dans l'espace. Les deux structures sont conçues avec des actionneurs qui peuvent générer des forces et couples pour les six degrés de liberté.

Les détails de conception de ces structures peuvent être trouvés dans les travaux suivants: [Villard 97] et [Moser 98]. Les caractéristiques principales du syntaxeur sont fournies dans l'Annexe F, "Caractéristiques du syntaxeur". La Figure 6-5 présente une photo du prototype réalisé.

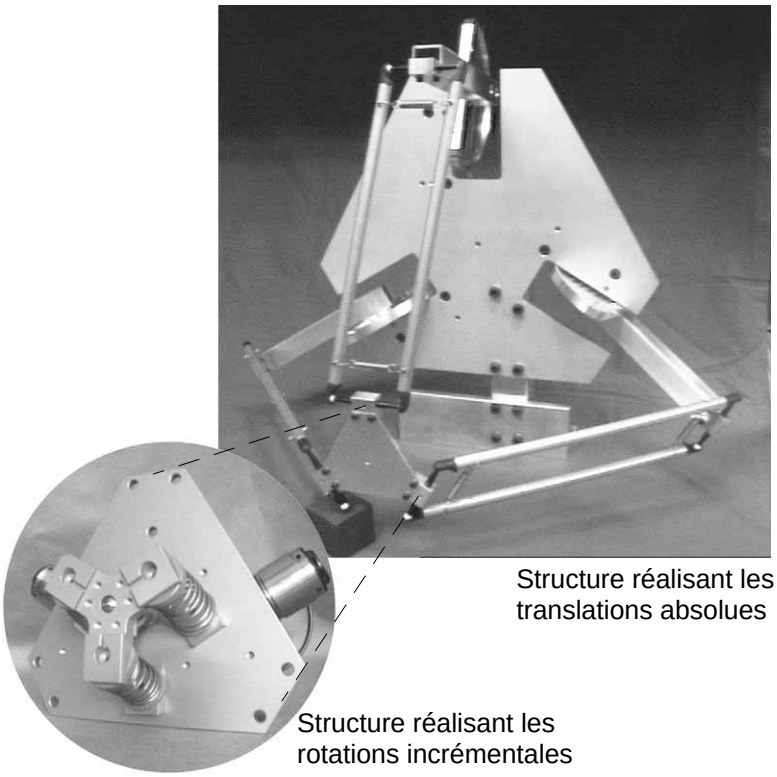



**Figure 6-5:** *Prototype du syntaxeur réalisé*

4. Plaque mobile sur laquelle se terminent les trois chaînes cinématiques du DELTA. Elle reste constamment parallèle à la plaque de base grâce à la structure mécanique.

### *Structure pour les translations: DELTA*

Le concept du robot DELTA présente plusieurs avantages pour la réalisation du syntaxeur:

- structure parallèle avec grande rigidité pour un poids minimal de la partie mobile,
- trois degrés de liberté en translation pure assurés de manière passive,
- actionneurs sur la plateforme fixe.

Le cahier des charges du syntaxeur exhibe certaines différences par rapport à celui d'un robot DELTA destiné à la manipulation industrielle:

- pas besoin de résister à des grandes accélérations ou grandes vitesses puisque c'est l'opérateur qui manipule la structure,
- optimiser le volume de travail par rapport à l'encombrement de la structure afin que le syntaxeur trouve facilement sa place sur une table à côté d'une station de travail,
- rendre le volume de travail aussi homogène que possible (hauteur, largeur et profondeur équivalentes) puisque contrairement à une tâche de manipulation, l'opérateur pilotant un robot ne reste pas dans une zone spécifique.

Pour ces raisons, les caractéristiques de la structure choisie ne sont pas celles désignées comme optimum par [Clavel 91] pour une application de robot industriel. L'homogénéité du volume de travail a été privilégiée au détriment d'apparition de singularités potentielles. La plateforme est fixée dans un plan vertical afin d'optimiser l'interaction bras-main-syntaxeur.

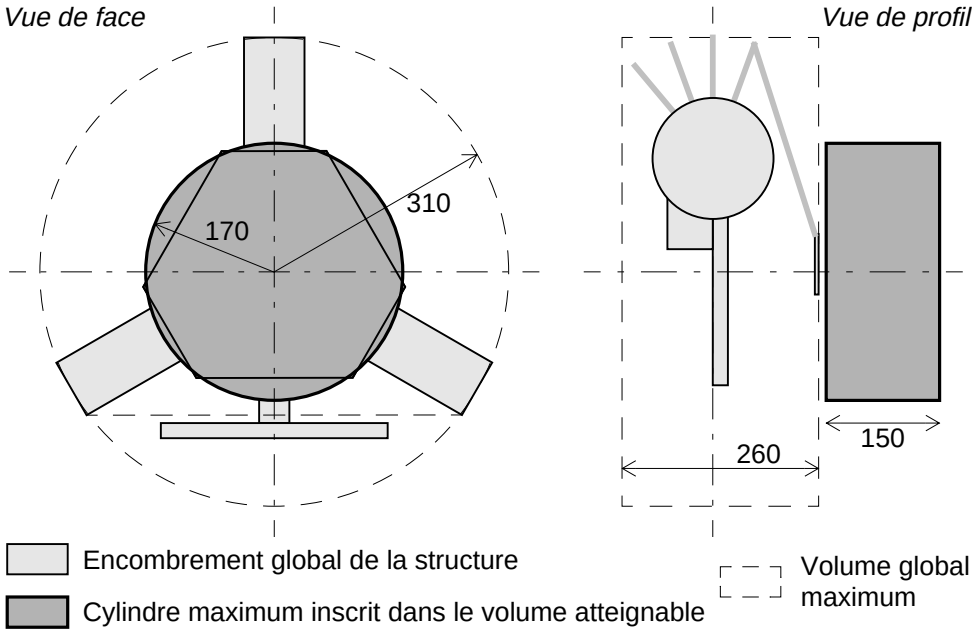



**Figure 6-6:** *Rapport de l'encombrement et du volume de travail du syntaxeur*

La réalisation finale autorise un volume de travail englobant un cylindre de 170mm de rayon pour 150 mm de haut, ce qui correspond à un volume

atteignable raisonnable pour l'opérateur sans déplacer son bras avec trop de fatigue. L'ensemble du syntaxeur en position de repos occupe alors un volume maximal pouvant être englobé dans un cylindre de 310 mm de rayon pour 260 mm de haut. La Figure 6-8 montre le rapport de ces volumes d'encombrement et de travail.

La motorisation est réalisée par trois petits moteurs à courant continu d'une puissance nominale de 20 W. La transmission qui est composée par une courroie tendue sur un large secteur et engrenée sur un petit pignon moteur, permet un rapport de réduction de 16. Cette configuration fourni des forces entre 5 N et 10 N (suivant la position et la direction) au niveau de la nacelle.

### *Structure pour les rotations: PARAMAT*

Le système permettant des rotations dans les trois axes de l'espace se trouve fixé sur la nacelle du DELTA; il a été nommé PARAMAT. Ce mécanisme doit donc être le plus léger possible. Puisque l'on utilise cette partie comme un joystick (valeurs incrémentales) l'amplitude des rotations reste dans la gamme d'un joystick classique. Afin de ne pas introduire de translation parasite, la partie préhensible doit pouvoir tourner dans toutes les directions autour d'un point fixe.

La structure cinématique se compose d'un "mât" central supportant une rotule autour de laquelle s'articule une plateforme mobile. Cette plateforme est reliée à la base par trois actionneurs linéaires. L'ensemble forme donc une structure parallèle à 3 boucles autorisant trois degrés de liberté en rotation pure autour de la rotule. Les actionneurs sont placés de manière asymétrique afin d'éviter les singularités principales. Ils sont constitués par un couple ressort-ficelle en kevlar, ceci afin de simplifier la construction: les rotules nécessaires aux extrémités des actionneurs sont supprimées. Les variations de forces sont réalisées par un moteur à courant continu miniature qui tend la ficelle. La direction des trois actionneurs permet à la nacelle de retrouver une position nominale lorsqu'aucun courant n'est fourni aux moteurs (les trois ressorts produisent alors trois couples qui s'annulent). La Figure 6-7 représente le concept de la structure PARAMAT ainsi que le principe des actionneurs.

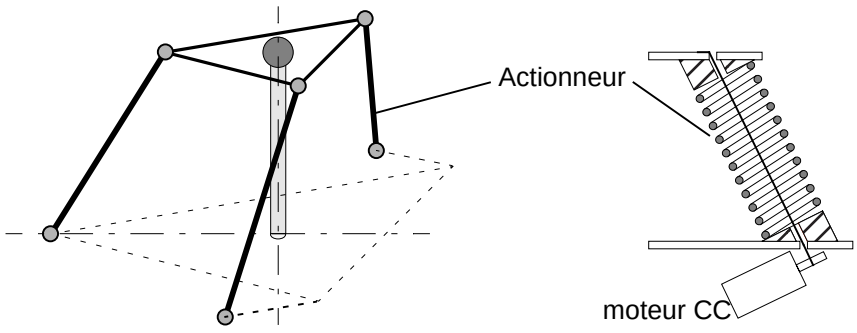



**Figure 6-7:** *Structure et actionneur du système PARAMAT*

Le système PARAMAT réalisé autorise des rotations d'angle de ±15° dans les trois directions. Le système d'actionneurs permet de changer la rigidité de l'ensemble indifféremment suivant les trois axes.

#### 6.1.3.3 Conclusion

Avec le nouveau type de syntaxeur proposé on peut générer des consignes pour un robot manipulateur en tenant compte des caractéristiques bras-main de l'homme. Grâce à son découplage des translations et des rotations une manipulation plus intuitive que les périphériques du marché est réalisable. Le système à retour d'efforts permettra de fermer la boucle de contrôle homme-interface.

## 6.2 Appréciation de la tâche

En dehors du système "syntaxeur" avec retour de forces présenté dans la section précédente, le retour d'information nécessaire au contrôle de la tâche est essentiellement à caractère visuel. L'affichage graphique de la scène tridimensionnelle se fait sur un écran d'ordinateur. L'utilisateur peut soit se contenter de la projection en deux dimensions sur l'écran, soit utiliser des lunettes stéréoscopiques pour obtenir une perception de la profondeur.

L'utilisation première de l'affichage graphique d'une scène virtuelle correspond à une représentation d'objets tridimensionnels modélisant leur équivalent dans le monde réel. Dans le cas de *VirtualRobot*, les robots virtuels ainsi que les autres éléments matériels de la scène (comme les objets à manipuler) sont reproduits sur l'écran sous forme de modèles graphiques. Toutefois, l'utilisation d'environnements virtuels permet d'aller plus loin que la représentation du monde réel[5]. Il est possible d'inclure dans la scène des objets graphiques ne possédant pas une signification matérielle mais fournissant une information supplémentaire.

### 6.2.1 Visualisation 3D

La visualisation de la scène tridimensionnelle dans laquelle évoluent les robots est donc le principal élément d'appréciation du comportement de ces structures articulées en fonction des consignes fournies. L'utilisateur peut à tout moment choisir le point de vue le plus efficace pour évaluer la situation. Ceci est un net avantage par rapport une visualisation réelle d'une scène car il est rarement possible de positionner une caméra vidéo à sa guise. Tout particulièrement lors de la réalisation d'une téléopération, cette possibilité de déplacer la caméra virtuelle est un atout certain que les scientifiques analysant une situation apprécient.

L'affichage peut se faire dans une fenêtre unique ou dans plusieurs fenêtres graphiques simultanément (chaque fenêtre présentant un point de vue

5. Et d'autre part, pour de nombreuses applications, chercher à représenter une scène aussi réaliste que possible n'apporte pas d'information pertinente supplémentaire. Une scène de réalité virtuelle trop "réelle" risque même de détériorer l'information fournie à l'opérateur.

différent). Il est aussi possible d'effectuer un rendu de la scène en mode stéréo. Dans cette dernière situation l'opérateur perçoit directement la profondeur comme dans la vision réelle, ce qui lui permet d'interpréter encore plus facilement le comportement des robots dans l'espace. Par exemple, la vision stéréoscopique des trajectoires des robots représentée graphiquement permet une bien meilleure perception que dans le cas de projection sur un écran standard (l'évaluation d'une trajectoire est plus difficile que celle de la posture d'un robot à cause du manque de référence physique).

Il est important de noter que l'information affichée par un environnement virtuel ne cherche pas à être une copie fidèle de la réalité. Au contraire, il est intéressant de ne sélectionner que l'information pertinente pour l'opérateur. Cette information pertinente peut être par exemple la posture courante du robot représentée par des objets simplifiés et la position des objets à saisir. De plus au lieu de représenter la couleur réelle du robot il est plus utile d'utiliser une couleur représentant par exemple l'effort qu'il subit. De même la couleur d'un rocher n'est pas forcément utile dans la phase de préhension alors que lui assigner une couleur variant avec sa distance à la pince peut être déterminant pour le succès de la tâche. Ces techniques s'apparentent à la "virtualité augmentée" définie dans la section suivante. Dans ce sens les environnements virtuels utilisés pour des tâches d'ingénierie sont plus riches en information que la réalité, même s'ils peuvent paraître plus pauvres, pour l'oeil de l'artiste, qu'une image réaliste créée par infographie.

### 6.2.2 Virtualité augmentée

Si la perception des objets graphiques modélisant une scène réelle est indispensable pour une compréhension intuitive, l'ajout d'informations supplémentaires à l'environnement virtuel améliore considérablement l'appréciation d'une situation.

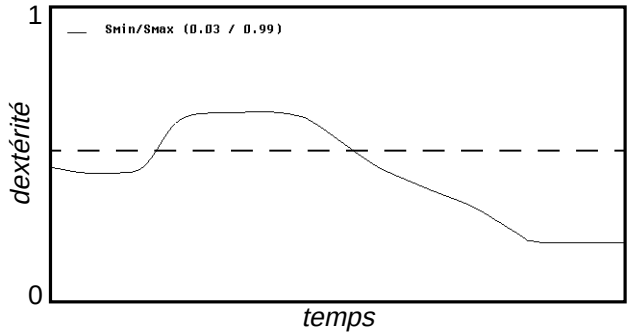



**Figure 6-8:** *Visualisation de la dextérité† d'un robot au cours du temps*

†. La dextérité dans cet exemple est calculée comme le rapport de la valeur singulière maximum sur celle minimum lors de la SVD de la matrice jacobienne

La visualisation de paramètres divers par l'utilisation de courbes dans des graphes conventionnels est possible avec le programme *VirtualRobot* comme le montre la Figure 6-8. Ce type de présentation de l'information est courant dans tous les logiciels de simulation. Il permet d'analyser en détail le comportement d'un système quelconque qui fait l'objet d'une simulation.

Dans le cas des robots, on observe souvent l'évolution de paramètres en fonction du temps pour une trajectoire définie.

*VirtualRobot* introduit en plus le concept de virtualité augmentée: on superpose ou on intègre dans une scène virtuelle des informations qui sont invisibles normalement dans le monde réel. Par analogie avec la réalité augmentée[6] nous nommons virtualité augmentée cette technique.

La virtualité augmentée est réalisée par la représentation symbolique de la variation d'un paramètre quelconque par des objets graphiques directement inclus dans la scène. Par exemple, la Figure 6-9 représente un robot auquel sont superposées des sphères transparentes au niveau de chaque articulations. La taille de chaque sphère est proportionnelle à la vitesse instantanée de l'articulation. Cette méthode montre de manière directe quelles articulations sont le plus sollicitées lors de la réalisation d'une tâche donnée. Ceci procure de bons critères pour optimiser la conception de la structure (par exemple diminution des moteurs sous-utilisés pour gagner du poids)

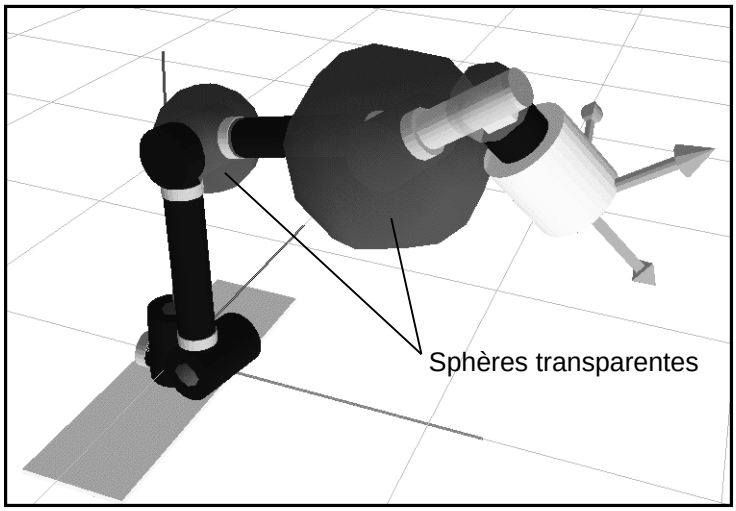



**Figure 6-9:** *Vitesses des articulations symbolisées par des sphères transparentes*

La virtualité augmentée permet de représenter des paramètres internes au robot (vitesses articulaires, accélérations, échauffement d'une pièce, etc.) ou des mesures externes comme la distance d'un point du robot à un obstacle. Il existe des possibilités variées de représenter ces grandeurs [Villard 98] avec des objets supplémentaires ajoutés à la scène:

- variation de la taille d'un objet,
- variation de la forme d'un objet,
- variation de la couleur d'un objet,
- variation de la transparence d'un objet,
- modification de la texture appliquée à un objet.

6. La réalité augmentée est une technique de traitement d'image qui permet de superposer à la vision d'une scène réelle des informations graphiques pour aider à l'appréciation d'une scène. Par exemple on superpose un robot filaire à l'image d'un robot réel (dont on connaît le modèle et les positions courantes) qui évolue dans un environnement avec de la fumée.

Il est évidemment possible de combiner ces différents modes.

Au lieu d'ajouter des objets à la scène, il est aussi possible de modifier directement les propriétés graphiques d'un élément du robot (sa couleur, transparence, texture). Cette technique s'approche de la représentation graphique utilisée pour montrer les résultats de simulations par éléments finis[7]. Toutefois les informations présentées en virtualité augmentée doivent être plus simples car elles doivent être interprétables en temps réel par l'utilisateur. Par contre elles présentent un caractère dynamique lié aux actions de l'utilisateur.

La virtualité augmentée permet d'"attacher" une information directement à la pièce mécanique concernée. L'utilisateur n'a pas à effectuer la liaison mentale entre une courbe d'un graphe et l'objet dont elle provient. Cette méthode peut améliorer considérablement l'évaluation d'un système grâce à la visualisation de paramètres directement en fonction de la tâche que l'opérateur effectue.

## 6.3 Etude de cas

Après avoir présenté les moyens d'interaction que *VirtualRobot* propose à l'utilisateur pour piloter des robots, cette section décrit quelques robots simulés, afin de montrer des exemples concrets de potentialités pratiques offertes par le programme.

Cette section ne traite donc que de certains points mis en évidence par ces exemples. Le détail des fichiers MAD modélisant ces différentes structures sont fournis en Annexe B, "Exemples de fichiers MAD de simulations réalisées".

### 6.3.1 Robot sériel à 5 d.d.l.

Le robot MacDac[8] est développé par McDonnell Douglas. Ce robot est utilisé par la NASA entre autres comme bras manipulateur monté sur le robot mobile Marshokod. La modélisation de ce robot à 5 degrés de liberté en rotations a été réalisée pour présenter la rapidité de mise en oeuvre d'un environnement permettant de téléopérer un nouveau robot. Cette situation se produit en effet dans les missions scientifiques accomplies par la NASA: différents bras manipulateurs peuvent être montés sur une plateforme mobile. Ceci est d'autant plus vrai que le robot MacDac est modulaire et donc configurable à volonté suivant les applications.

La Figure 6-10 montre une image de la simulation du robot MacDac réalisée avec *VirtualRobot*. L'organe terminal porte un carrousel[9] développé spécialement pour une mission d'analyse géologique en Arizona. On voit sur cette image un `référentiel` supplémentaire symbolisé par trois flèches.

7. Par exemple les couleurs de la texture d'un piston représentent la température de chaque point correspondant.
8. Ce nom a été donné par les utilisateurs de ce bras de robot au seul exemplaire existant (le projet dont il est issus n'a finalement pas reçu l'appui de la NASA)

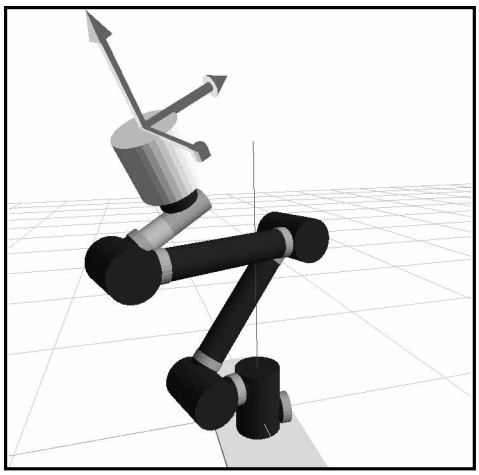


**Figure 6-10:** *Robot MacDac (sériel à 5 d.d.l.)*

Ce référentiel indique à quel niveau agissent les consignes fournies par le senseur manipulé par l'utilisateur.

Il est intéressant de noter que bien que ce robot ne possède que 5 degrés de liberté, il se pilote sans problème avec un senseur à 6 degrés de liberté. En effet, si l'utilisateur produit une consigne que le robot ne peut pas satisfaire, elle ne génère pas d'influence sur les mouvements. Cela libère l'utilisateur de savoir comment assigner les différents degrés de liberté d'un senseur au comportement du robot.

### 6.3.2 Robot hybride à une boucle

Le robot Hitachi-HPR est un robot industriel à 5 degrés de liberté qui comporte une boucle fermée de type parallélogramme. La structure est formée de sept corps mobiles et huit articulations rotoïdes. La Figure 6-11 montre un exemple de ce robot avec des paramètres quelconques.

Pour la description sous forme arborescente, la boucle cinématique a été divisée au niveau de l'articulation arrière-haute. Suivant la formule de Grübler pour les mécanismes tridimensionnels, cette structure est hyperguidée. Toutefois, si l'on considère la boucle comme idéalement planaire le mécanisme fonctionne. Cet hyperguidage peut poser des problèmes à certains programmes de simulation lorsqu'ils analysent la mobilité des boucles.

Dans le cas de *VirtualRobot* cet hyperguidage ne pose pas de problème et il n'est pas nécessaire d'ajouter des articulations supplémentaires pour normaliser le modèle. En effet, la fermeture de la boucle est simplement vue comme une contrainte que le programme essaie de satisfaire. Même si l'on introduit artificiellement une erreur de planéité dans la boucle, l'algorithme continue à fonctionner correctement

9. Ce carrousel est en fait un porte instruments multiple (caméra, spectromètre, pince, etc.) qui encapsule dans un cylindre les différents outils. Un système mécanique permet la rotation de l'ensemble du porte outils afin de présenter celui sélectionné devant une ouverture dans la face plane du cylindre.

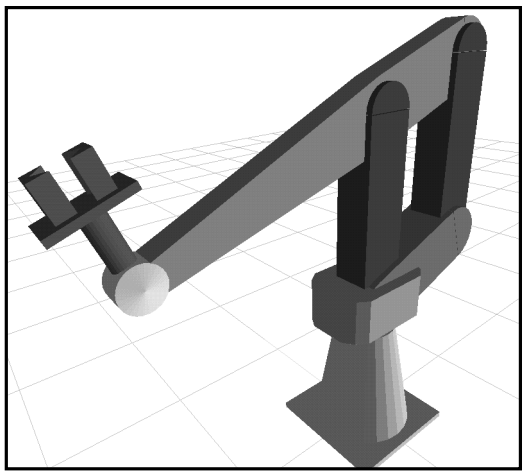


**Figure 6-11:** *Robot Hitachi-HPR*

### 6.3.3 Robot parallèle 3 d.d.l.

La Figure 6-12 montre un robot parallèle à trois degrés de liberté placé sous le fuselage d'un avion.

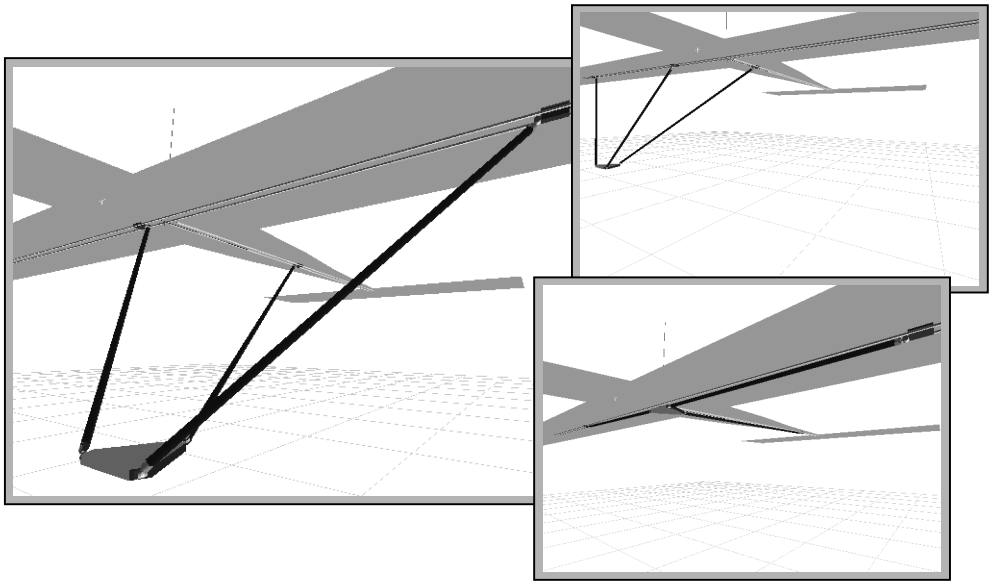


**Figure 6-12:** *Robot parallèle spatial à 3 d.d.l.*

Cette simulation a été créée pour proposer une solution à un problème défini par la NASA: concevoir un robot fixé sous un avion destiné à voler sur Mars, qui puisse analyser des échantillons de sol lorsque l'avion est posé. Le train d'atterrissage place le dessous de l'avion à 1 m du sol. L'envergure des ailes (qui doivent se déployer à l'arrivée sur Mars) atteint 21 m. Le robot doit posséder trois degrés de liberté et porter une charge utile de 4 kg environ. L'optimisation d'un tel robot consiste à obtenir un volume de travail maximum pour une masse minimum. Finalement, l'obligation de "ranger" le robot dans l'avion pour les phases de vol conditionne l'encombrement du robot.

Pour répondre élégamment à ce problème, un robot DELTA modifié a été proposé. Les actionneurs sont linéaires et placés sur deux rails perpendiculaires. La structure est asymétrique (deux actionneurs sur un rail, le troisième sur le second rail) afin d'exploiter au maximum le fuselage de l'avion. Une simulation a pu être réalisée très rapidement pour montrer à des non spécialistes comment ce type de manipulateur évolue, la taille du champ de travail en fonction des longueurs des rails, ainsi que sa facilité de rétraction dans l'avion. Une première estimation a permis de prévoir une structure de 1.2 kg (sans les rails) offrant une surface de travail de 2.8 $m^2$ (sans utiliser les parties pliables des ailes).

Ce manipulateur était à nouveau contrôlé par un périphérique à six degrés de liberté (Magellan). Toutefois, ni le créateur du fichier MAD, ni l'utilisateur n'ont à se soucier du fait que la structure ne possède que trois degrés de liberté et ce uniquement en translation. *VirtualRobot* garantit automatiquement l'orientation constante de la nacelle par rapport à la base en satisfaisant les contraintes des boucles.

### 6.3.4 Structure planaire

Pour montrer les capacités de *VirtualRobot* à résoudre aussi la cinématique directe de structures parallèles, une structure très intéressante au niveau académique a été modélisée. Innocenti l'a en effet utilisée comme exemple d'un robot manipulateur qui peut évoluer d'une *configuration* à une autre sans passer par une singularité[10] [Innocenti 92]. La structure exemple est un manipulateur parallèle plan à trois degrés de liberté (deux translations et une rotation). La plateforme mobile triangulaire est mue par trois actionneurs linéaires. Une articulation rotoïde passive à chaque extrémité des actionneurs les connecte respectivement à la base et à la plateforme mobile.

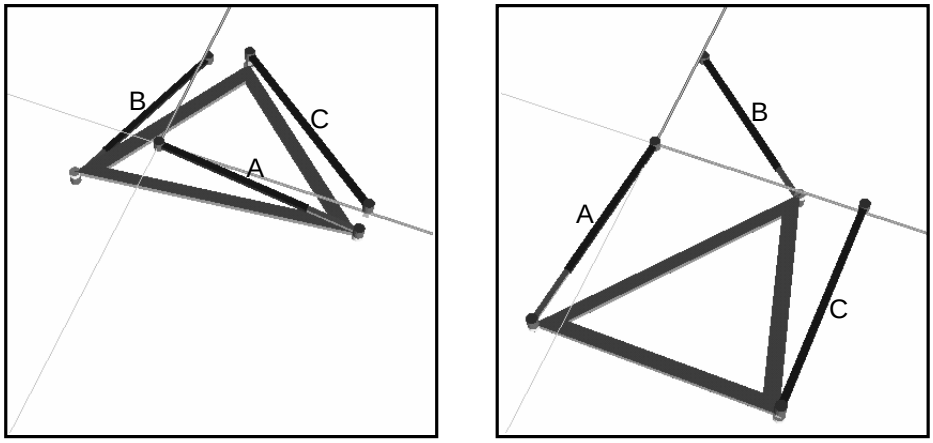



**Figure 6-13:** *Evolution d'un manipulateur parallèle plan*

La création du fichier MAD correspondant à cette structure (en respectant les dimensions fournies dans [Innocenti 92]) a permis d'en vérifier le comportement. Si l'on suit soigneusement les variations de deux actionneurs

10. Cette propriété n'était pas du tout évidente dans l'esprit des roboticiens.

(parcours cyclique) proposées par Innocenti, la plateforme change en effet de configuration sans jamais passer par une singularité. Après le cycle complet des variations articulaires, elle atteint une posture différente alors que l'on retrouve les mêmes valeurs des actionneurs qu'au départ.

En un temps minimal, *VirtualRobot* a permis de créer une simulation vérifiant la théorie d'Innocenti. L'aspect visuel de cette simulation en fait un excellent support de démonstration qui peut par exemple être utilisé dans un but éducatif. D'autre part, cette simulation confirme bien la généralité de *VirtualRobot* qui permet de simuler aussi bien des structures planes que spatiales sans aucune spécification particulière.

### 6.3.5 Robot redondant

Afin de compléter la gamme des structures simulées, un robot imaginaire fortement redondant a été créé. Pour simplifier la mise en oeuvre de cette simulation, la plupart des liens sont représentés automatiquement. En effet, il est possible de spécifier dans le fichier MAD le mot clef `auto` à la place d'un nom de fichier graphique. Dans ce cas, *VirtualRobot* construit automatiquement un objet générique pour représenter le lien correspondant. Ceci est particulièrement utile dans une phase de mise au point d'un robot dont on ne connaît pas encore les dimensions finales. Il serait donc fastidieux de dessiner et modifier constamment les objets graphiques le représentant[11].

La Figure 6-14 montre le robot contenant 6 articulations rotoïdes dont 5 sont à axes parallèles. Ce robot est construit essentiellement avec des liens automatiques. L'utilisation d'objets graphiques volumiques plutôt qu'une simple représentation symbolique (avec des lignes par exemple) permet de mieux appréhender la structure. Il serait d'ailleurs envisageable d'ajouter des paramètres supplémentaires à ces liens générés automatiquement. Par exemple l'épaisseur du bras serait fonction de la charge à transporter et la taille de l'articulation serait proportionnelle au moteur nécessaire pour actionner le bras. Ceci fournirait une vision immédiate de la validité ou non d'une nouvelle structure, destinée à une tâche donnée, en vue d'une conception réelle.

Le comportement de ce robot redondant, contrôlé avec un senseur agissant au niveau de l'organe terminal, permet de bien comprendre la résolution des contraintes qui suit la loi des moindres carrés (cf. Section 3.3.3): pour une consigne dans l'axe privilégié de mobilité du robot, les variations articulaires se répartissent également sur toutes les articulations. Le robot évolue un peu comme un accordéon. Cet exemple permet aussi de démontrer l'utilité de multiples senseurs: la redondance peut-être exploitée de manière manuelle[12] en ajoutant un senseur supplémentaire contrôlant un lien

11. Ce problème disparaît avec un modeleur paramétrique. Toutefois, à la connaissance de l'auteur, aucun outil de CAO de ce type permet d'effectuer des simulations interactives tout en modifiant les paramètres dimensionnels.
12. Au lieu de suivre des critères mathématiques de comportement, l'homme évalue la situation et choisit une solution.

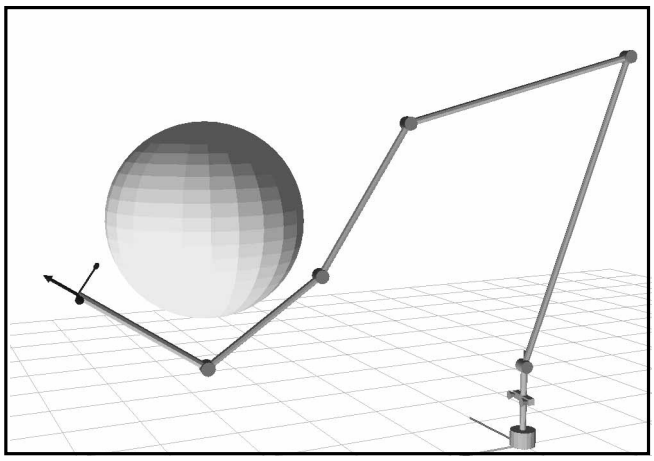


**Figure 6-14:** *Robot redondant*

au milieu de la chaîne. Il est possible de “déformer” la chaîne vers le haut ou vers le bas en fonction de la tâche désirée (par exemple éviter un obstacle).

## 6.4 Résumé

Ce chapitre présente les types d’interactions opérateur–robot virtuel que propose le programme *VirtualRobot*. On en dégage ainsi les avantages de ce type d’interface: manipulation intuitive pour la cinématique directe et inverse, visualisation tridimensionnelle et virtualité augmentée. Un nouveau type de périphérique de saisie avec retour de forces destiné au contrôle de robot est aussi décrit.

La dernière partie de ce chapitre qui clôt ce mémoire montre à travers des exemples de simulation de structures variées les différentes fonctionnalités que le programme *VirtualRobot* offre: manipulation aisée de robot sous-actionnés ou redondants, simulation de structure planes ou spatiales (hyperguidées ou non) sans programmation particulière, prototypage rapide de nouvelles structures.

CHAPITRE

# 7

# CONCLUSION

La simulation de mécanismes, les méthodes de téléopération de robots et les interfaces homme-machine en général sont devenus des domaines clef dans la recherche et l'industrie. Ils permettent la maîtrise de la conception et du contrôle de dispositifs complexes. Le projet «CINEGEN» qui fait l'objet du présent mémoire apporte des contributions à ces trois domaines en proposant de nouveaux concepts d'interaction homme-machine pour le pilotage et l'analyse de robots. Ces concepts sont démontrés par la réalisation d'un programme, *VirtualRobot*, permettant une manipulation intuitive dans un environnement virtuel de structures mécaniques articulées, à partir de leur description par un simple fichier texte.

Pour montrer le réel besoin d'outils du type de *VirtualRobot* il est intéressant de citer l'exemple de l'évolution d'un produit commercial. La jeune compagnie Knowledge Revolution, active dans le domaine de la simulation de systèmes mécaniques, a été fondée en 1989 simultanément à l'émergence d'un réel marché pour ce type de produit (rendu possible par l'augmentation de la puissance des ordinateurs). A la fin 1994 – marquant le début de ce travail de thèse – une version arrivée à maturité de son principal produit WorkingModel v2.0 qui permet la simulation dynamique de systèmes planaires est disponible pour Windows. Au milieu de l'année 1996 WorkingModel 3D, étendant les fonctionnalités de la version précédente aux systèmes spatiaux arrive sur le marché. La dernière version de ce logiciel (printemps 1998), permet d'effectuer une simulation en déplaçant les objets graphiques avec une souris. De l'exemple de cette compagnie qui est aujourd'hui un leader mondial dans la simulation dynamique, on peut tirer un double constat: 1) les logiciels de simulation deviennent indispensables et représentent un énorme marché; 2) le type d'interface classique qu'ils proposent manque de convivialité (même si la tendance se dirige vers une meilleure interactivité).

La notion essentielle qui est encore négligée dans les produits commerciaux est la prise en compte du facteur humain. Le projet «CINEGEN» propose des méthodes pour résoudre le problème d'une meilleure insertion de l'homme dans le processus de développement et

d'interaction avec des machines. Les nouveaux concepts d'interface homme-robot proposés dans cette recherche ont abouti à un outil original avec des potentialités directement utilisables dans le monde de la recherche (étude de nouvelles structures et définition de tâches pour des robots quelconques) et avec une technologie valant certainement la peine d'être transférée dans le monde industriel.

## 7.1 Contributions et résultats

La force du système «CINEGEN» repose sur l'intégration de différents domaines – la Réalité Virtuelle, la cinématique des robots, la programmation objet, les périphériques de saisie et les systèmes mécaniques à retour de forces – dans un outil unique d'interface entre l'homme et la machine. Cette intégration se manifeste encore au niveau du programme proprement dit, qui grâce à l'alliance de plusieurs outils informatiques offre les performances nécessaires ainsi que la fiabilité et l'ouverture requise.

En tant qu'interface homme-robot, le système «CINEGEN» contrôle les échanges d'informations bidirectionnels entre l'opérateur humain et la machine. La Figure 7-1 schématise les interactions apparaissant dans le projet «CINEGEN». Le système bouclé homme-machine permet la prise en compte du facteur humain dans le pilotage et l'analyse de robots. La réalisation originale de chaque système ou moyen de communication schématisé sur la Figure 7-1 ainsi que l'intégration de ces différents éléments offre un type d'interface homme-robot innovant.

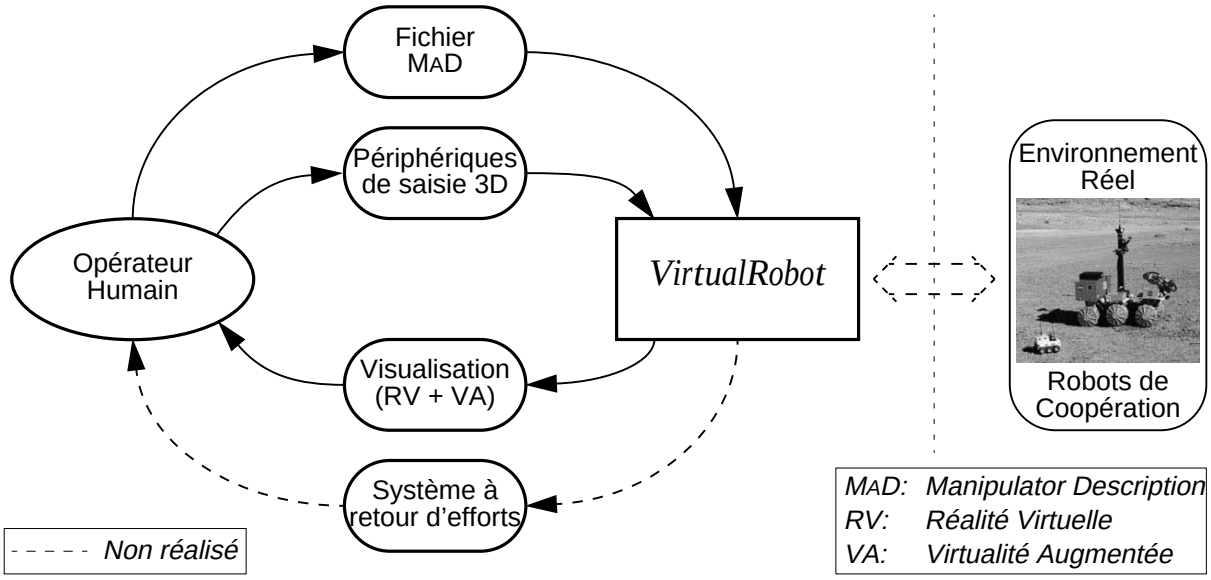



**Figure 7-1:** *«CINEGEN», un système homme-machine bouclé*

Ces éléments clefs représentant les contributions de ce travail de thèse sont résumés dans les points suivants.

- **Fichier de description des robots MAD**: la mise au point d'une représentation unifiée des structures articulées quelconques (sérielles,

arborescentes ou bouclées) permet la description des robots dans un format de fichier spécifiquement développé pour cette application. Il sert non seulement à stocker les caractéristiques géométriques des robots, mais aussi leurs types d'interactions (contraintes) avec l'opérateur. Grâce à ce format de fichier on peut décrire de multiples robots dans la même scène et on peut mettre en oeuvre un environnement graphique complet en un temps minimum.

- **Périphériques de saisie 3D**: l'utilisation de périphériques autorisant la définition de consignes dans l'espace, et surtout l'usage que le programme fait de ces consignes (traitement automatique de la redondance, des systèmes sous actionnés, des mécanismes contraints spatialement) offre un mode d'interaction intuitif avec les robots que l'opérateur analyse ou pilote.
- **Le programme** *VirtualRobot* : il construit automatiquement, sans devoir être recompilé, un environnement virtuel avec les modèles des robots décrits dans un fichier MAD. Pendant toute la durée de l'analyse ou du pilotage des robots, il résout en permanence et en temps réel les contraintes gouvernant les robots (suivi des consignes de l'opérateur et respect des boucles cinématiques). Ce moteur de résolution des contraintes est spécifique aux robots manipulateurs et permet de faire évoluer n'importe quelle structure articulée suivant sa cinématique directe ou inverse. Grâce à sa modularité, le programme *VirtualRobot* pourra aussi réaliser très facilement la liaison avec des robots réels dans le cas de téléopérations. Enfin, l'ouverture et l'extension aisée de ce programme permet de partager la simulation avec d'autres programmes de visualisation.
- **La visualisation 3D**: le programme *VirtualRobot* renvoie en permanence l'information de l'évolution des robots à l'utilisateur en fonction de ses consignes. La visualisation basée sur les techniques de Réalité Virtuelle (RV) permet une appréciation du comportement des robots grâce à leur représentation graphique dans un monde tridimensionnel. La Virtualité Augmentée (VA) offre des outils supplémentaires pour l'évaluation de paramètres au cours d'une tâche donnée.
- **Le système à retour d'efforts**: le nouveau concept de syntaxeur développé procure une double fonctionnalité. Il permet la génération de consignes destinées aux robots de manière optimale grâce au découplage des translations et rotations. Il permettra surtout une meilleure évaluation du comportement des robots à travers le retour de forces/couples (collisions, singularités, attraction vers des zones d'intérêt, etc).

Le programme *VirtualRobot* concrétise une étape manquante entre les logiciels commerciaux de simulation et les systèmes de téléopération destinés à une mission particulière. En effet les logiciels de simulation sont très performants au niveau de l'analyse dynamique des systèmes mécaniques; cependant ils sont limités d'une part dans leur capacité de communication

(uniquement transfert avec d'autres programmes de CAO) et d'autre part dans leurs possibilités d'intégration avec d'autres environnements. Quant aux systèmes de téléopération existants, ils sont le plus souvent "faits main" pour une application spécifique. Par exemple la téléopération d'un nouveau bras de robot demandera une période d'adaptation du programme. *VirtualRobot* permet la simulation de mécanismes quelconques et assure une ouverture vers tout autre environnement de téléopération.

## 7.2 Orientations futures

Le programme *VirtualRobot* concrétise l'aboutissement de cette recherche. Par ailleurs, sa conception même procure une ouverture à plusieurs développements intéressants. En effet, plutôt qu'un programme fermé, *VirtualRobot* doit être compris comme une bibliothèque de fonctionnalités pouvant s'insérer dans d'autres applications ou être étendue sans changement à la base. A cette ouverture informatique correspondent des potentialités de recherches prometteuses dont les principales sont résumées ci-dessous.

- La modélisation d'autres types de liaisons mécaniques (comme des additionneurs mécaniques, cames, etc.) augmenterait les possibilités de simulation de structures articulées. Dans le même contexte, étendre le formalisme dédicacé aux robots manipulateurs pour autoriser la simulation de robots mobiles permettrait de combiner la simulation de bras de robots sur des plateformes mobiles. Le résolveur de contraintes devrait être modifié pour tenir compte des nouveaux types de mobilités introduites (déplacement non holonome de robots mobiles, additionneurs mécaniques, etc).
- La gestion des problèmes survenant pendant l'analyse ou le pilotage de structures articulées (comme singularités mécaniques) pourrait être améliorée grâce à la généralisation de méthodes semi-manuelles faisant intervenir les capacités décisionnelles de l'opérateur. Dans ce contexte il serait avantageux d'introduire des méthodes d'interaction multi-modales (geste + voix + oeil par exemple) pour un meilleur contrôle.
- La liaison de *VirtualRobot* avec un module assurant la communication avec des robots réels autoriserait la réalisation de téléopération de systèmes robotisés. Différents modes de téléopération seraient envisageables (exécution différée, simultanée avec délais, etc.). Il serait alors nécessaire que *VirtualRobot* puisse tenir compte d'informations provenant de l'analyse de la scène réelle par différents systèmes de saisie et de capteurs.
- La réalisation finale du nouveau type de syntaxeur élaboré ajoutera une dimension trop souvent négligée dans les systèmes de simulation: la sensation de force. Le développement de ce syntaxeur est certainement prometteur au vu de l'importance que prennent les outils de ce type pour l'interaction homme machine. Le syntaxeur proposé est un outil de pointage tridimensionnel inédit grâce à son découplage des

translations et des rotations. De plus il permettra une meilleure appréciation de tâches diverses grâce au retour de forces. Une évaluation en situation d'utilisation réelle de ce syntaxeur devra être effectuée en vue son optimisation.

Le projet «CINEGEN» s'appuie sur l'intégration de plusieurs domaines techniques et son avenir repose sur le même principe afin de procurer des nouveaux types d'interactions homme-machine: les canaux de communication peuvent être optiques, mécaniques ou sonores; une base théorique permet de cacher la complexité des calculs à l'opérateur; l'informatique permet la cohésion des différents éléments et procure le moteur de l'interface.

La Réalité Virtuelle, très à la mode lorsque ce projet a débuté, a perdu de son éclat aujourd'hui (pour s'en convaincre on peut comparer le nombre d'ouvrages écrits à ces deux périodes). Cela provient certainement du fait qu'elle n'a évidemment pas pu répondre à la demande d'un emploi universel et idéal. Par contre l'auteur reste convaincu qu'une utilisation adéquate de la Réalité Virtuelle à des domaines précis possède un avenir prometteur dans les interactions homme-machine.

Il faudra garder présent à l'esprit lors de la réalisation de développements futurs que toute interface utilisateur reste avant tout un outil destiné, non pas à la machine, mais à l'homme.

# BIBLIOGRAPHIE


[Artigues 85] Francis Artigues, André Barraco, and Philippe Coiffet. *Dictionnaire de la productique*. Hermes, Paris, 1985.

[Asada 86] H. Asada and J.-J. E. Slotine. *Robot analysis and control*. John Wiley and Sons, New York, 1986.

[Askew 93] R. S. Askew and M. A. Diftler. Ground control testbed for space station freedom robot manipulators. In *IEEE Annual Virtual Reality International Symposium*, Seattle, WA, USA, 1993. IEEE Service Center.

[Bahder 95] Thomas B. Bahder. *Mathematica for scientists and engineers*. Addison-Wesley, 1995.

[Becker 93] T. Becker and V. Weispfenning. *Gröbner basis: a computational approach to commutative algebra*. Springer-Verlag, 1993.

[Brunkhart 94] Mark W. Brunkhart. Interactive geometric constraint systems. Master's thesis TR N° CSD-94-808, University of California, May 1994.

[Buchberger 85] B. Buchberger. Gröbner basis: an algorithmic method in polynomial ideal theory. In Nirmal K. Bose, editor, *Multidimensional systems theory*, volume Chapter 6, pages 184–232. Reidel Publishing, Dordrecht, 1985.

[Burdea 93] Grigore Burdea and Phillipe Coiffet. *La Réalité Virtuelle*. Hermès, Paris, 1993.

[Burdea 96] Grigore Burdea. *Force and touch feedback for virtual reality*. John Wiley & Sons, New York, 1996.

[Christian 97] Daniel Christian, David Wettergreen, Maria Bualat, Kurt Schwehr, Deanne Tucker, and Eric Zbinden. Field experiments with the ames marsokhod rover. In *Field and Service Robotics Conference*, Canberra, Autralia, 1997.

[Clavel 91] Reymond Clavel. Conception d'un robot parallèle rapide à quatre degrés de liberté. Thèse de Doctorat N° 925, EPFL (Ecole Polytechnique Fédérale de Lausanne), 1991.

[Coiffet 92] Philippe Coiffet. *La Robotique, principes et applications*. Traité des Nouvelles Technologies. Hermès, Paris, 3ème édition, 1992.

[Coiffet 93] Philippe Coiffet. *Robot habilis robot sapiens*. Collection Perspective. Hermès, Paris, 1993.

[Craig 89] John J. Craig. *Introduction to robotics*. Addison-Wesley Publishing Company, Massachusetts (etc.), second edition, 1989.

[Crampes 97] Jean-Bernard Crampes. *Interfaces graphiques ergonomiques: conception et modélisation*. Technosup. Ellipse, Paris, 1997.

[Dombre 88] Etienne Dombre and Wisama Khalil. *Modélisation et commande des robots*. Hermès, Paris, 1988.

[Dorf 90] Richard C. Dorf. *Concise international encyclopedia of robotics: applications and automation*. John Wiley & Sons, 1990.

[Doty 93] K. L. Doty, C. Melchiorri, and C. Bonivento. A theory of generalized inverses applied to robotics. *Int J Robot Res*, 12(1):1–19, 1993.

[Eckel 95] Bruce Eckel. *Thinking in C++*. Prentice Hall, Englewood Cliffs, 1995.

[Flückiger 94] Lorenzo Flückiger. Robotique, Réalité virtuelle et vision. Rapport Technique IMT-94-01, EPFL (Ecole Polytechnique Fédérale de Lausanne), 1994.

[Fron 94] Annick Fron. *Programmation par contraintes*. Addison-Wesley, Paris, 1994.

[Funda 88] Janez Funda and Richard P. Paul. A comparison of transforms and quaternions in robotics. *IEEE Robotics and Automation*, pages 886–891, 1988.

[Funda 91] J. Funda and R. P. Paul. A symbolic teleoperator interface for time-delayed underwater robot manipulation. In *Proceedings of Oceans '91*, volume 3, pages 1526–1533, Honolulu, HI, USA, 1991. IEEE.

[Gardner 93] J. F. Gardner, A. Brandt, and G. Luecke. Applications of neural networks for coordinate transformations in robotics. *Journal of Intelligent Robotic Systems*, 8:361–373, 1993.

[GL 93] M. J. González-López and T. Reico. The ROMIN inverse geometric model and the dynamic evaluation method. In A. M. Cohen, editor, *Computer algebra for industry*, pages 117–141. John Wiley & Sons, 1993.

[Gogu 96] Grigore Gogu and Philippe Coiffet. *Représentation du mouvement des corps solides*. Mathématiques pour la robotique. Hermès, Paris, 1996.

[Gogu 97] Grigore Gogu, Philippe Coiffet, and André Barraco. *Représentation des déplacements des robots*. Mathématiques pour la robotique. Hermès, Paris, 1997.

[Golub 96] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[Gosselin 92] C. Gosselin, J. Sefrioui, and M. J. Richard. On the direct kinematics of general spherical three-degree-of-freedom parallel manipulator. In *22nd AMSE Mechanism Conference*, volume 1, pages 7–11, Phoenix, 1992.

[Gosselin 94a] C. Gosselin, J. Sefrioui, and M. J. Richard. On the direct kinematics of spherical three-degree-of-freedom parallel manipulators of general architecture. *ASME Journal of Mechanical Design*, 116(2):594–598, 1994.

[Gosselin 94b] Clément M. Gosselin, Louis Perreault, and Charles Vaillancourt. SMAPS: A computer-aided design package for the analysis and optimization of spherical parallel manipulators. In Mohammad Jamshidi, Charles Nguyen, Ronald Lumia, and Junku Yuh, editors, *Robotics and Manufacturing: Recent Trends in Research, Education, and Applications*, volume 5, pages 115–120. ASME Press, New York, 1994.

[Harrison 97] Mark Harrison. *Tcl/Tk Tools*. O'Reilly & Associates, Boston, 1997.

[Heck 96] André Heck. A bird's-eye view of Gröbner bases. Internal paper http://www.can.nl/ heck, CAN Expertise Center, October 11 1996.

[Hiller 92] Manfred Hiller and Manfred Möller. Computer-based kinematical analysis of spatial multiloop mechanisms. In *Flexible Mechanisms, Dynamics, and Analysis*, volume DE-Vol. 47, pages 135–142, Scottsdale, Arizona, 1992.

[Hine 95] Butler Hine, Phil Hontalas, Terrence Fong, Laurent Piguet, Erik Nygren, and Aaron Kline. VEVI: A virtual environment teleoperation interface for planetary explorations. In *SAE 25th International Conference on Environmental Systems*, San Diego, 1995.

[Hollman 93] Joachim Hollman and Lars Langemyr. Algorithms for non-linear algebraic constraints. In Frédéric Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming. Selected Research*, Logic Programming, pages 113–131. MIT Press, Cambridge, 1993.

[Husty 94] Manfred L. Husty. An algorithm for solving the direct kinematic of Stewart-Gough-Type platforms. Technical Report cim-94-1, McGill University, June 1994 1994.

[Illingworth 91] Valerie Illingworth, editor. *Dictionnaire d'informatique*. Hermann, Paris, 1991.

[Innocenti 92] C. Innocenti and V. Parenti-Castelli. Singularity-free evolution from one configuration to another in serial and fully-parallel manipulators. *ASME Journal of Robotics, Spatial Mechanisms, and Mechanical Systems*, 45:553–560, 1992.

[Johnson 96] Eric F. Johnson. *Graphical applications with Tcl&Tk*. M&T Books, New York, 1996.

[Kalawski 93] Roy S. Kalawski. *The science of virtual reality and virtual environments*. Addison-Wesley, 1993.

[Kecskeméthy 94] Andrés Kecskeméthy and Manfred Hiller. An object-oriented approach for an effective formulation of multibody dynamics. In *Computer methods in applied mechanics and engineering*, volume 115, pages 287–314. Elsevier Science, N.H., 1994.

[Kecskeméthy 95] Andrés Kecskeméthy. *Mobile 1.2.6, user's guide and reference manual*. Fachgebiet Mechatronik, Universität Duisburg, Duisburg, 1995.

[Khalil 97] Wisama Khalil and Denis Creusot. SYMORO+: a system for the symbolic modelling of robots. *Robotica*, 15:153–161, 1997.

[Kircanski 95] M. Kircanski. Symbolic singular-value decomposition for simple redundant manipulators and its application to robot control. *International Journal of Robotics Research*, 14(4):382–398, 1995.

[Kleinfinger 86] J. F. Kleinfinger. *Modélisation dynamique de robots à chaîne cinématique simple, arborescente ou fermée, en vue de leur commande*. E.n.s.m, Université de Nantes, 1986.

[Kramer 92] Glenn A. Kramer. *Solving Geometric Constraint Systems: A Case Study in Kinematics*. MIT Press, London, 1992.

[Kuroe 94] Yasuaki Kuroe, Yasuhiro Nakai, and Takehiro Mori. A new neural network learning of inverse kinematics of robot manipulator. In *Internatinal*

*conference on neural networks*, volume V of *IEEE world congress on computational intelligence*, pages 2819–2824, Orlondo, Florida, 1994. IEEE service center.

[Lazard 92] D. Lazard. Stewart plateform and Gröbner basis. In *Advances in Robot Kinematics (ARK)*, pages 136–142, Ferrara, Italy, 1992.

[Li 90] T. Y. Li and Xiaoshen Wang. A homotopy for solving the kinematics of most general six- and five-degree-of-freedom manipulators. In M. McCarthy, S. Derby, and A. Pisano, editors, *Mechanism Synthesis and Analysis*, volume 25, pages 249–252. ASME, Chicago, Illinois, 1990.

[Manseur 92a] Rachid Manseur and Keith L. Doty. A complete kinematic analysis of four-revolute-axis robot manipulators. *Mechanism & Machine Theory*, 27(5):575–586, 1992.

[Manseur 92b] Rachid Manseur and Keith L. Doty. Fast inverse kinematics of five-revolute-axis robot manipulators. *Mechanism & Machine Theory*, 27(5):587–597, 1992.

[McKerrow 93] Phillip John McKerrow. *Introduction to robotics*. Addison-Wesley, 1993.

[Megahed 93] Saïd M. Megahed. *Principles of robot modelling and simulation*. John Wiley and Sons, Chichester, 1993.

[Mehlhorn 98] Kurt Mehlhorn and Stefan Näher. *The LEDA platform of combinatorial and geometric computing*. Cambridge University Press, in press, 1998.

[Merlet 97] Jean-Pierre Merlet. *Les robots parallèles*. Collection robotique. Hermès, Paris, 2nd edition, 1997.

[Moser 98] Roland Moser. Paramat: mini-joystick à retour de force avec 3 degrés de liberté rotatifs. Rapport de projet, EPFL (Ecole Polytechnique Fédérale de Lausanne), Juin 1998.

[Natonek 95] E. Natonek, T. Zimmerman, and L. Flückiger. Model based vision as feedback for virtual reality. In *IEEE Virtual Reality Annual International Symposium*, pages 110–117, Los Alamitos, CA, 1995.

[Nethery 93] John F. Nethery. *Robotica: users's guide and reference manual*, volume 1. ftp://ftp.csl.uiuc.edu/pub/robotica/manual.ps, 1993.

[Nethery nc] John F. Nethery and Mark W. Spong. Robotica: A mathematica package for robot analysis. Technical Report ftp://ftp.csl.uiuc.edu/pub/robotica/robotica_paper.ps, University of Illinois, n.c.

[Noort 98] Alex Noort, Maurice Dohmen, and Willem F. Bronsvoort. Solving over- and underconstrained geometric models. In *Geometric Constraint Solving & Applications*, Ilmenau, Germany, to be published, 1998.

[Ogata 94] Hiroyuki Ogata and Tomoichi Takahashi. Robotic assembly operation teaching in a virtual environment|. *IEEE Transactions on Robotics and Automation*, 10(3):391–399, 1994.

[Omidvar 97] Omid Omidvar and Patrick van der Smagt, editors. *Neural systems for robotics*. Academic Press, 1997.

[Orin 84] David E. Orin and William W. Schrader. Efficient computation of the jacobian for robots manipulators. *The International Journal of Robotics Research*, 3(4):66–75, 1984.

[Parent 83] Michel Parent and Claude Laurgeau. *Langages et méthodes de programmation*. Les robots: Tome 5. Hermès, Paris, 1983.

[Parr 97] Terence John Parr. *Language translation using PCCTS & C++, a reference guide*. Automata, San Jose, 1997.

[Paul 83] Richard P. Paul. *Robot manipulators: mathematics, programming, and control. The Computer Control of Robot Manipulators*. The MIT Press, Cambridge, Massachusetts and London, England, 5th edition, 1983.

[Pieper 69] D. Pieper and B. Roth. The kinematics of manipulators under computer control. In *Second International Congress on Theory of Machines and Mechanisms*, volume 2, pages 159–169, Zakopane, Poland, 1969.

[Piguet 94] Laurent Piguet. General kinematics simulation of serial links mechanisms. Diploma report, EPFL (Ecole Polytechnique Fédérale de Lausanne), February 1994.

[Piguet 95] Laurent Piguet, Terry Fong, Butler Hine, Phil Hontalas, and Erik Nygren. VEVI: A virtual reality tool for robotic planetary explorations. In *Proceedings of Virtual Reality World*, Stuttgart, Germany, 1995.

[Pimentel 95] Ken Pimentel and Kevin Teixera. *Virtual Reality: through the new looking glass*. Mc Graw Hill, New York, 2nd edition, 1995.

[Press 95] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C*. Cambridge University Press, Cambridge, second edition, 1995.

[Raghavan 90] Madhusudan Raghavan and Bernard Roth. Inverse kinematics of the general 6R manipulator and related linkage. In M. McCarthy, S. Derby, and A. Pisano, editors, *Mechanism Synthesis and Analysis*, volume 25, pages 59–65. ASME, Chicago, Illinois, 1990.

[Raghavan 91] Madhusudan Raghavan and Bernard Roth. The Stewart plateform of general geometry has 40 configurations. In *ASME Design and Automation*, volume 32-2, pages 397–402, Chicago, 1991.

[Ránky 85] P. G. Ránky and C. Y. Ho. *Robot modelling: control and applications with software*. Springer-Verlag / IFS(Publications) Ltd., UK, Berlin, 1985.

[Richard 96] Paul Richard, Georges Birebent, Philippe Coiffet, Grigore Burdea, Daniel Gomez, and Noshir Langrana. Effect of frame rate and force feedback on virtual object manipulation. *Presence*, 5(1):95–108, 1996.

[Rogue Wave 96a] Rogue Wave. *Lapack.h++: object-oriented library for linear algebra*. Rogue Wave Software, Corvallis, 1996.

[Rogue Wave 96b] Rogue Wave. *Math.h++: object-oriented library for numeric computation*. Rogue Wave Software, Corvallis, 1996.

[Roth 93] Bernard Roth. Computations in kinematics. In Jorge Angeles, Günter Hommel, and Peter Kovàcs, editors, *Computational Kinematics*, volume 28 of *Solid mechanics and its applications*, pages 3–14. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[Roth 94] Bernard Roth. Computationnal advances in robot kinematics. In Jadran Lenarcic and Bahram Ravani, editors, *Advances in Robot Kinematics and Computational Geometry*, pages 7–16. Kluwer Academic Publishers, Dordrecht Boston London, 1994.

[Salomons 95] Otto Willem Salomons. *Computer support in the design of mechanical products*. PhD thesis, University of Twente, 1995.

[Sato 93] Shinichi Sato and Akira Aiba. An application of CAL to robotics. In Frédéric Benhamou and Alain Colmerauer, editors, *Constraint Logic Programming. Selected Research*, Logic Programming, pages 161–173. MIT Press, Cambridge, 1993.

[Schenker 94] Paul S. Schenker and Won S. Kim. Remote robotic operations and graphics-based operator interfaces. In Mohammad Jamshidi, Charles Nguyen, Ronald Lumia, and Junku Yuh, editors, *Robotics and Manufacturing: Recent Trends in Research, Education, and Applications*, volume 5, pages 355–368. ASME Press, New York, 1994.

[Sense8 98] Sense8. *WordToolKit reference manuel, release 8*. Sense8 Corporation, Mill Valley, 1998.

[Shimoga 92] K. Shimoga. Finger force and touch feedback issues in dextrous telemanipulation. In *NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration*, Troy, NY, 1992.

[Silicon Graphics Inc. 98] Silicon Graphics Inc. *IRIX system programming guide, on-line reference manual*. Silicon Graphics Inc., 1998.

[Spring 86] Kerry W. Spring. Euler parameters and the use of quaternion algebra in the manipulation of finite rotations: a review. *Mechanism and Machine Theory*, 21(5):365–373, 1986.

[Stroustrup 97] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley, Reading, third edition, 1997.

[Tancredi 95] Luc Tancredi. *De la simplification et la résolution du modèle géométrique direct des robots parallèles*. Thèse de doctorat, Ecole des Mines de Paris, 1995.

[Tancredi 96] Luc Tancredi, Monique Teillaud, and Olivier Devilliers. Symbolic elimination for parallel manipulators. Rapport de recherche 2809, INRIA, février 1996 1996.

[Tsai 85] L. W. Tsai and A. P. Morgan. Solving the kinematics of the most general six- and five-degree of freedom manipulators by continuation methods. *Journal of Mechanisms, Transmissions, and Automation in Design*, 107(2):189–200, 1985.

[Villard 97] Jimmy Villard. Conception et réalisation d’un syntaxeur pour piloter une interface virtuelle. Rapport de projet, EPFL (Ecole Polytechnique Fédérale de Lausanne), Février 1997.

[Villard 98] Jimmy Villard. 3D visual data and user interface. Diploma report, EPFL (Ecole Polytechnique Fédérale de Lausanne), February 1998.

[Vince 95] John Vince. *Virtual Reality Systems*. Addison-Wesley, 1995.

[VRNews 97] VRNews. 3D navigational & gestural devices. *VR News*, 6(1):28–33, January/February 1997.

[Vukobratovic 86] M. Vukobratovic and M. Kircanski. *Kinematics and trajectory synthesis of manipulation robots*, volume 3 of *Scientific fundamentals of robotics*. Springer-Verlag, Berlin, 1986.

[Vukobratovic 89] Miomir Vukobratovic, editor. *Introduction to robotics*. Springer-Verlag, Berlin, 1989.

[Wampler 90] C. W. Wampler, A. P. Morgan, and A. J. Sommese. Numerical continuation methods for solving polynomial systems arising in kinematics. *Journal of Mechanical Design*, 112:59–68, 1990.

[Wampler 91] Charles Wampler and Alexander Morgan. Solving the 6R inverse position problem using a generic-case solution methodology. *Mechanism & Machine Theory*, 26(1):91–106, 1991.

[Weiss 93] Jürgen Weiss. Resultant methods for the inverse kinematics problem. In Jorge Angeles, Günter Hommel, and Peter Kovàcs, editors, *Computational Kinematics*, volume 28 of *Solid mechanics and its applications*, pages 41–52. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[Welch 97] Brent B. Welch. *Practical programming in Tcl and Tk*. Prentice Hall PTR, New Jersey, 2nd edition, 1997.

[Wernecke 93] Josie Wernecke. *The Inventor Mentor: programming object-oriented 3D graphics with Open Inventor, release 2*. Open Inventor Architecture Group. Addison-Wesley, Reading, 1993.

[Wu 94] Guang Wu and Juang Wang. A recurent neural network for manipulator inverse kinematics computation. In *Internatinal conference on neural networks*, volume V of *IEEE world congress on computational intelligence*, pages 2715–2720, Orlondo, Florida, 1994. IEEE service center.

[Yoshikiwa 90] Tsuneo Yoshikiwa. *Foundations of robotics: analysis and control*. The MIT Press, 1990.

[Young 95] Douglas A. Young. *Object-oriented programming with C++ and OSF/Motif*. Prentice Hall PTR, Saddle River, New Jersey, 2nd edition, 1995.

# URLs

**wwwADA:** ADAMS
*http://www.adams.com/*
Mechanical Dynamics, Inc
2301 Commonwealth Blvd.
Ann Arbor, Michigan 48105
USA

**wwwANA:** Analytix / Kinematix / Dynamix
*http://www.saltire.com/*
Saltire Software
P.O. Box 1565
Beaverton, OR 97075
USA

**wwwANT:** ANTLR
*http://www.ANTLR.org/*
MageLang Institute, LLC
2755 Campus Drive, Suite 130
San Mateo, CA 94403 2755
USA

**wwwCIM:** Cim Robotics Station
*http://www.silma.com/Robo.html*
SILMA, a Division of Adept Technology, Inc.
150 Rose Orchard Way
San Jose, California 95134
USA

**wwwCOC:** The Cocoon Utilities, Version 3.1
*http://www-users.cs.umn.edu/~kotula/cocoon/cocoon.htm*

**wwwDAD:** DADS (Dynamics Analysis and Design Systems)
*http://www.cadsi.com/dads.html*
CADSI

**wwwIGR:** IGRIP / TeleGrip
*http://www.deneb.com/products/igrip/igrip.html*
Deneb Robotics, Inc.
5500 New King Street
Troy, MI 48098
USA

**wwwJAM:** James
*http://www.simulog.fr/FR/html/prods/james.html*
Simulog
1, Rue James Joule
78286 Guyancourt Cedex
FRANCE

**wwwLED:** LEDA Research
*http://www.mpi-sb.mpg.de/LEDA/*

**wwwMEC:** MECHANICA / ProE
*http://www.ptc.com/products/func/mechanica.htm*
Parametric Technology Corporation
128 Technology Drive
Waltham, MA 02453
USA

**wwwSAG:** Logiciels pour les robots parallèles du projet SAGA
*http://www.inria.fr/saga/logiciels/RP/catalogue.html*

**wwwRAP:** RapidApp
*http://www.sgi.com/developers/devtools/languages/rapidapp.html*

**wwwROB:** Robcad
*http://www.tecnomatix.com/*
Tecnomatix Technologies Ltd.
Delta House
16 Hagalim Avenue
Herzeliya 46733
Israel

**wwwROG:** Math.h++ et Lapack.h++
*http://www.roguewave.com/*
Rogue Wave Software, Inc. (Stingray)
9001 Aerial Center, Suite 110
Raleigh, NC 27560
USA

**wwwROO:** The ROOT System
*http://root.cern.ch/root/*

**wwwROP:** ROBOOP - A robotics object oriented package in C++
*http://www.ind2.polymtl.ca/ROBOOP/*

**wwwSEN:** WorldToolKit
*http://www.sense8.com*
SENSE8 Corporation
100 Shoreline Highway, Suite 282
Mill Valley, CA 94941
USA

**wwwTCL:** Tcl/Tk, SunScript
*http://sunscript.sun.com/*

**wwwTIX:** TIX
*http://www.xpi.com/tix*

**wwwVIE:** Iris ViewKit Application FrameWork
*http://www.sgi.com/developers/devtools/apis/viewkit.html*

**wwwVIR:** Documentation de libPtk (bibliothèque de base du projet VIRGY) en ligne
*http://imtsg7/projects/virgy/public/main/main.html*

**wwwVIZ:** *http://img.arc.nasa.gov/~schwehr/Viz/*

**wwwWMO:** WorkingModel
*http://www.workingmodel.com/products/wm3d.html*
Knowledge Revolution
66 Bovet Road, Suite 200
San Mateo, CA 94402
USA

**wwwWSP:** Workspace
*http://www.rosl.com/brochure.htm*
Robot Simulations Ltd.
21 Amethyst Road
Newcastle-upon-Tyne
NE4 7YL
UK

**wwwXFO:** Xforms, a graphical user interface toolkit for X
*http://bragg.phys.uwm.edu/xforms*

# GLOSSAIRE

**analyseur:** Programme ou ensemble de routines permettant d'effectuer l'analyse syntaxique.

**analyse syntaxique:** [Illingworth 91]: Processus qui permet de décider si une chaîne de symboles en entrée est une phrase d'un langage donné, et si oui, de déterminer sa structure syntaxique telle qu'elle est définie par une grammaire du langage [...].

**API:** Application Programmer Interface. Interface programmeur application. L'ensemble des routines qu'un programme utilise pour accéder à des services plus bas niveau. C'est la couche d'une bibliothèque que l'utilisateur perçoit.

[Illingworth 91]: Spécification de la communication entre un programme d'application et un programme utilitaire.

**articulation prismatique:** [Artigues 85]: Liaison de type glissière autorisant exclusivement la translation suivant un axe.

**articulation rotoïde:** [Artigues 85]: Liaison de type charnière autorisant exclusivement la rotation autour d'un axe.

**automatisation:** Utilisation de technologies diverses permettant la production de biens à l'aide de machine automatiques (production assurée auparavant le plus souvent par des humains). Une fois automatisé, le processus de production fonctionne sans intervention humaine. L'automatisation se base sur la programmation d'opérations combinées avec un retour d'information pour déterminer si les commandes ont bien été exécutées.

**call-back:** Fonction appelée en réponse à un événement.

**CFAO:** Conception et Fabrication Assistée par Ordinateur.

**chaîne cinématique:** [Artigues 85]: Par chaîne cinématique nous désignons l'ensemble des éléments mécaniques qui, à partir des actionneurs, transmettent le mouvement jusqu'aux différents éléments en mouvement. Le terme désigne aussi bien la réalité technologique, vis à billes, engrenages, que la modélisation que le concepteur utilise pour représenter cette réalité.

**configuration:** La configuration d'un robot est la posture caractéristique de tous ses liens. Un robot peut souvent atteindre une même position et orientation de l'espace avec plusieurs configurations différentes.

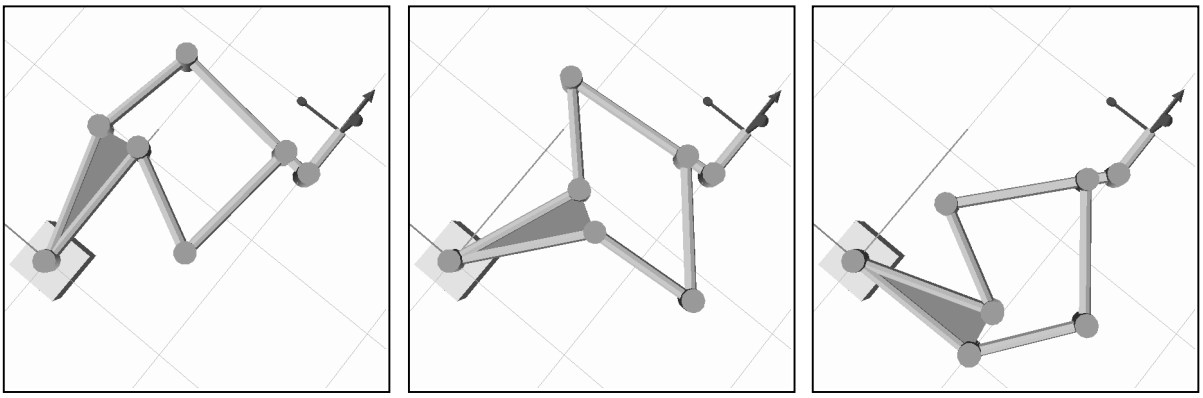


**Figure 7-1:** *Différentes configurations possibles pour un robot hybride redondant*

**d.d.l.:** **D**egré **D**e **L**iberté.

**degrés de liberté:** [Artigues 85]: Le nombre de degré de liberté d'un robot est le nombre de mouvements indépendants que ce robot peut effectuer dans un référentiel lié à sa base. [...]

**GUI:** Graphical User Interface. Interface graphique utilisateur.

**graphe de scène:** Collection ordonnée de noeuds représentant des objets graphiques ou des propriétés de ces objets. Le graphe de scène sert à représenter l'organisation hiérarchique de tous les éléments d'une scène graphique. Il permet au moteur graphique de l'application de "construire" l'environnement tridimensionnel en fonction des positions et attributs des éléments. L'expression "graphe de scène" est traduite littéralement du terme très spécifique anglais "scene graph" utilisé en infographie.

**kinematics:** Suivant [McKerrow 93]: "**Kinematics** *is the relationships between the positions, velocities and accelerations of the links of a manipulator*, where a manipulator is an arm, finger or leg."

**lien:** Partie élémentaire conceptuelle (on ne tient pas compte de la motorisation) composant un bras de robot. Un lien est l'ensemble articulation-corps rigide. L'articulation relie le lien courant au lien précédent en ajoutant un degré de liberté. Le corps rigide relie le lien à l'articulation du lien suivant.

**MAD (fichier):** **MA**nipulator **D**escription. Cet terme exprime soit le fichier de description d'un manipulateur lui-même, soit le "format" de ce fichier.

**mega-widget:** Un élément d'une GUI se comportant comme un objet unique mais composé de plusieurs widgets de base, voir d'autres mega-widgets. Le concept de mega-widget est introduit par la couche Tix permettant justement de créer des nouveaux objets widgets composés de plusieurs widgets Tk.

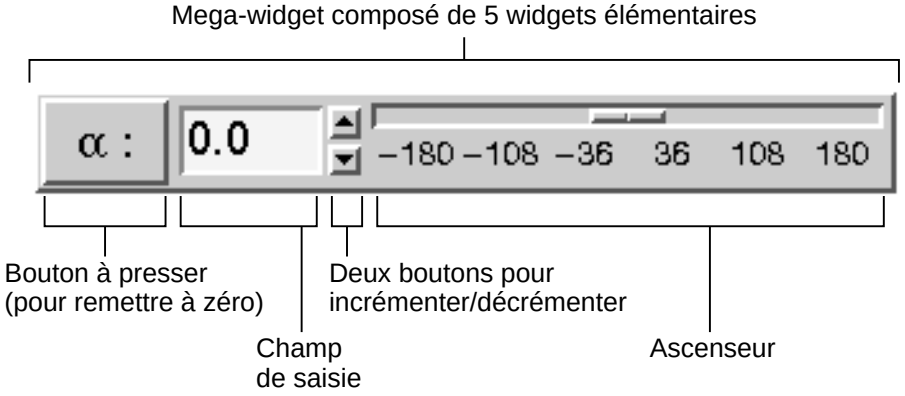



**Figure 7-2:** *Mega-Widget Slider*

Dans l'exemple de la Figure 7-2 tous les widgets élémentaires composant le mega-widget "slider" agissent sur la même variable interne. Ainsi les trois moyens de contrôle sont couplés.

**milieu hostile:** [Artigues 85]: Pour la robotique on désigne par milieu hostile les fonds marins, les zones exposées à des pollutions, à des radiations et de manière générale les lieux d'accès dangereux pour l'homme.

**orientation:** L'orientation d'un objet peut être représentée de nombreuse manières différentes (matrices, quaternions, angles de rotation), mais il est nécessaire d'avoir au minimum trois scalaires pour la représenter (un seul suffit dans le plan).

**parser:** Analyseur ou analyseur syntaxique.

**parser generator:** [Illingworth 91]: Programme qui accepte la description syntaxique d'un langage de programmation et qui génère un analyseur pour ce langage.

**position:** La position d'un objet est représentée par trois scalaires dans l'espace (seulement deux dans le plan) qui placent l'objet dans l'espace par rapport à un référentiel de référence.

**posture:** Le terme posture est utilisé dans ce mémoire pour décrire simultanément la position et l'orientation d'un solide ou référentiel dans l'espace.

Une posture est donc représentée par un minimum de six scalaires. Une des représentations des plus courantes (et utilisées au sein de ce mémoire) sont les matrices homogènes 4x4 de transformation.

**précision:** [Artigues 85]: C'est l'ensemble des paramètres définissant l'incertitude de positionnement absolu de l'organe terminal d'un robot.

**productique:** "Application de l'automatique et de l'informatique au processus de production industrielle", selon le Petit Robert.

[Artigues 85]: C'est l'ensemble des technologies associées à la production industrielle. De manière plus restrictive on réunit sous ce vocable les technologies nouvelles de fabrication, d'usinage et de montage: robotique, machines outils à commande numérique, cellules et ateliers flexibles, les techniques de préparation et gestion assistées par ordinateur: CFAO, GPAO, MAO et enfin la conception et le dessin assisté par ordinateur: CAO, DAO.

**programmation en-ligne:** Programmation "on-line". [Artigues 85]: La phase qui permet d'apprendre au robot ce qu'il devra exécuter par la suite s'appelle la programmation du robot. Lorsque cette phase se fait sur les lieux mêmes où se situent le robot et son environnement on parle de programmation "on-line". La production est arrêtée, le robot lui-même est utilisé et donc cette technique est très coûteuse en temps. Pour cette raison les techniques de programmation "off-line" sont de plus en plus étudiées.

**programmation hors-ligne:** Programmation "off-line". [Artigues 85]: Depuis quelques années les moyens de simulation du comportement des robots ont été développés. Il est possible de simuler, sur un écran graphique, une scène et une opération effectuée par un robot. Le résultat de cette simulation est l'obtention des valeurs des coordonnées articulaires correspondant aux points définissant la trajectoire. A partir de ces données, en ajoutant des ordres spécifiques à chaque robot, il est possible d'engendrer un programme exécutable. La phase d'apprentissage n'utilise plus le robot réel mais le robot virtuel. [...]

**pthread:** Thread définit suivant la norme POSIX.

**référentiel:** Dans ce mémoire un référentiel est l'objet informatique ou conceptuel représentant un repère de l'espace.

**repère:** La donnée de trois vecteurs unitaires formant une base orthonormée (dans ce mémoire) permet de définir un repère dans l'espace tridimensionnel. Un repère permet de caractériser une système de coordonnées par rapport à une autre repère (en position et en orientation).

**répétabilité:** [Artigues 85]: La répétabilité définit la précision avec laquelle un robot passe sur un point d'une trajectoire aux cours des cycles répétitifs définis par programmation ou apprentissage.

C'est l'aptitude d'un robot à repasser par un même point lors de l'exécution répétitive d'une trajectoire. La répétabilité qui est une borne supérieure d'erreur de position n'est pas la même en tout point d'une trajectoire et dépend de la configuration du robot.

**retour d'efforts:** Système mécanique permettant de restituer la sensation d'effort à l'opérateur qui le manipule. Ces systèmes étaient d'abord utilisés pour effectuer des téléopérations avec bras-maître et bras-esclave. Maintenant les systèmes à retour d'effort utilisés en réalité virtuelle sont plus généralistes et permettent d'appréhender aussi bien les forces sur un robot, que la résistance d'un tissus lors d'une opération.

**robot:** Provient du tchèque "robota", travail forcé. La pièce de théâtre "Rossum's Universal Robots" de Karel Capek jouée à Paris vers 1920 popularisa le terme robot. Elle mettait en scène des petits êtres artificiels exécutant parfaitement les consignes de leur maître.

**robotique:** "Ensemble des études et techniques permettant l'élaboration de robots", selon le Petit Robert. Cette définition bien que correcte et acceptée pose certains problèmes dans la communauté scientifique à cause de la délimitation du domaine. D'un côté les puristes considèrent la robotique comme la science des robots en tant qu'outil isolé. De l'autre les personnes percevant le robot comme outil production dans l'industrie étendent le terme à l'ensemble des systèmes permettant l'automatisation (qui ne sont pas forcément des robots!).

[Artigues 85]: Discipline dont l'objet est l'automatisation d'un grand nombre de secteurs de l'activité humaine par utilisation de machines ou systèmes divers de robots.

**robotique manufacturière:** [Artigues 85]: C'est la robotique directement liée à la production industrielle.

**robot manipulateur:** [Artigues 85]: Système mécanique articulé doté d'actionneurs commandés par programme numérique et terminé par un organe de préhension. En langage courant synonyme de robot.

**robotique non-manufacturière:** [Artigues 85]: Robotique qui n'est pas directement liée à la production industrielle, par exemple: robotique agricole, robotique d'exploration spatiale et sous-marine, robotique d'intervention en milieu hostile.

**senseur:** Dans le cadre de «CINEGEN» un senseur est le terme générique pour tout périphérique de saisie utilisé pour fournir des consignes au programme (souris conventionnelle, périphérique 3D du marché ou syntaxeur dédié).

Le contexte détermine si le terme senseur correspond directement au système physique ou si l'on parle de sa représentation abstraite.

**SVD:** Singular Value Decomposition. Décomposition par valeurs singulières d'une matrice.

**syntaxeur:** Un syntaxeur est un périphérique de saisie mécanique utilisé pour fournir des consignes à un système robotisé.

**traqueur:** Objet créé par *VirtualRobot* pour stocker la liste des senseurs qu'un élément (lien ou référentiel) du robot doit suivre. Lorsque le programme analyse le graphe représentant le robot et qu'il découvre un traqueur, il sait alors que l'objet qui est son parent doit tenter de suivre les consignes provenant de senseurs.

**Tcl:** Tool Command Language. Langage de programmation interprété.

**Tcl/Tk:** voir séparément Tcl et Tk. Langage de programmation permettant de réaliser des scripts et des interfaces graphiques portables.

**teach box:** ou "teach pendant". Boîtier mobile d'apprentissage. C'est un boîtier muni de touches que l'opérateur tient dans la main. Ces touches permettent de télécommander le robot pour le déplacer dans différentes positions. La "teach box" sert aussi d'interface entre l'opérateur du robot et le contrôleur du robot. L'apprentissage en utilisant une "teach box" consiste en une séquence de deux actions: 1) déplacement du manipulateur à une position désirée; 2) stockage du point en mémoire.

**thread:** Littéralement processus dans le domaine informatique. La programmation par "threads" permet d'avoir plusieurs processus qui s'exécutent parallèlement au sein d'un même programme.

**Tix:** Extension de Tcl/Tk introduisant le concept de mega-widget. Tix fournit une panoplie de mega-widget prédéfinis et permet de programmer de nouveaux mega-widgets.

**Tk:** ToolKit. Librairie apportant des fonctionnalités graphique à Tcl.

**token:** Un élément textuel non réductible dans un ensemble de données qui est analysé. Par exemple les mots clefs et les variables d'un langage de programmation.

**widget:** Elément d'une GUI composé de plusieurs primitives graphiques de base (lignes, texte, couleurs, etc.) auquel sont assignées des propriétés événementielles. Par exemple un bouton 

est un widget qui va valider une action lorsque l'on clique dessus avec la souris (il va d'ailleurs à ce moment changer ses propriétés graphiques pour paraître "enfoncé").

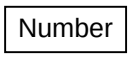
**wrapper:** Un "wrapper" sert à "emballer" un ensemble de fonctionnalités dans une couche de plus haut niveau. Par exemple, WordlToolKit propose un wrapper C++ pour sa bibliothèque écrite en C. Cela permet de fournir à l'utilisateur une API orientée-objet alors que les fonctions sont en fait codées en C (langage procédural).

ANNEXE

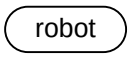
# A

# GRAMMAIRE MAD

La Figure E-1 (répartie sur les trois page suivantes) présente la grammaire MAD (MAnipulator Description) sous forme de diagrammes syntaxiques. Les conventions suivantes sont utilisées:

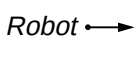


unité syntaxique élémentaire,

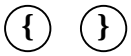


mot clef du langage,

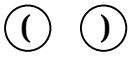
bloc syntaxique défini plus loin,



définition de bloc syntaxique,



début, fin de bloc (caractère identique dans le langage MAD),



début fin de liste (caractère identique dans le langage MAD),

Les unités syntaxiques élémentaires suivantes ne sont pas définies expressément (car communément utilisées).

- Number: un nombre entier,
- Float: un nombre à virgule flottante
- Name: une chaîne de caractères,
- File: un nom de fichier suivant les conventions Unix.

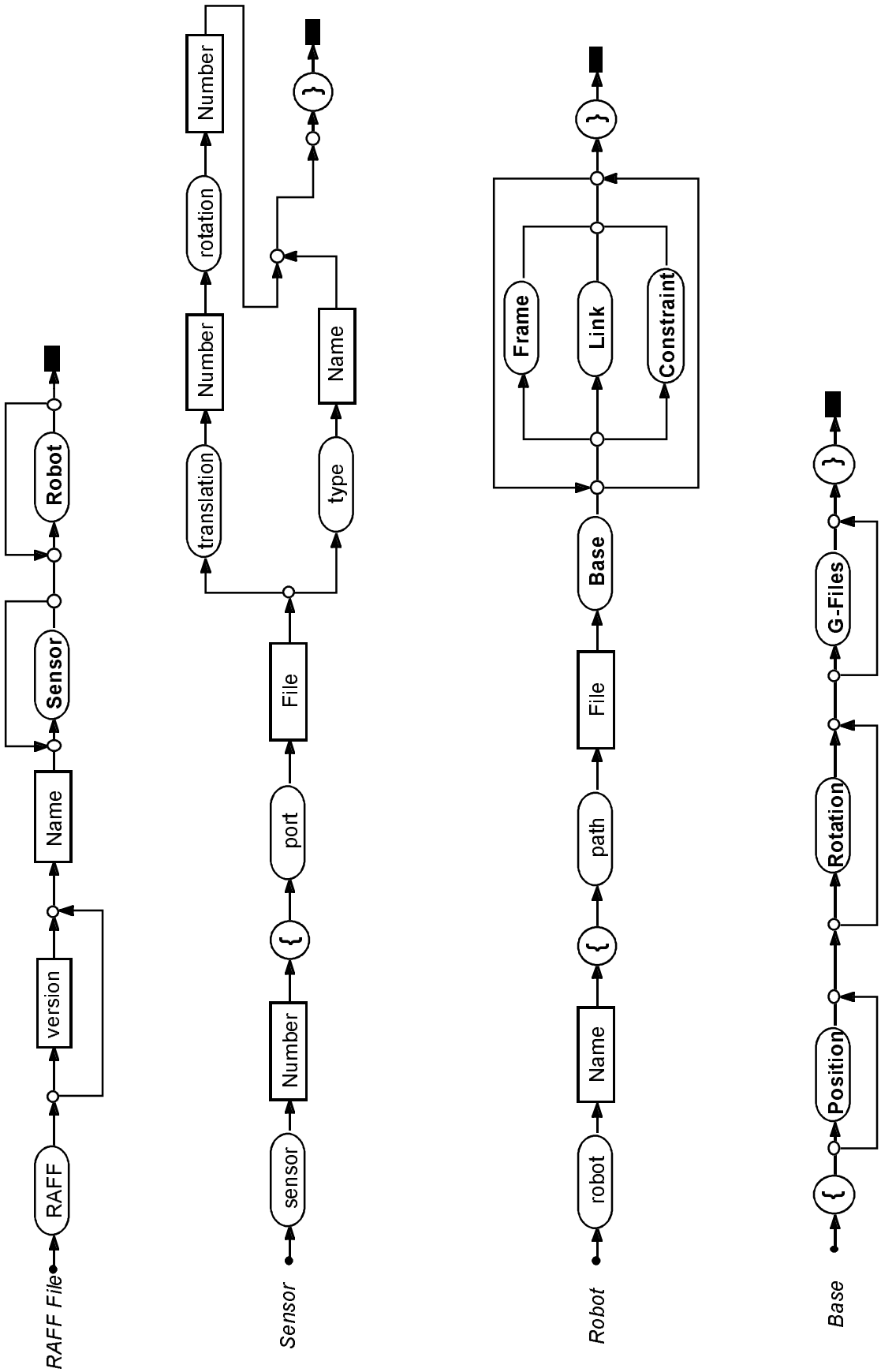



**Figure 7-3:** *Grammaire MAD*

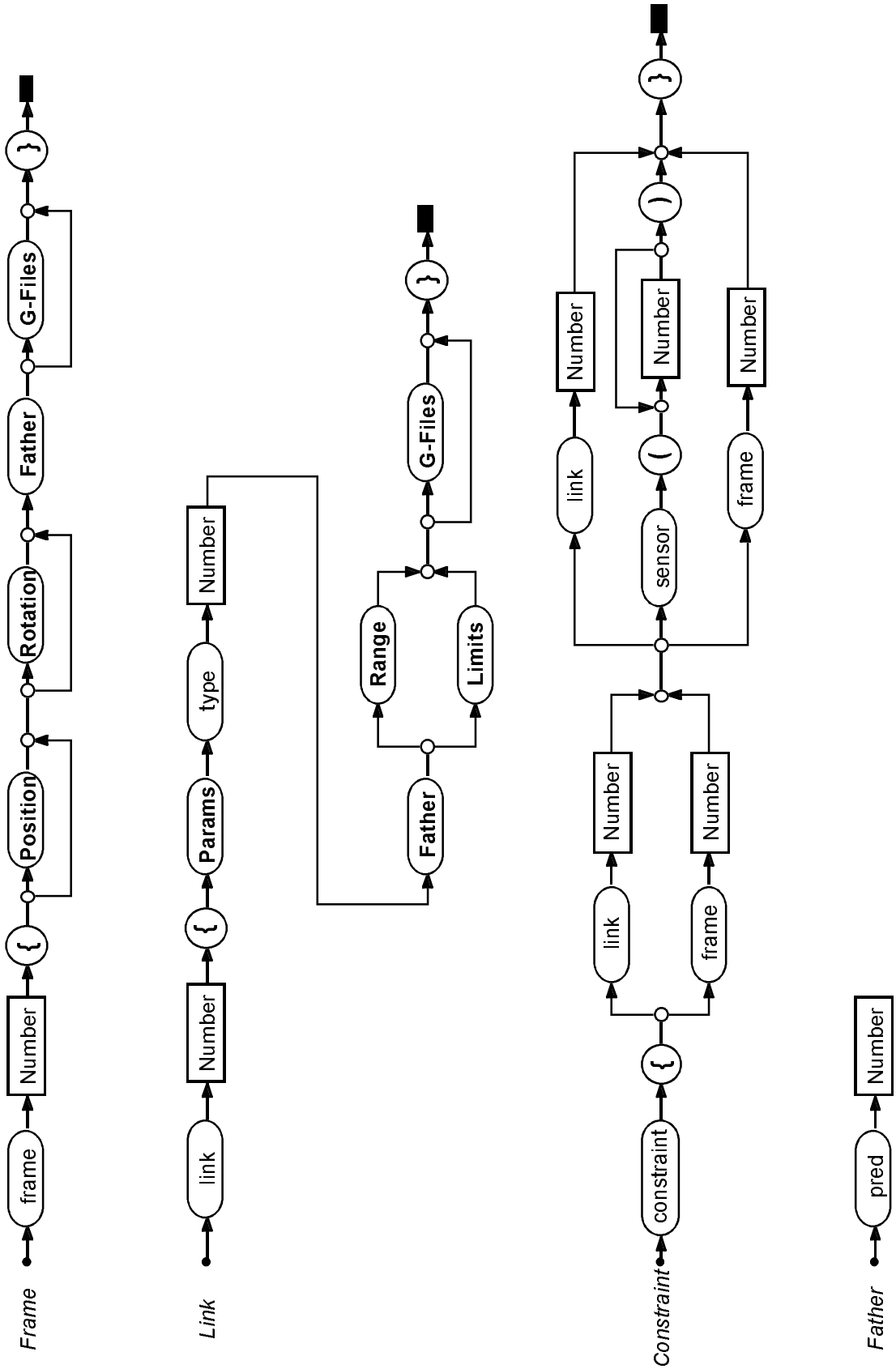



**Figure 7-3:** *Grammaire MAD*

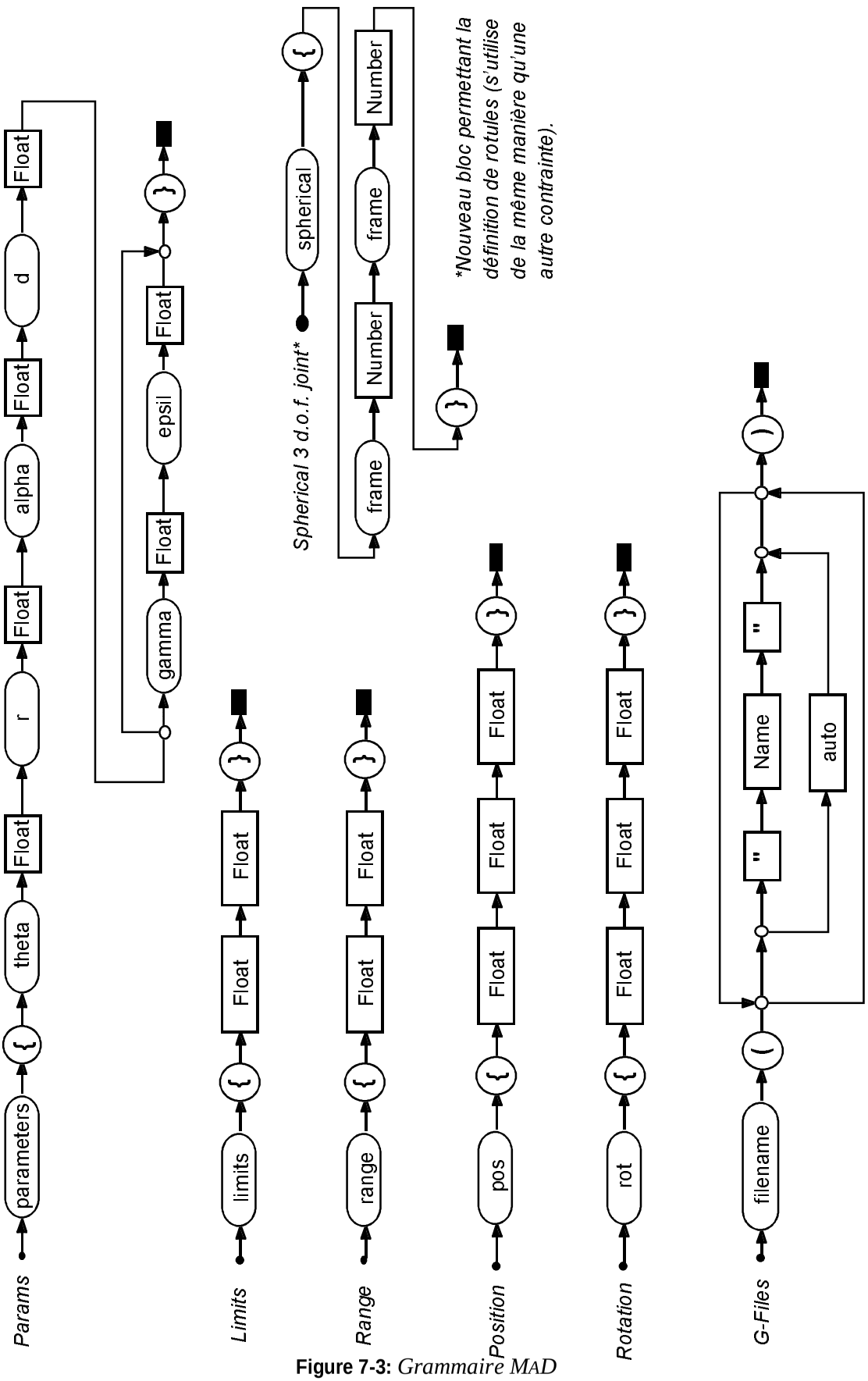



**Figure 7-3:** *Grammaire MAD*

ANNEXE

# B

# EXEMPLES DE FICHIERS MAD DE SIMULATIONS RÉALISÉES

Cette annexe fourni quelques exemples de fichiers MAD de simulations réalisées avec *VirtualRobot*. Pour des raisons évidentes de place, nous n'avons retenu que trois exemples significatifs.

- macdac.mad: robot série à 5 degrés de liberté présenté dans la Section 6.3.1. La cinématique inverse est contrôlée au niveau d'un référentiel supplémentaire attaché au dernier lien du manipulateur.
- cross.mad: robot parallèle à trois degrés de liberté possédant 15 articulations présenté dans la Section 6.3.3. Cet exemple montre comment sont modélisées des boucles cinématiques grâce à des contraintes. De plus, ce fichier utilise plusieurs référentiels supplémentaires pour placer certains objets graphique additionnels correctement.
- redont.mad: robot redondant de la Section 6.3.5. Il est intéressant dans cet exemple de noter l'utilisation du mot clef auto pour la représentation automatique des liens. Ce robot peut être contrôlé par deux senseurs différents afin de tirer profit de sa redondance. Le lien 2 utilise les 6 paramètres Kleinfinger-Khalil pour sa définition.

# macdac.mad

```
// MACDAC manipulator used a the IMG

RAFF MacDacArm

sensor 0 { }

sensor 1 {
  port “/dev/ttyd2”
  type spaceball
}

robot macdac {

  path
“/usr/people/lorenzo/GKG/Models/macdac/
”

  base {
    filename ( “joint_0.nff” )
  }

  link 1 {
    parameters {
      theta    0.0
      r        73.0
      d        0.0
      alpha    0.0
  }
    type 2
    pred 0
    limits { -60.0 60.0 }
    filename ( “joint_1.nff” )
  }

  link 2 {
    parameters {
      theta    150.0
      r        73.0
      d        122.0
      alpha    90.0
    }
    type 2
    pred 1
    limits { -90.0 150.0 }
    filename ( “joint_2.nff” )
  }

  link 3 {
    parameters {
      theta    -165.0
      r        -73.0
      d        398.0
      alpha    0.0
    }
    type 2
    pred 2
    limits { -165.0 10.0 }
    filename ( “joint_2.nff” )
  }

  link 4 {
    parameters {
      theta    255.0
      r        73.0
      d        398.0
      alpha      0.0
    }
    type 2
    pred 3
    limits { 0.0 255.0 }
    filename ( “joint_4.nff” )
  }

  link 5 {
    parameters {
      theta180.0
      r    172.0
      d    0.0
      alpha90.0
    }
    type 2
    pred 4
    limits { 90.0 270.0 }
    filename ( “joint_5.nff” )
  }

  frame 6 {
    pos { 200.0 0.0 0.0 }
    ori { 0.0 0.0 0.0 }
    pred 5
    filename ( “../repere-C.nff” )
  }

  constraint {
    frame 6
    sensor ( 0 1 )
  }

} // end of macdac
```

# cross.mad

```
RAFF v1.1 Delta_Lineaire_enCroix

// Description du CROSS-DELTA: robot
delta avec actionneurs
// lineaires places en croix.
// Ideal pour placer sous le fuselage
d'un avion...

// Note: les referentiels decrits au
debut possedent des numeros
// superieurs a ceux des liens pour ne
pas induire de confusion
// (démarage à 20)

sensor 0 { }

sensor 1 {
  port "/dev/ttyd2"
  type spaceball
}

robot crossdelta {

  path
"/usr/people/lorenzo/GKG/Models/cross-d
elta/"

  base {
  filename ( "../repere-M.nff" )
  }

  // referentiels speciaux pour placer
le fuselage et les rails
  frame 20 {
    pos { 16.0 0.0 0.0 }
    ori { 0.0 0.0 0.0 }
    pred 0
    filename ( "plane.nff"  )
  }
  frame 21 {
    pos { 0.0 0.0 1300.0 }
    ori { 0.0 0.0 0.0 }
    pred 0
    filename ( "rails.nff"  )
  }
  frame 22 {
    pos { 0.0 0.0 0.0 }
    ori { -90.0 0.0 0.0 }
    pred 0
  }
  frame 23 {
    pos { 0.0 0.0 -1250.0 }
    ori { 0.0 0.0 0.0 }
    pred 22
    filename ( "rails.nff"  )
  }
  frame 24 {
    pos { 0.0 0.0 0.0 }
    ori { 90.0 0.0 0.0 }
    pred 0
  }
  frame 25 {
    pos { 0.0 0.0 -1250.0 }
    ori { 0.0 0.0 0.0 }
    pred 24
    filename ( "rails.nff"  )
  }

  // h = 800
  // c = 200
  // e = 155.47

  // premier patin (1)
  link 1 {
    // l' = 1612.4515
    // al = 26.3878
    parameters {
      theta   0.0
      r       -1785.65
      d       0.0
      alpha   180.0
    }
    type 1
    pred 0
    limits { -2500.0 -100.0 }
    filename ( "patin.nff" )
  }

  // cardan superieur 1
  link 2 {
    parameters {
      theta   116.3878
      r       0.0
      d       0.0
      alpha   90.0
    }
    type 2
    pred 1
    limits { 90.5 160.0 }
    filename ( "cardan.nff" "joint.nff"
)
  }
  link 3 {
    parameters {
      theta   0.0
      r       0.0
      d       20.0
      alpha   90.0
```

```
    }
    type 2
    pred 2
    limits { -70.0 70.0 }
    filename ( “bras-1800.nff”
“embouts-e1800.nff” )
  }

    // cardan inferieur 1
    link 4 {
      parameters {
        theta    0.0
        r        0.0
        d        1800.0
        alpha    0.0
      }
    type 2
    pred 3
    limits { -70.0 70.0 }
    filename ( “cardan.nff” )
  }
  link 5 {
    parameters {
      theta    -116.3878
      r        0.0
      d        20.0
      alpha    -90.0
    }
    type 2
    pred 4
    range { -80.0 80.0 }
    filename ( “nacelle.nff”
“joint.nff” )
  }


  // patin gauche (2)
  link 6 {
    // l’ = 894.4272
    // al = 41.8103
    parameters {
      theta    0.0
      r        -994.43
      d        0.0
      alpha    90.0
    }
    type 1
    pred 0
    limits { -2500.0 800.0 }
    filename ( “patin.nff” )
  }

  // cardan superieur 2
    link 7 {
      parameters {
        theta    131.81
        r        0.0
        d        0.0
        alpha    90.0
      }
      type 2
      pred 6
      limits { 90.5 170.0 }
      filename ( “cardan.nff” “joint.nff”
)
    }
    link 8 {
      parameters {
        theta    0.0
        r        0.0
        d        20.0
        alpha    90.0
      }
      type 2
      pred 7
      range { -70.0 70.0 }
      filename ( “bras-1200.nff”
“embouts-e1200.nff” )
    }

    // cardan inferieur 2
    link 9 {
      parameters {
        theta    0.0
        r        0.0
        d        1200.0
        alpha    0.0
      }
      type 2
      pred 8
      range { -70.0 70.0 }
      filename ( “cardan.nff” )
    }
    link 10 {
      parameters {
        theta-131.81
        r    0.0
        d    20.0
        alpha-90.0
      }
      type 2
      pred 9
      range { -80.0 80.0 }
      filename ( “joint.nff” )
    }

    // positionnement de la fin de la
boucle gauche
    frame 16 {
      pos { 0.0 100.0 -170.0 }
      ori { 90.0 0.0 0.0 }
      pred 10
```

```
    filename ( )
  }


  // patin droit (3)
  link 11 {
    parameters {
      theta    0.0
      r        -994.43
      d        0.0
      alpha    -90.0
    }
    type 1
    pred 0
    limits { -2500.0 800.0 }
    filename ( “patin.nff” )
  }

// cardan superieur 3
  link 12 {
    parameters {
      theta    131.81
      r        0.0
      d        0.0
      alpha    90.0
    }
    type 2
    pred 11
    limits { 90.5 170.0 }
    filename ( “cardan.nff” “joint.nff”
)
  }
  link 13 {
    parameters {
      theta    0.0
      r        0.0
      d        20.0
      alpha    90.0
    }
    type 2
    pred 12
    range { -70.0 70.0 }
    filename ( “bras-1200.nff”
“embouts-e1200.nff” )
  }

  // cardan inferieur 3
  link 14 {
    parameters {
      theta    0.0
      r        0.0
      d        1200.0
      alpha    0.0
    }
    type 2
    pred 13
    range { -70.0 70.0 }
    filename ( “cardan.nff” )
  }
  link 15 {
    parameters {
      theta    -131.81
      r        0.0
      d        20.0
      alpha    -90.0
    }
    type 2
    pred 14
    range { -80.0 80.0 }
    filename ( “joint.nff” )
  }

  // positionnement de la fin de la
boucle droite
  frame 17 {
    pos { 0.0 100.0 170.0 }
    ori { -90.0 0.0 0.0 }
    pred 15
  filename ( )
  }


  // fermeture de la boucle gauche
  constraint {
    link 5
    frame 16
  }

  // fermeture de la boucle droite
  constraint {
    link 5
    frame 17
  }

  // contrainte pour cinematique
inverse sur nacelle
  constraint {
    link 5
    sensor ( 0 1 )
  }

} // fin de cross-delta
```

# redont.mad

```
RAFF simple

sensor 0 { }

// This simple redondant robot can be
controled
// by its inverse kinematics with the
spacemouse.
sensor 1 {
  port “/dev/ttyd2”
  type spaceball
}

// This control panel allow to move a
joint to
// fully exploit the redondany
sensor 2 {
  port "Mover"
  type panel
}

// this robot has 6 d.o.f. but 5 axes
are colinears
robot 6dof {

  path
“/usr/people/lorenzo/GKG/Models/”

  base {
    filename ( auto “obstacle.nff” )
  }

  link 1 {
    parameters {
      theta   90.0
      r       0.0
      alpha   0.0
      d       0.0
    }
    type 2
    pred 0
    range { -160.0 160.0 }
    filename ( “small-base.nff” )
  }

  //
  // this second link use the full six
  // khalil-kleinfinger parameters:
  // it’s a good example of how to
  // use this capability.
  //
  link 2 {
    parameters {
      theta   60.0
      r       0.0
      alpha   90.0
      d       0.0
      epsil   500.0
      gamma   0.0
    }
    type 2
    pred 1
    range { -110.0 110.0 }
    filename ( auto )
  }

  link 3 {
    parameters {
      theta   -30.0
      r       0.0
      alpha   0.0
      d       2000.0
    }
    type 2
    pred 2
    range { -110.0 110.0 }
    filename ( auto )
  }

  link 4 {
    parameters {
      theta   -30.0
      r       0.0
      alpha   0.0
      d       1500.0
    }
    type 2
    pred 3
    range { -110.0 110.0 }
    filename ( auto )
  }

  link 5 {
    parameters {
      theta   -30.0
      r       0.0
      alpha   0.0
      d       1000.0
    }
    type 2
    pred 4
    range { -110.0 110.0 }
    filename ( auto )
  }

  link 6 {
    parameters {
      theta   -30.0
```

```
      r        0.0
      alpha    0.0
      d        800.0
    }
    type 2
    pred 5
    range { -110.0 110.0 }
    filename ( auto )
  }

  frame 7 {
    pos { 800.0 0.0 0.0 }
    ori { 0.0 0.0 0.0 }
    pred 6
    filename ( “repere-B.nff” )
  }

  constraint {
    frame 7
    sensor ( 0 1 )
  }
  constraint {
    link 4
    sensor ( 0 2 )
  }

} // end of 6dof
```

ANNEXE

# C

# COMPARAISON DES GUI[1] POUR *VirtualRobot*

Le choix des langages et outils de programmation pour réaliser des interfaces graphiques utilisateur est très large. Il existe aussi bien des très bons outils développés par la communauté scientifique et donc gratuits, que des outils commerciaux. Les “packages” gratuits sont de qualité très variable et il est important de porter son choix sur un outil déjà largement utilisé afin de garantir la pérennité de l’application et une certaine fiabilité. Dans la gamme des produits commerciaux l’éventail reste large et le prix du “package” dépend souvent de la qualité de mise en oeuvre et de la portabilité de l’interface entre différentes plateformes.

Le Tableau C-1 présente une comparaison des GUI en fonction de critères spécifiques aux besoins du programme *VirtualRobot* (elle n’est donc pas applicable directement à d’autres applications). Les “packages” faisant l’objet de cette comparaison sont les suivants:

- Tcl/Tk, package bien connu du domaine public [wwwTCL];
- TIX, extension de Tcl/Tk [wwwTIX] (cf. Section 5.4.2);
- WTK GUI, fonctionnalité pour la création de GUI fournie par Sense8 avec la bibliothèque WorldToolKit [wwwSEN] (cf. Section 5.4.1);
- Motif/Xt, bibliothèque standard (versions commerciales et dérivées gratuite) [Young 95];
- ViewKit, bibliothèque développée par Silicon Graphics Inc. basée sur Motif et fournissant un accès plus facile et orienté objet;
- ViewKit+RapidApp, outil de développement d’interfaces proposé par Silicon Graphics Inc., il permet d’intégrer les fonctionnalités ViewKit et OpenInventor [wwwRAP];
- Java, comporte une bibliothèque portable largement reconnue pour réaliser des GUI.

1. GUI: Graphical User Interface, Interface Graphique Utilisateur.

**Tableau C-1:** *Evaluation des différents langages pour l'interface de VirtualRobot*

| | ***Tcl/-Tk*** | ***Tcl/Tk +TIX*** | ***WTK GUI*** | ***Motif/Xt*** | ***ViewKit*** | ***ViewKit+ RapidApp*** | ***Java*** |
|---|---|---|---|---|---|---|---|
| **Language** | *6* | *9* | *9* | *1* | *5* | *6* | *8* |
| Simplicité d'apprentissage | +++ | ++ | ++++ | 0 | + | + | ++ |
| Rapidité de développement | +++ | ++++ | ++++ | 0 | + | ++ | ++ |
| Approche OO du language | 0 | +++ | + | + | +++ | +++ | ++++ |
| **Possibilités** | *12* | *15* | *9* | *12* | *10* | *10* | *12* |
| Menus et boutons | +++ | ++++ | +++ | ++++ | ++++ | ++++ | ++++ |
| Sliders | ++++ | ++++ | ++++ | ++++ | ++++ | ++++ | ++++ |
| Graphs d'analyse | ++ | +++ | ++ | ++ | + | + | ++ |
| Autres Widgets / Extensions | +++ | ++++ | 0 | ++ | + | + | ++ |
| **Intégration** | *5* | *7* | *10* | *11* | *14* | *14* | *2* |
| Interfaçage avec C++ | + | ++ | ++ | +++ | ++++ | ++++ | -1 |
| Insertion dans l'applic WTK | + | + | ++++ | ++ | +++ | +++ | -1 |
| Rapidité de l'application | + | + | +++ | ++++ | +++ | +++ | ++ |
| "Look" de l'interface | ++ | +++ | + | ++ | ++++ | ++++ | ++ |
| **Portabilité** | *13* | *13* | *12* | *7* | *1* | *1* | *15* |
| Unix machines | ++++ | ++++ | ++++ | ++++ | + | + | ++++ |
| PC+WinNT | +++ | +++ | ++++ | + | 0 | 0 | ++++ |
| A travers le WEB | ++ | ++ | 0 | 0 | 0 | 0 | ++++ |
| Prix du pakage (pour "clients) | ++++ | ++++ | ++++ | ++ | 0 | 0 | +++ |
| **Total** | **36** | **44** | **40** | **31** | **30** | **31** | **37** |

ANNEXE

# D

# DOCUMENTATION DE *VirtualRobot*

La documentation de *VirtualRobot* consiste en un ensemble de pages au format HTML décrivant les différents objets du programme ainsi que leurs méthodes. Pour des raisons de place, il a finalement été décidé de ne pas publier cette documentation puisqu'elle est accessible par le World-Wide-Web.

On peut donc la trouver sur le serveur du Groupe VRAI accessible depuis le serveur du département de Microtechnique: *dmtwww.epfl.ch*

Ou alors de préférence directement à l'adresse suivante:
*http://imtsg7.epfl.ch/projects/cinegen/docs/Virob/VirtualRobot.html*

Le Tableau D-1 fournit la liste des différents fichiers composant le programme *VirtualRobot* ainsi que les objets principaux qui y sont décrits.

**Tableau D-1:** *Objets et fichiers composant VirtualRobot*

| Classe | Fichier entête | Fichier source | Remarque |
|---|---|---|---|
| | `Virtual-Robot.H` | `Virtua-Robot.C` | Programme principal |
| | | `raff.g` | description de la grammaire MAD |
| **`GKGRobot`** | `Robot.H` | `Robot.C` | robot manipulateur |
| **`GKGGraph`** | `Graph.H` | `Graph.C` | structure d'un robot |
| **`GKGGeom`** | `Geom.H` | `Geom.C` | objets graphiques |
| **`GKGNode`** | `Node.H` | `Node.C` | noeud générique composant un robot |
| **`GKGLink`** | `Link.H` | `Link.C` | lien d'un robot |
| **`GKGBase`** | `Base.H` | `Base.C` | base d'un robot |
| **`GKGFrame`** | `Frame.H` | `Frame.C` | référentiel d'un robot |
| **`GKGTracker`** | `Tracker.H` | `Tracker.C` | traqueur d'un robot |
| **`GKGSensor`** | `Sensor.H` | `Sensor.` | classe générique pour les senseurs |
| **`GKGScene`** | `Scene.H` | `Scene.C` | monde gérant tous les senseurs et robots |

**Tableau D-1:** *Objets et fichiers composant VirtualRobot*

| Classe | Fichier entête | Fichier source | Remarque |
|---|---|---|---|
| **GKGUniverse** | Universe.H | Universe. | univers graphique |
| **GKGWindow** | Window.H | Window.C | gestion des fenêtres |
| **GKGPhantom** | Phantom.H | Phantom.C | robot fantôme |
| **ObjTix** | ObjTix.H | ObjTix.C | interface avec les objets TIX |
| **TixInterp** | TixInterp.H | TixInterp.C | interface avec interpréteur TIX |
| | utils.H | utils.C | routines utilitaires |
| | convert.H | convert.C | routines de conversion |
| **XDRcomm** | XDRcomm.H | XDRcomm.C | gestion de la communication avec Viz |
| **ConstrTab** | Constraint.H | | stockage des contraintes |
| | GKGdefs.H | | définitions générales de *VirtualRobot* |
| **DLGifstream** | ifstrLexer.H | | gestion des fichiers d'entrée |
| **extANTLR-Token** | myToken.H | | gestion des tokens |

ANNEXE

# E

# INTÉGRATION DE LA GUI AVEC LA BOUCLE DE SIMULATION

## E.1 Approche par des phtreads

Une première approche pour l'intégration de la GUI avec la boucle de simulation a d'abord été implémentée en utilisant des "*thread*". En effet, la solution la plus élégante au problème posé consiste à séparer le processus de la boucle de simulation de celui de la boucle d'interface de Tcl/Tk. L'utilisation de "*pthread*" aux normes POSIX a permis la réalisation d'un programme principal lançant deux processus "légers": le premier étant la boucle de simulation 3D et le second la boucle événementielle de Tcl/Tk (cf Figure E-1). Des processus dis "légers" partagent la même zone de donnée en mémoire ce qui facilite l'échange de variables entre processus. Par exemple, la boucle de simulation ainsi qu'un élément d'interface Tk peuvent tous les deux modifier la position d'un objet, moyennant la précaution de ne pas faire des accès simultanés[2].

L'obstacle principal à cette approche est que Tcl/Tk n'est pas "thread safe"[3]. C'est à dire que l'appel d'une même fonction Tcl par deux processus différents risque de provoquer un arrêt critique du programme. Il est possible de contourner ce problème en utilisant des drapeaux supplémentaires pour obliger que ce soit toujours le même processus qui fasse des appels Tcl. Mais on rencontre alors de graves problèmes de blocages car la communication entre l'interface Tcl/Tk et la boucle de simulation doit être bidirectionnelle et parfois récursive: un changement dans l'interface impose une modification d'une variable dans la boucle de simulation qui va se répercuter par l'actualisation d'une valeur dans l'interface.

2. Ceci est réalisé en utilisant des mutex (drapeaux gérés très rapidement) qui permettent de bloquer l'accès à une variable par un processus. Tout autre processus voulant accéder à cette variable devra attendre que le premier processus libère ce mutex.
3. Des tentatives pour rendre Tcl "thread safe" sont entreprises, toutefois l'état actuel de ces projets ne permet pas leur utilisation dans l'environnement «CINEGEN» qui nécessite des outils stables et fiables.

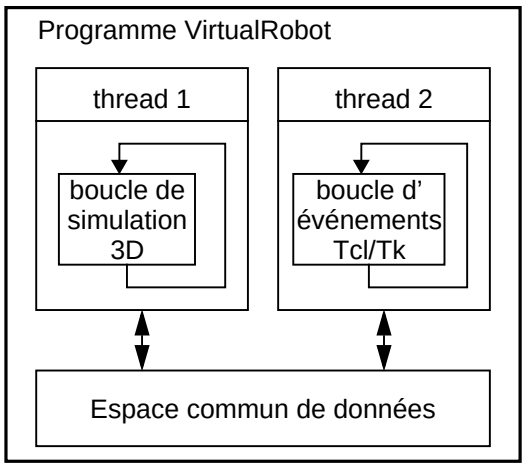



**Figure E-1:** *Intégration de l'interface Tcl/Tk avec des pthreads*

Si l'approche par pthreads semble intéressante à un niveau conceptuel, elle est complexe à mettre en oeuvre dans le cas de *VirtualRobot* qui implique des échanges bidirectionnels fortement couplés[4].

## E.2 Approche par gestion externe de boucle Tcl

Cette méthode est décrite dans la Section 5.5.4. Nous détaillons seulement ici les problèmes techniques de cette méthode qui consiste à éclater la boucle de gestion d'événements de Tcl. C'est alors *VirtualRobot* qui prend en charge la boucle de gestion des événements issus de Tcl.

Tcl propose deux modes de contrôle des événements:

- lecture normale des événements: la procédure reste en attente d'un événement, puis renvoie son code lorsqu'il survient. Ce modèle n'est pas utilisable dans notre cas car si aucun événement n'a lieu la boucle de simulation reste bloquée.
- lecture forcée des événements: la procédure essaye de lire le dernier événement de la queue et retourne un code même si aucun événement n'était présent. C'est le modèle utilisé pour lire les événements depuis *VirtualRobot*.

Le problème avec cette approche, c'est que la fréquence de la boucle de simulation est trop faible par rapport à celle nécessaire pour bien gérer les événements de l'interface (essentiellement la souris)[5]. C'est-à-dire que si l'on appelle la procédure de lecture d'un événement Tcl seulement à chaque passage dans la boucle de simulation, alors les réactions de l'interface Tcl deviennent trop lentes. Par exemple, les 15 ou 20Hz de rafraîchissement de l'environnement virtuel ne permettent pas de manipuler interactivement un ascenseur Tk: il se déplace par saccades.

4. Il faut encore noter que cette approche serait portable sur plateforme PC car Windows NT supporte les pthreads POSIX
5. D'où l'idée précédente d'isoler les deux boucles dans des processus différents fonctionnant chacun à leur rythme (pthread)!

Pour cette raison il a été décider de vider la queue des événements Tcl avant de rendre la main à la boucle de simulation principale. Toutefois cette idée (qui fonctionne) comporte un risque.

En effet, imaginons que l'utilisateur ne cesse de manipuler un ascenseur qui positionne un objet dans la scène graphique. Etant donné que le processus attend que la queue des événements soit vide avant de refaire une nouvelle boucle de simulation, il se pourrait que la procédure lisant ces événements garde constamment le contrôle: la position de l'objet graphique ne serait alors modifiée qu'une fois l'ascenseur relâché! Heureusement, Tcl est suffisamment rapide et "intelligent" pour rendre la main suffisamment souvent afin de permettre au programme principal de continuer à fonctionner correctement. Cette solution donne pleine satisfaction, même avec des simulations de robots complexes et de nombreux panneaux graphiques de contrôle ouverts.

ANNEXE

# F

# CARACTÉRISTIQUES DU SYNTAXEUR

Comme le décrit la Section 6.1.3 le syntaxeur proposé est composé d'une structure réalisant les translations basée sur le concept du robot delta, et une structure réalisant les rotations basée sur une structure parallèle articulée autour d'une rotule. Cette annexe résume les caractéristiques principales de ces deux structures telles qu'elle ont été réalisées.

## F.1 Structure DELTA du syntaxeur

La Figure F-1 représente le concept du robot DELTA tel qu'il est définit par Clavel [Clavel 91]. Une image tirée d'une simulation (réalisée avec *VirtualRobot*) montre (sur cette même figure) le DELTA tel qu'il est utilisé pour le syntaxeur.

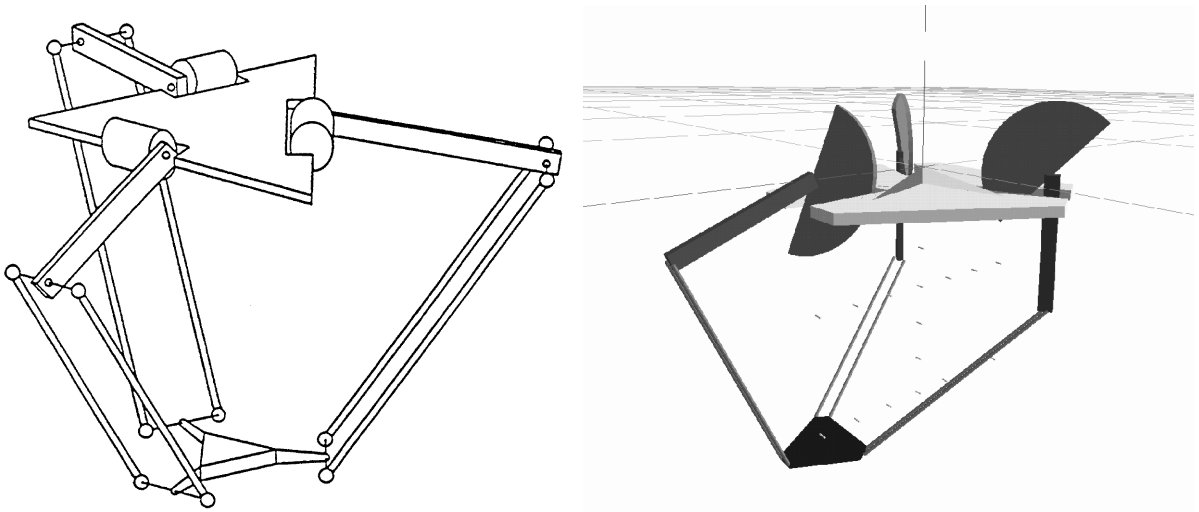


**Figure F-1:** *Structure DELTA et syntaxeur virtuel*

L'étude complète du mécanisme du syntaxeur (DELTA-syntaxeur) se trouve dans [Villard 97]. Le Tableau F-1 donne les dimensions qui caractérisent complètement le modèle géométrique du DELTA-syntaxeur.

**Tableau F-1:** *Caractéristiques dimensionnelles DELTA-syntaxeur*

| Description du paramètre | Nom† | Valeur (mm) |
|---|---|---|
| Distance entre le centre de la base fixe et l'axe de rotation du moteur | RA | 150 |
| Distance entre le centre de la nacelle et le côté du parallélogramme solidaire de la nacelle | RB | 50 |
| Longueur du bras | LA | 160 |
| Longueur d'une barre parallèle | LA | 265 |
| Ecartement de barres parallèles (distance entre deux rotules, pas significatif pour la cinématique) | | 60 |

†. Suivant les notations utilisées dans [Clavel 91].

Le Tableau F-2 donne les caractéristiques principales de l'entraînement du DELTA-syntaxeur.

**Tableau F-2:** *Caractéristiques de l'entraînement du syntaxeur*

| Paramètre | Valeur |
|---|---|
| Diamètre du secteur d'entraînement | 144mm |
| Diamètre du pignon moteur (extérieur) | 9mm |
| Rapport de transmission | 16 |
| Puissance nominale du moteur (Maxon RE-025) | 20W |

## F.2 Structure PARAMAT

Les principales caractéristiques de la structure PARAMAT du syntaxeur sont données dans la Section 6.1.3.

La Figure F-2 présente à dessin d'ensemble du mécanisme PARAMAT réalisé. Ce dessin fourni à titre illustratif (pas complet) est tiré de [Moser 98].

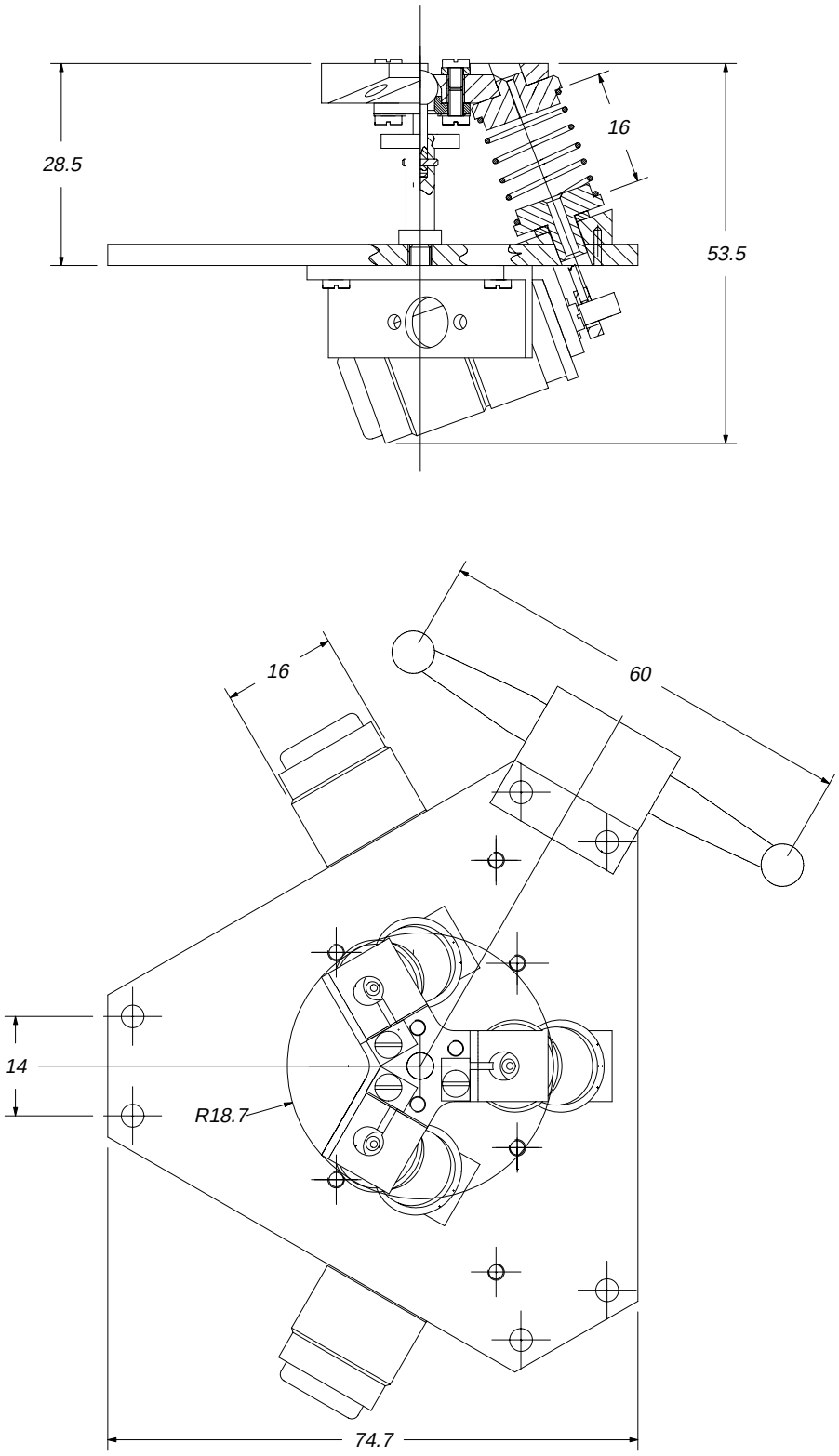



**Figure F-2:** *Vue globale du mécanisme PARAMAT*

# TABLE DES MATIÈRES

# LISTE DES FIGURES

# LISTE DES TABLEAUX

# Curriculum vitae

## Renseignement généraux

| | |
|---|---|
| *Nom* | Flückiger |
| *Prénom* | Lorenzo |
| *Date de naissance* | 23 mai 1970 |
| *Lieu d'origine* | Genève (GE) et Auswill (BE) |
| *Nationalité* | Suisse |

## Formation

| | |
|---|---|
| *1980-1988* | Baccalauréat Scientifique (type C)<br>Lycée Madame de Stäel (France, Haute Savoie)<br>Certificat de Maturité Fédérale (Genève, Suisse) |
| *1988-1994* | Diplôme d'ingénieur en Microtechnique<br>Ecole Polytechnique Fédérale de Lausanne, Suisse |
| *1994-1998* | Thèse de doctorat<br>Ecole Polytechnique Fédérale de Lausanne, Suisse |

## Publications


Flückiger, L. (1998). A robot interface using virtual reality and automatic kinematics generator. In *The 29th International Symposium on Robotics (ISR'98)*, Birmingham.

Flückiger, L., Baur, C., et Clavel, R. (1998). CINEGEN: a rapid prototyping tool for robot manipulators. In Schweitzer, G., Siegwart, R., et Cattin, P., editors, *The Fourth International Conference on Motion and Vibration Control (MOVIC'98)*, volume 1, pages 129–134, Zürich, Switzerland.

Flückiger, L., Piguet, L., et Baur, C. (1996). Generic robotic kinematic generator for virtual environment interfaces. In *SPIE Telemanipulator and Telepresence Technologies III*, volume 2901, pages 186–195, Boston.

Flückiger, L. (1994). Robotique, réalité virtuelle et vision. Rapport Technique IMT-94-01, EPFL (Ecole Polytechnique Fédérale de Lausanne).

Natonek, E., Zimmerman, T., et Flückiger, L. (1995). Model based vision as feedback for virtual reality. In *IEEE Virtual Reality Annual International Symposium*, pages 110–117, Los Alamitos, CA.

Natonek, E., Flückiger, L., Zimmerman, T., et Baur, C. (1994). Virtual reality: an intuitive approach to robotics. In *SPIE Telemanipulator and Telepresence Technologies*, volume 2351, pages 260–270, Boston.