Mémoire de stage

présenté à

l'Institut des Sciences et Techniques de Grenoble

par

Frédéric Arnaudon

3i 3ème année - Option système informatique

Conception d'actions robotiques sous ORCCAD

TOME **II**

INRIA Rhône-alpes
période du 2 Avril au 26 Septembre 1997

Remerciements

Je tiens tout d'abord à exprimer mes plus vifs remerciements à Monsieur Roger Pissard-Gibollet, ingénieur de recherche et chef du service robotique de l'INRIA Rhône-Alpes, pour m'avoir accueilli au sein du projet ORCCAD et qui m'a fait bénéficier de ses conseils et de son savoir pour mener à bien ce stage.

Je souhaite également remercier Monsieur Hervé Mathieu, ingénieur de recherche au service robotique, pour la gentilhesse avec laquelle il a bien voulu répondre à toutes mes questions.

Je les remercie également de l'ambiance chaleureuse qu'ils ont fait reigner tout au long des six mois que j'ai passé en leurs présences.

Remerciement aussi à Monsieur Konstantin Kappelos, ingénieur de recherche à l'unité de l'INRIA Sophia-Antipolis, pour l'aide précieuse qu'il m'a apporté.

Et enfin, je terminerai par une pensée à tous les gens de l'INRIA Rhône-Alpes que j'ai eu le plaîsir de rencontrer dans mon travail ou pendant les moments de détente (football, pétanque ...) que j'ai eu pendant ce stage.

Frédéric Arnaudon

Resumé

Le service robotique de l'INRIA Rhône-Alpes a pour mission la mise en oeuvre des systèmes robotiques nécessaires au projet de recherche en robotique et vision. Pour cela, il participe au développement d'un logiciel de contrôle-commande robotique. Ce logiciel, ORCCAD (*Open Robot Controller Computer Aided Design*) est un environnement logiciel qui a pour ambition de faciliter et de factoriser l'implémentation d'un contrôleur de robot. Par l'intermédiaire d'interfaces dédiés et en liaison avec des outils logiciels adaptés à chaque niveau de spécification, ORCCAD offre une solution complète, allant de la spécification de haut niveau à la génération du code exécutable temps-réel.

Le but de ce stage est d'intégrer au logiciel une hiérarchie dite spécialisée pour la commande des robots manipulateurs. Cette hiérarchie regroupe l'ensemble des algorithmes permettant de contrôler et de commander ces robots.

Le travail demandé pendant le stage se décompose en deux parties, la première s'inscrit dans une phase d'utilisation du logiciel ORCCAD et la seconde dans une phase de participation au développement. Le tome 1 du rapport de stage décrit la prise de contact avec l'environnement de programmation ORCCAD et un exemple d'implémentation d'une procédure pour un robot pendule. La tome 2 détaille les algorithmes de la hiérarchie spécialisée pour bras manipulateur et décrit mon travail de programmation.

Abstract

The robotic department of INRIA Rhône-Alpes takes part in the development of a software environment dedicated to the design and the implementation of robotic control systems. This software is called ORCCAD : Open Robot Controller Computer Aided Design.

This document deals with the techniques used to add a new software tool to ORCCAD allowing the automatic generation of ORCCAD module tasks. This tool will bring together all algorithms relating to manipulator arms on a hierarchical view so as to propose a set of pre-programmed algorithms to make the implementation of robotic missions easier.

The first chapter focuses on the hierarchy specialised in manipulator robot command and on the algorithms implemented in this hierarchy. The next two chapters describe the methods used to realize these implementations and the ways to use graphic interfaces I created during this period.

Some results are illustrated with graphic examples in annex.

Table des matières

Introduction générale

Longtemps utilisées pour des tâches manufacturières répétitives, les robots[1] sont en passe aujourd'hui d'être reconnus comme indispensables au bien-être de l'homme. Utilisés notamment en milieu hostile ou dans des tâches d'assistance à un opérateur humain, les applications de la robotique moderne connaissent un second souffle grâce à des systèmes autonome comme *sojourner*, le véhicule planétaire explorateur de la surface du sol martien ou encore plus récement avec le *P2*, le robot bipède dernier né de la firme japonaise *Honda*.

Ces systèmes, qui ont largement profité de l'accroissement des performances technologique, sont désormais capable de réagir partiellement à leur environnement. Mais qu'ils soient manufacturier ou non, ils sont généralement pilotés par une architecture informatique complexe : *le contrôleur de robot.*

L'INRIA (Institut National de Recherche en Informatique et en Automatique), dont bon nombre de recherche sont basées sur des applications robotiques, travaille depuis plusieurs années au développement d'un logiciel de contrôle-commande robotique. Le logiciel, ORCCAD (Open Robot Controller Computer Aided Design), est un environnement logiciel facilitant la conception et la mise en oeuvre de contrôleur de robot. Cette conception va de la spécification à la génération de code en passant par la validation des missions à réaliser par le système.

Il dispose pour cela de nombreux outils logiciels lui permettant entre autre (cf. tome 1 du rapport de stage) :

- de construire graphiquement des applications robotiques, avec plusieurs niveaux d'abstractions ;
- d'effectuer des vérifications et des simulations sur ces applications ;
- et de générer du code C++/temps-réel exécutable destiné au système de commande du robot.

La présentation et le mode d'utilisation du logiciel ORCCAD sont détaillés dans le tome 1 du rapport de stage, je renvoie donc le lecteur à ce document. La partie du sujet du stage traitée ici s'inscrit dans une phase de développement logiciel. Il s'agit en fait d'intégrer dans ORCCAD une hiérarchie spécialisée pour la commande des bras manipulateurs. Cette hiérarchie spécialisée est le résultat d'une recherche effectuée au sein de l'institut, elle est présentée dans le chapitre 2.

Le présent document est décomposé en quatres chapitres. Certain termes techniques appartenant à la robotique sont expliqués dans l'annexe 1. La bonne compréhension de ce document passe par une lecture préalable du tome 1 du rapport de stage.

- **Chapitre 1 :** ce chapitre introduit la hiérarchie spécialisée pour la commande des bras manipulateurs et les algorithmes implémentés dans cette hiérarchie. Il donne également des indications sur le rôle des hiérarchies spécialisées dans ORCCAD et les avantages et inconvénients des algorithmes ;

1. robot : du tchèque robota (corvée) introduit pour la première fois dans la littérature par Karel Capek en 1920

- **Chapitre 2 :** le but de ce chapitre est deprésenter quelques précisions techniques sur l'implémentation logicielle ;
- **Chapitre 3 :** ce chapitre est le manuel utilisateur de la partie logicielle que j'ai développée ;
- **Chapitre 3 :** une partie économique.

Ce document contient également un certain nombre d'annexes qui illustrent le travail effectué pendant le stage, il possède notamment un ensemble de résultats expérimentaux.

Chapitre 1

Spécification

1.1 Introduction

Dans ORCCAD, la spécification d'une application robotique se basée sur trois entités abstraites,le module, la Tâche-Robot **TR** et la Procédure-Robot **PrR**. Ces trois entités sont détaillées dans le tome 1 du rapport de stage. Certains domaines d'application possèdent un grand nombre d'algorithmes établis qu'il semble profitable de représenter sous une vue hiérarchique. Nous verrons ici que les hiérarchies spécialisées sont disponibles au niveau de la création des modules, leur but est d'intégrer, dans le logiciel, les connaissances des domaines auxquels elles sont dédiées, ceci afin de proposer un outils supplémentaire facilitant l'implémentation d'une procédure-robot.

L'objet de ce chapitre est de présenter brièvement la hiérarchie spécialisée pour les robots de type bras manipulateur. Nous décrirons plus précisément les algorithmes implémentés dans cette hiérarchie pour les générateurs de trajectoire et pour les ressources physiques.

1.2 Hiérarchie spécialisée pour robot manipulateur

Le logiciel ORCCAD compte couvrir les domaines d'application suivant :

- bras manipulateurs,
- robots mobiles.

L'objectif qu'il m'a été donné est de réaliser la mise en oeuvre de la hiérarchie pour les robots manipulateur, je vais donc la présenter dans ce paragraphe.

Les hiérarchies spécialisées appartiennent au niveau de spécification le plus bas, c'est à dire au niveau Module. Leur but est de mettre à disposition de l'utilisateur un outils permettant de générer automatiquement des modules dédiés à une catégorie de robot. Ces modules reprennent un ensemble d'algorithmes dont le choix est proposé à l'utilisateur.

L'étude de la hiérarchie spécialisée pour les robots manipulateurs (cf. [7]) est détaillé ici, elle se décompose en 7 parties :

- **PR :** Ressource Physique ;
- **TG :** Génération de Trajectoire ;

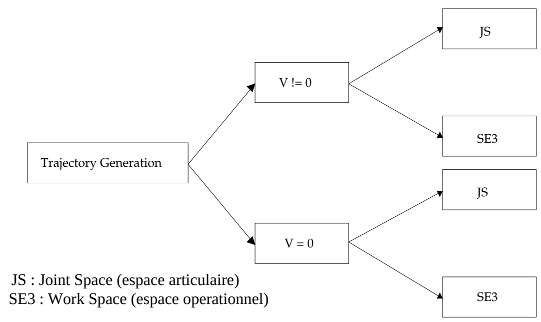

FIG. 1.1 – *Arborescence de la hiérarchie TG.*

- **TF :** Fonction de Tâche ;
- **MOD :** MODélisation ;
- **CO :** COntrôle ;
- **OBS :** OBSservateur ;
- **ATR :** Automate de la Tâche Robot.

1.2.1 Ressource physique (PR)

Cette partie est utilisée pour la saisie des caractéristiques physique du robot. Elle n'implémente aucun algorithme. Son rôle est de décrire les caractéristiques physiques du bras manipulateur. Ces caractéristiques sont :

- nombre de degré de liberté (ou nombre d'articulation du robot) ;
- paramètres de DH : description géométrique du robot ;
- repère de base et repère atelier ;
- masse, matrice d'inertie et repère de la matrice d'inertie ;
- valeurs limites (position, vitesse, accélération et courant).

stocker les données du robot sous un format bien précis afin de pouvoir les ré-utiliser ultérieurement sans avoir à les saisir à nouveau.

1.2.2 Génération de trajectoire (TG)

C'est l'une des trois parties qui possède une arborescence, elle est représentée figure 1.1. Son rôle est de créer des modules générateur de trajectoire qui calculent des consignes pour les autres modules de la tâche-robot. Nous avons modifié le contenu des deux noeuds intermédiaires (par rapport à l'arborescence présenté dans le tome 1 et dans [7]) et il est maintenant possible de générer une trajectoire passant par plusieurs points avec soit une vitesse nulle sur ces points (V=0), soit une vitesse non nulle (V!=0).

Cette partie propose le choix entre 9 algorithmes différents (5 à vitesse nulle sur les points de passage et 4 à vitesse non nulle), ils sont tous adaptés à l'espace articulaire et à l'espace opérationnel et sont décrits au paragraphe 1.3.

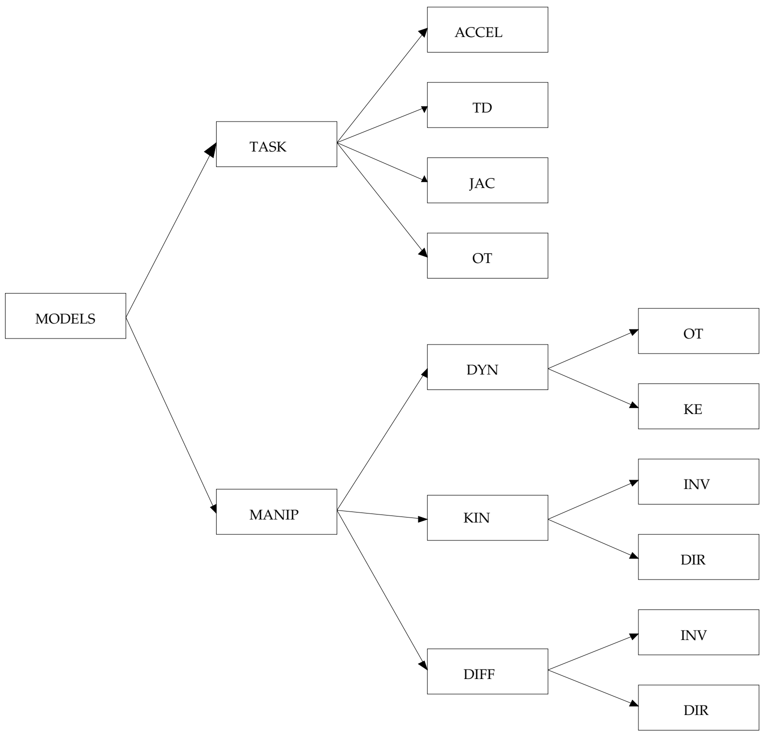

FIG. 1.2 – *Arborescence de la hiérarchie MOD.*

1.2.3 Fonction tâche (TF)

La fonction de tâche ($e(q,t)$) est la fonction d'erreur à reguler suivant la tâche robotique à réaliser (cf. [11]). Les exemples classiques de fonctions de tâches pour les bras manipulateurs sont :

- la commande en position articulaire sur une trajectoire désirée q_d. On choisit alors : $e(q,t) = q - q_d(t)$;
- la commande en position cartésienne (ou dans l'espace opérationnel) sur une trajectoire de position x_d. La fonction de tâche est de la forme : $e(q,t) = x(q) - x_d(t)$.

Dans [11], il est montré que satisfaire une tâche peut être exprimé comme une régulation à zéro de $e(q,t)(\forall t \in [0,T])$.

1.2.4 Modèle (MOD)

Cette partie possède l'arborescence la plus complexe. Les entités modélisées (figure 1.2) concernent le manipulateur et la fonction de tâche. Les sous parties MOD.KIN et MOD.DIF évaluent respectivement les modèles cinématique et difféntiel direct et inverse du manipulateur. Les sous parties MOD.DYN et MOD.TF évaluent les modèles de la matrice d'inertie (Jacobienne ..).

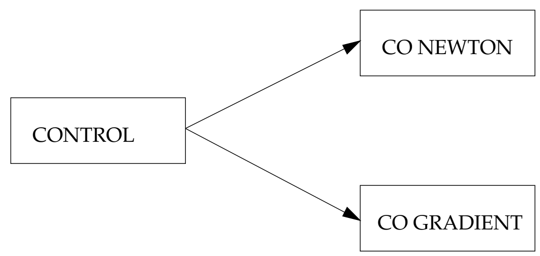

FIG. 1.3 – *Arborescence de la hiérarchie CO.*

1.2.5 Contrôle (CO)

C'est la troisième partie qui possède une arborescence (figure 1.3). Son rôle est de calculer le vecteur des couples à appliquer aux actionneurs suivant le type de contrôle adopté (Gradient ou Newton).

1.2.6 Observateur (OBS)

Elle concerne l'observation de l'environnement et la surveillance de la bonne exécution de la tâche. Les observateurs détectent par exemple le dépassement d'un seuil par la norme de l'erreur, l'entrée en configuration singulière, l'entrée en butée mécanique.

1.2.7 Automate de la tâche robot (ATR)

C'est une partie essentielle dans le modèle proposé. Son rôle est de ryth mer l'évolution du comportement de la Tâche-Robot. Elle est caractérisée par l'ensemble des événements, leur traitement et la durée de la Tâche-Robot.

1.3 Description des algorithmes de génération de trajectoire

Ce paragraphe contient une présentation exhaustive des algorithmes de génération de trajectoire pour les bras manipulateurs (cf. [3]). Ces algorithmes se décomposent en deux sous-groupes princpaux :

- trajectoire entre deux points, ce sont les mêmes algorithmes que ceux avec vitesse nulle au point de passage. Nous proposons ici la possibilité d'enchaîner plusieurs trajectoires entre deux points. Le mouvement résultant est alors constitué d'une suite de mouvement simple entre deux points avec arrêt sur les points intermédiaires.

- trajectoire avec points intermédiaires, ce sont les mêmes algorithmes que ceux avec vitesse non nulle au point de passage. Ici, chaque segment de trajectoire est interpolé par un polynôme de degré 3 ou 5 et la vitesse aux points intermédiaires est calculée suivant différentes méthodes (paragraphe 1.3.2) de facon á respecter certaines conditions de continuité en vitesse ou en accélération.

Les algorithmes sont présentés dans l'espace articulaire, ils peuvent facilement être portés dans l'espace opérationnel, c'est l'objet du paragraphe 1.3.3.

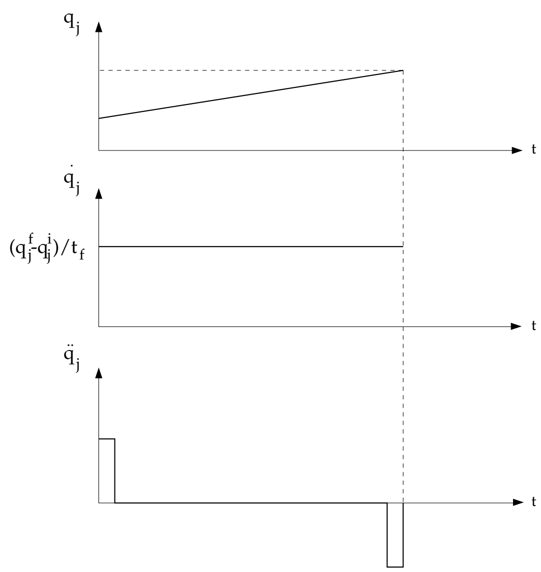

FIG. 1.4 – *Interpolation lináire sur une articulation j.*

1.3.1 Génération de mouvement entre deux points

On considère un robot à n degrés de liberté. Soit q^i et q^f les configurations articulaires initiales et finales. On désigne respectivement par k_v et k_a les vitesse et accélération maximales.

Le mouvement, interpolé entre q^i et q^f en fonctions du temps t, est décrit par l'équation :

$$q(t) = q^i + r(t)D \tag{1.1}$$

avec : $D = q^f - q^i$ et $0 \leq t \leq t_f$

Plusieurs fonctions d'interpolations permettent de satisfaire le passage par q^i à t = 0 et par q^f à $t = t_f$. Les plus utilisées en robotique sont l'interpolation polynômiale, la loi Bang-Bang et la loi Bang-Bang avec palier de vitesse ou loi trapèze. Elles sont présentées dans les paragraphes suivants.

Interpolation polynômiale

Les modes d'interpolation polynômiale les plus fréquement rencontrés sont l'interpolation linéaire et l'interpolation par des polynômes de degrés trois ou cinq.

a) Interpolation linéaire

Il s'agit de l'interpolation la plus simple : le mouvement de chaque articulation est décrit par une équation linéaire en temps. L'équation du mouvement s'écrit :

$$q(t) = q^i + \frac{t}{t_f} D$$

Cette loi de mouvement, qui impose une vitesse constante le long de la trajectoire, est continue en position (figure 1.4) mais discontinue en vitesse et en accélération. Pratiquement, le début et la fin du mouvement sont marqués par des à-coups.

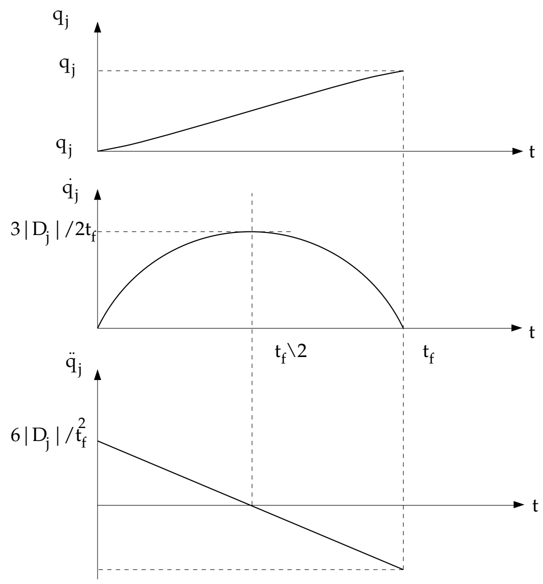

FIG. 1.5 – *Loi polynômiale de degré trois.*

b) Polynôme de degré trois

Si l'on impose une vitesse nulle aux points de départ et d'arrivée, on ajoute deux contraintes supplémentaire aux deux contraintes de position de l'interpolation linéaire. Le degré minimal du polynôme qui satisfait ces quatres contraintes est de degré trois et a pour forme générale :

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

avec comme conditions initiales :

$$\begin{cases} q(0) = q^i \\ q(t_f) = q^f \\ \dot{q}(0) = 0 \\ \dot{q}(t_f) = 0 \end{cases}$$

La résolution du système ainsi obtenu nous donne l'expression des coefficients :

$$\begin{cases} a_0 = q^i \\ a_1 = 0 \\ a_2 = \frac{3}{t_f^2} D \\ a_3 = -\frac{2}{t_f^3} D \end{cases}$$

La figure 1.5 donne, pour l'articulation j, l'évolution des positions, vitesses et accélérations. Cette loi de mouvement assure la continuité des vitesses mais pas celle des accélérations. En pratique, les robots industriels sont suffisament rigides pour que cette discontinuité soit filtrée par la mécanique. L'utilisation d'un polynôme de degré trois est satisfaisante dans la plupart des applications.

c) Polynôme de degré cinq

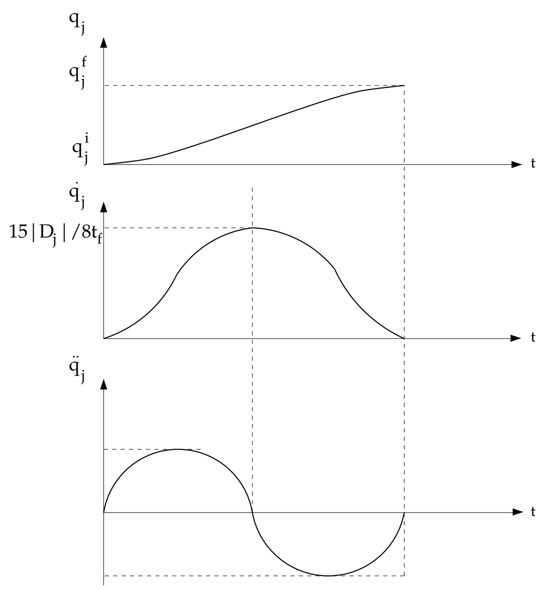

FIG. 1.6 – *Loi polynômiale de degré cinq.*

Si l'on veut éliminer la discontinuité sur les accélérations, il faut ajouter deux nouvelles contraintes :

$$\begin{cases} \ddot{q}(0) = 0 \\ \ddot{q}(t_f) = 0 \end{cases}$$

Le polynôme d'interpolation est donc de degré cinq, il se met sous la forme :

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

Le calcul des coefficients nous donne :

$$\begin{cases} a_0 = q^i \\ a_1 = 0 \\ a_2 = 0 \\ a_3 = \frac{10}{t_f^3} D \\ a_4 = \frac{-15}{t_f^4} D \\ a_5 = \frac{6}{t_f^5} D \end{cases}$$

Les évolutions des positions, vitesses et accélérations pour l'articulation j sont données figure 1.6.

La loi Bang-Bang

Le mouvement est composé dans ce cas d'une phase d'accélération constante jusqu'à $t_f/2$ puis d'une phase de décélération constante (figure 1.7). Les vitesses de départ et d'arrivée sont nulles. Le mouvement est donc continu en vitesse et en position, mais discontinu en accélération.

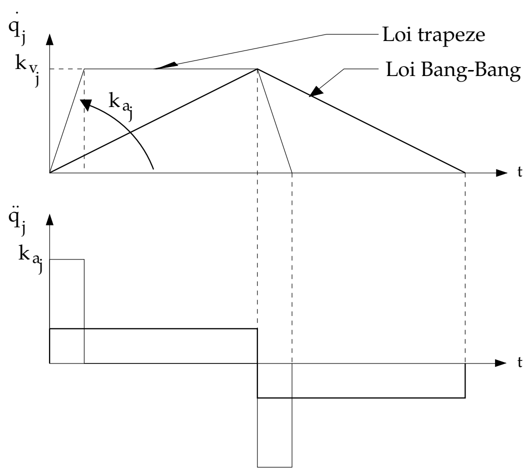

FIG. 1.7 – *Loi Bang-Bang et loi trapèze.*

La position est donnée par :

$$\begin{cases} q(t) = q^i + 2(\frac{t}{t_f})^2 D & pour\ 0 \leq t \leq \frac{t_f}{2} \\ q(t) = q^i + [-1 + \frac{4t}{t_f} - 2(\frac{t}{t_f})^2]D & pour\ \frac{t_f}{2} \leq t \leq t_f \end{cases}$$

La loi trapèze

La loi trapèze reprend le principe de la loi Bang-Bang mais en ajoutant un palier de vitesse pendant un temps τ. Ce palier de vitesse permet de saturer l'accélération et ainsi de diminuer le temps de parcours (figure 1.7). La loi trapèze est la loi optimale en temps parmi celles qui assurent la continuité en vitesse. Le mouvement de l'articulation j est représenté par les relations suivantes :

$$\begin{cases} q_j(t) = q_j^i + 2\frac{t^2}{2})k_{a_j}Sign(D_j) & pour\ 0 \leq t \leq \tau_j \\ q_j(t) = q_j^i + (t - \frac{\tau_j}{2})k_{v_j}Sign(D_j) & pour\ \tau \leq t \leq t_{f_j} - \tau_j \\ q_j(t) = q_j^f + (t_{f_i} - t)^2 K_{a_j}Sign(D_j) & pour\ t_{f_j} - \tau_j \leq t \leq t_{f_j} \end{cases} \tag{1.2}$$

Afin de synchroniser les phases d'accélération et de freinage de toutes les articulations, durée τ_j identique pour toutes les articulations, on utilise ici une méthode proportionnelle. Selon cette méthode, les lois de vitesses des diverses articulations sont homothétiques et comportent des phases d'accélération et de freinage de mêmes durée τ. On en déduit alors les relations :

$$\begin{cases} \dot{q}_j(t) = \lambda_j \dot{q}_k(t) \\ \ddot{q}_j(t) = \nu_j \ddot{q}_k(t) \end{cases} \quad \text{pour } j = 1, ..., n \text{ n représente le nombre de degrés de liberté du robot considéré}$$

où $\dot{q}_k$ est la vitesse de l'articulation contraignante, c'est à dire la vitesse de l'articulation qui doit parcourir le plus de chemin dans le même temps que les autres et $\dot{q}_j(t)$, la vitesse, à calculer, de l'articulation j (idem pour les articulations).

Les coefficients λ et ν se calculent de la facon suivante :

$$\begin{cases} \lambda_{i_{opt}} = MIN[1, \frac{k_{v_j}|D_i|}{k_{v_i}|D_j|}] \\ \nu_{i_{opt}} = MIN[1, \frac{k_{a_j}|D_i|}{k_{a_i}|D_j|}] \end{cases} \quad i = 1, ..., n;\ j = 1, ..., n;\ i \neq j$$

La fait de synchroniser le mouvement des articulations implique de remplacer dans 1.2 les valeurs des vitesses $\dot{q}_j$ et des accélérations $\ddot{q}_j$ par leurs valeurs synchronisées soit respectivement $\lambda_j \dot{q}_j$ et $\nu_j \ddot{q}_j$.

1.3.2 Génération de mouvement avec points intermédiaires

Ce paragraphe aborde la problème de la génération de mouvement dans le cas où la trajectoire est contrainte de passer, sur ou au voisinage d'un certain nombre de points intermédiaires. Le rôle de ces points est de déformer la trajectoire pour éviter les collisions entre le robot et son environnement. Imposer une vitesse non nulle sur ces points permet de diminuer le temps de parcour total et de lisser la trajectoire.

Interpolation linéaire et transition parabolique

Cet algorithme construit un mouvement composé de parties linéaires reliées par des transitions paraboliques. La particularité est que la trajectoire ne passe pas sur les points intermédiaires mais à proximité.

- En ce qui concerne la partie linéaire, on procède comme au paragraphe 1.3.1 a). Le mouvement entre τ_{k-1} et $t_k - \tau_k$ (figure 1.8) est défini par une vitesse constante telle que :

$$\dot{q}_j^k = \frac{D_j^k}{t_k}$$

La position articulaire évolue alors selon la relation :

$$q_j^k = t\dot{q}_j^k$$

- En ce qui concerne la transition sur le k^{ieme} point de la trajectoire, en supposant que t = 0 lors du passage au voisinage du point q_j^k, elle commence τ_k secondes avant et continue τ_k secondes après cet instant.

Les conditions de continuité en position et en vitesse aux extrémités de la phase de transition s'écrivent :

$$\begin{cases} q(-\tau_k) = q^k - \frac{\tau_k}{t_k} D^k \\ q(\tau_k) = q^k + \frac{\tau_k}{t_{k+1}} D^k \\ \dot{q}(-\tau_k) = \frac{D^k}{t_k} \\ \dot{q}(\tau_k) = \frac{D^{k+1}}{t_{k+1}} \end{cases}$$

Ces quatres contraintes peuvent être décritent par un polynôme de degré deux. Le calcul des coefficients nous donne l'équation du mouvement :

$$q(t) = q^k - \frac{D^k}{4\tau_k t_k}(t - \tau_k)^2 + \frac{D^{k+1}}{a\tau_k t_{k+1}}(t + \tau_k)^2$$

Ce mouvement est continu en position et en vitesse. Pour avoir un mouvement continu en accélération, on peut utiliser une fonction du cinquième degré pour la transition, les accélérations initiale et finale étant alors égales à zéro.

Remarque : Si l'on désire passer rigoureusement sur un point intermédiaire, il suffit d'ajouter après ce point, un point de même valeur.

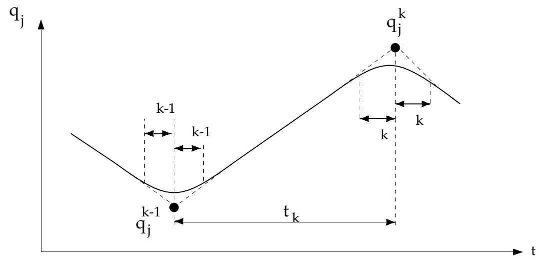

FIG. 1.8 – *Interpolation linéaire et transition parabolique.*

Utilisation de polynôme du troisième degré

Dans les méthodes présentées ci-dessous, le mouvement entre deux points q^{k-1} et q^k est décrit par un polynôme du troisième ou cinquième degré. Contrairement à la précédente, ces méthodes engendrent des trajectoires passant exactement sur les points intermédiaires.

Soit t_k le temps de parcours entre les points q^{k-1} et q^k, les coefficients du polynôme se déduisent facilement des conditions aux limites :

$$\begin{cases} q(0) = q^{k-1} \\ q(t_k) = q^k \\ \dot{q}(0) = \dot{q}^{k-1} \\ \dot{q}(t_k) = \dot{q}^k \end{cases}$$

On obtient pour les coefficients, le système suivant, on remarque qu'ils dépendent des vitesses au points intermd́iaires :

$$\begin{bmatrix} a_{0,k} \\ a_{1,k} \\ a_{2,k} \\ a_{3,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-3}{t_k^2} & \frac{3}{t_k^2} & \frac{-2}{t_k} & \frac{-1}{t_k} \\ \frac{2}{t_k^3} & \frac{-2}{t_k^3} & \frac{1}{t_k^2} & \frac{1}{t_k^2} \end{bmatrix} \begin{bmatrix} q^{k-1} \\ q^k \\ \dot{q}^{k-1} \\ \dot{q}^k \end{bmatrix} \tag{1.3}$$

Remarque : Le calcul des coefficients d'un polynôme du cinquième degré s'effectue en ajoutant deux nouvelles contraintes sur les accélérations. On suppose alors que celle-ci sont nulle sur les points intermédiaires. Cette solution assure une continuité sur les accélérations.

a) Vitesse calculée de facon locale

On remplace dans l'équation matricielle 1.3 les vitesses par la pente moyenne sur les intervalles (t_k, t_{k+1}), qui s'écrit :

$$\dot{q}^k = \frac{q^{k+1} - q^{k-1}}{t_k + t_{k+1}}$$

Toutefois attention, cette méthode ne convient pas lorsque deux portions de trajectoire consécutives sont de longueur très inégales, la pente moyenne conduit alors à des vitesses trop élevées et peut amener des dépassement de vitesse. Le calcul des vitesses de facon locale s'applique également pour des polynômes de degré cinq.

b) Utilisation des fonctions splines

Ici, comme dans le paragraphe précédent, chaque segment de la trajectoire est représentés par une fonction cubique sauf le premier et le dernier qui nécéssitent une fonction du quatrième degré. Les fonctions splines s'écrivent :

$F_k = q^k(t) = a_{0,k} + a_{1,k}t + a_{2,k}t^2 + a_{3,k}t^3$
$F_{k+1} = q^{k+1}(t) = a_{0,k+1} + a_{1,k+1}t + a_{2,k+1}t^2 + a_{3,k+1}t^3$

Le calcul des vitesses s'effectue en imposant une condition de continuité sur les accélérations.

$\ddot{F}_k(t_k) = \ddot{F}_{k+1}(0)$

on obtient alors l'égalité suivante :

$$\begin{bmatrix} 2(t_1+t_2) & t_1 & 0 & ... & 0 \\ t_3 & 2(t_2+t_3) & t_2 & ... & 0 \\ 0 & t_4 & 2(t_3+t_4) & ... & 0 \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ 0 & ... & 0 & t_m & 2(t_{m-1}+t_m) \end{bmatrix} \begin{bmatrix} \dot{q}^1 \\ \dot{q}^2 \\ \dot{q}^3 \\ ... \\ ... \\ \dot{q}^{m-1} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{t_1 t_2}[t_1^2(q^2-q^1) + t_2^2(q^1-q^0)] \\ \frac{3}{t_2 t_3}[t_2^2(q^3-q^2) + t_3^2(q^2-q^1)] \\ \frac{3}{t_3 t_4}[t_3^2(q^4-q^3) + t_4^2(q^3-q^2)] \\ ... \\ ... \\ \frac{3}{t_{m-1} t_m}[t_{m-1}^2(q^m-q^{m-1}) + t_m^2(q^{m-1}-q^{m-2})] \end{bmatrix} \tag{1.4}$$

Cette équation se généralise pour chaques articulations et se met sous la forme générale :

$$M\dot{q}_j = N$$

où la matrice M est une matrice tridiagonale (facilement inversible). Les deux équations 1.3 et 1.4 permettent de calculer les coefficients des fonctions cubiques F et donc de trouver l'interpolation de la trajectoire.

Pour les premier et dernier segments, on suppose que les accélérations initiale et finale sont nulles. Ces portions de trajectoire sont alors interpolées par des polynômes de quatrième degré. Leurs coefficients se déduisent des conditions aux limites:

$F_1(0) = q^0$, $\dot{F}_1(0) = 0$, $\ddot{F}_1(0) = 0$
$F_1(t_1) = q^1$, $\dot{F}_1(t_1) = \dot{q}^1$

et de la condition de continuité sur les accélérations:

$\ddot{F}_1(t_1) = \ddot{F}_2(0)$

Tous calculs faits, on trouve la relation suivante :

$[\frac{3}{t_1} + \frac{2}{t_2}]\dot{q}^1 + \frac{1}{t_2}\dot{q}^2 = \frac{3}{t_2^2}(q^2 - q^1) + \frac{6}{t_1^2}(q^1 - q^0)$

Un raisonnement identique sur le dernier segment nous donne :

$\frac{1}{t_{m-1}}\dot{q}^{m-2} + [\frac{3}{t_m} + \frac{2}{t_{m-1}}]\dot{q}^{m-1} = \frac{6}{t_m^2}(q^m - q^{m-1}) + \frac{3}{t_{m-1}^2}(q^{m-1} - q^{m-2})$

Il suffit donc de remplacer la première et la dernière équation du système 1.4 par les deux équations précédentes.

L'utilisation des fonctions splines possède l'avantage de satisfaire aux contraintes de continuité en vitesse et en accélération tout en assurant le chemin le plus court.

1.3.3 Passage de l'espace articulaire à l'espace opérationel

Dans l'espace opérationnel, l'orientation du repère outils est définit par une matrice de transformation homogène. Dans [3], il est montré que la différence d'orientation de deux points de l'espace opérationnel s'exprime en fonction de :

- un vecteur unitaire u porté par la droite reliant les deux points (origines des repères),
- un angle θ indiquant la rotation à effectuer autour de u pour passer de l'orientation initiale à l'orientation finale.

Le mouvement dans l'espace opérationnel se décompose en un mouvement de rotation de θ autour de u pour aligner les repères outils et en un mouvement de translation en ligne droite entre les origines de ces deux repéres.

Les algorithmes décrits dans les paragraphe précédents peuvent être implémentés dans l'espace opérationnel comme dans l'espace articulaire. Il suffit, pour cela, de considérer les trois valeurs de la position P et l'angle θ comme "*des variables articulaires*".

Par analogie avec l'équation 1.1, on a :

$P(t) = P^i + r(t)(P^f - P^i)$
$A(t) = A^i A(u, r(t)\theta)$

On s'apercoit que le probème est identique à celui de l'espace articulaire et revient à calculer une fonction d'interpollation $r(t)$ pour passer de l'état initiale $(^i)$ à l'état final $(^f)$.

1.3.4 Avantage d'un mouvement dans l'espace articulaire

La génération de mouvement dans l'espace articulaire présente plusieurs avantage dont :

- le mouvement est minimal sur chaque articulation,
- elle nécessite moins de calcul en ligne (pas d'appel au changeur de coordonnées),
- le mouvement n'est pas affecté par le passage en configuration singulière,

– les contraintes de vitesses sont connues avec précision puisqu'elles correspondent aux limites physiques des actionneurs.

En contrepartie, la géométrie de la trajectoire du repère outils est imprévisible, il y a donc risque de collision entre l'organe terminal et son environnement si celui-ci est encombré. Ce type de mouvement convient pour des déplacements rapides dans un espace dégagé.

1.3.5 Avantage d'un mouvement dans l'espace opérationnel

La génération de mouvement dans l'espace opérationnel permet de contrôler la géométrie de la trajectoire (déplacement de l'outils). Par contre :

– elle implique un appel au changeur de coordonnées (pour passer de la matrice de transformation homogène aux coordonnées articulaires),

– elle est mise en échec lorsque la trajectoire calculée passe par une position singulière,

– les limites en vitesses des articulations ne sont pas directement utilisables puisque qu'elles sont définies dans l'espace articulaire. Pour éviter tout dépassement, on impose au robot de travailler en dessous de ses moyens réels.

1.4 Conclusion

L'essentiel de ce chapitre a été consacré à la présentation des algorithmes implémentés dans la partie hiérarchique de génération de trajectoire pour les bras manipulateur. Nous avons vu que certain d'entre eux sont relativement simple par rapport à d'autre qui possèdent des caractéristiques de continuité en vitesse et en accélération. Nous les avons tous implémentés afin de proposer à l'utilisateur une gamme la plus large possible de trajectoire. Il pourra alors effectuer son choix en fonctions du "rapport" performance de l'algorithme / temps de calcul. L'objet du chapitre suivant est de présenter les méthodes et les astuces employées pour générer des modules ORCCAD.

Chapitre 2

Implémentation logicielle

2.1 Introduction

Le logiciel ORCCAD est développé à partir du langage C++. Ce langage permet le développement d'application de taille relativement modeste par une méthode de type RAD (Rapid Application Development). L'idée est d'écrire un logiciel très ouvert c'est à dire que l'on puisse y apporter facilement des modifications, y ajouter des nouvelles fonctionnalités et y connecter d'autres logiciels.

Mon travail s'est décomposé en deux parties, mettre en oeuvre les algorithmes présentés au chapitre 1 et créer les interfaces graphiques nécessaire à la génération de ce code. Le but de ce chapître est de présenter les méthodes utilisées pendant ce stage pour intégrer au logiciel les parties PR et TG de la hiérarchie pour les bras manipulateurs.

2.2 Technique de génération de modules

Dans ORCCAD, le module est le grain élémentaire d'une procédure robot. Au niveau informatique, il est décrit par deux fichiers graphiques (*nom_module*.data et *nom_module*.tm) et six fichiers contenant le code algorithmique. Ces fichiers sont :

- **inc.h :** contient les *define* et les *include*s ;
- **var.c :** contient les déclarations des variables et des ports utilisés ;
- **param.c :** contient le code de paramétrisation des modules ;
- **init.c :** contient le code de initialisation des variables contenues dans *var.c* ;
- **compute.c :** contient le code de calcul du module ;
- **end.c :** code de fin du module ;

D'un point de vue exécution de la procédure robot, ces six fichiers sont chacun encapsulés dans une méthode de la classe du module (le module est considéré comme un objet). Pendant la phase d'initialisation de la procédure robot, la méthode *init* est exécutée pour accomplir les initialisations nécessaire au module. Puis pendant le déroulement normal, c'est à dire à chaque nouvelle période d'échantillonnage,

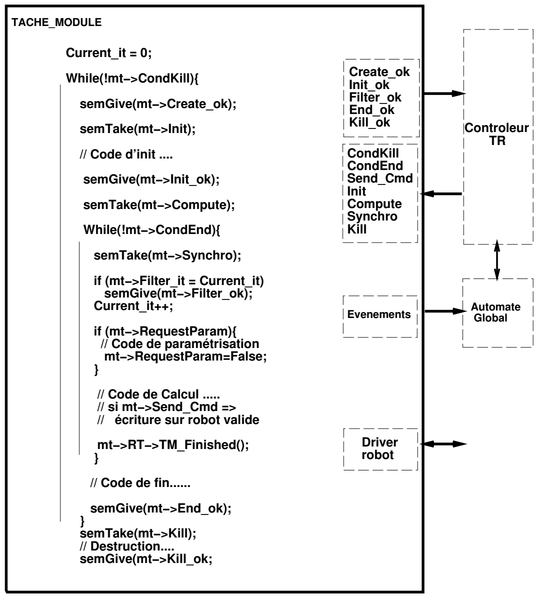

FIG. 2.1 – *Algorithme d'une Tâche-Module.*

c'est *compute* qui s'exécute. Puis finalement, c'est *end* qui termine la procédure (l'exécution d'une TM temps réel est décrite dans la figure 2.1).

Pour générer automatiquement des modules ORCCAD, deux solutions s'offraient à nous :

- créer l'ensemble des fichiers composant le module ;
- ou modifier les fichiers d'un module "pré-créé" avec l'éditeur de module.

Si la première méthode nous donne les tailles de fichier et les temps de calcul optimums, elle impose de générer tous les fichiers. Pour des raisons de simplicité, nous avons donc adopté la seconde méthode. Elle nécessite la mise à jour des deux fichiers contenant la description graphique du module et de compléter le fichier d'initialisation (init.c). Le principal avantage de cette méthode est qu'elle limite le nombre de ligne de code à générer (seulement la fin de init.c).

2.2.1 Modification des fichiers graphiques

Les fichiers graphiques *nom_module*.data et *nom_module*.tm contiennent l'ensemble des éléments permettant de construire graphiquement le module (pour l'utilisation dans l'éditeur de tâche robot). Les modifications à apporter à ces fichiers sont :

- mise à jour du nom du module généré ;
- mise à jour de la taille des ports de ce module.

Ces modifications sont effectuées par la commande système : "sed" (Unix). Le fait d'utiliser une commande spécifique au système Unix, ne nous permet pas de porter l'application sur un autre système d'exploitation (Windows NT par exemple) mais nous a permis, pendant la phase de développement logiciel, de changer la structure de nos modules sans avoir à "recoder" les fonctions de mise à jour des fichiers graphiques.

2.2.2 génération du fichier d'initialisation

Les fichiers d'initialisation (init.c) générés ont la même structure pour tous les algorithmes :

- initialisation des variable *methode*, $nbAxe$, $nbPoint$ et *time* ;
- lecture de la position courante ;
- lecture des points de la trajectoire ;
- calcul de D_j (distances à parcourir entre chaque couple de points) ;
- calcul des vitesses aux points de passage (pour les méthodes implémentant une trajectoire avec vitesse non nulle sur ces points) ;
- mise à zéro du port paramètre NbPoint ;
- vérification de la validité de la trajectoire ;

Pour générer le code, on utilise la classe *ofstream* de C++. Il suffit d'écrire, dans le fichier *init.c*, les lignes correspondant à la méthode d'interpolation sélectionnée.

Les modules générés possèdent donc un fichier d'initialisation dédié à la méthode choisie par l'utilisateur mais, à l'inverse, ils possèdent un fichier *compute.c* qui contient tous les algorithmes de génération de trajectoire. Le fait d'avoir tous les algorithmes dans le même fichier est insignifiant sur le temps de calcul puisque cela ajoute seulement un test sur la variable *methode*.

2.3 Exemple de modules générés

Les modules générateurs de trajectoire créés ont tous le même aspect (figure 2.2), seul leur nom permet de les différentier. Ils se composent de :

- deux ports événements *badtraj* et *endtraj* qui prennent la valeur 1 respectivement lorsque la trajectoire sort du domaine accessible par le robot ou lorsque la trajectoire se termine ;
- un port de sortie *trajectory*, c'est le port sur lequel est écrit la consigne générée ;
- un port d'entrée *posi* qui est connecté au module physique (le robot) et sur lequel est disponible, à chaque instant t, la position courante du robot ;
- deux ports paramètres *NbPoint* et *Ptraj*, le premier contient le nombre de point de la trajectoire alors que le second contient la durée de chaque segment de la trajectoire et la configuration articulaire des points de la trajectoire.

Les ports paramètres sont une des nouveautés du logiciel (par rapport à la version présentée dans le tome 1). Leur caractéristique principale est qu'ils peuvent être initialisés ou modifiés soit au niveau de l'édition de tâche robot soit pendant l'exécution de la procédure robot.

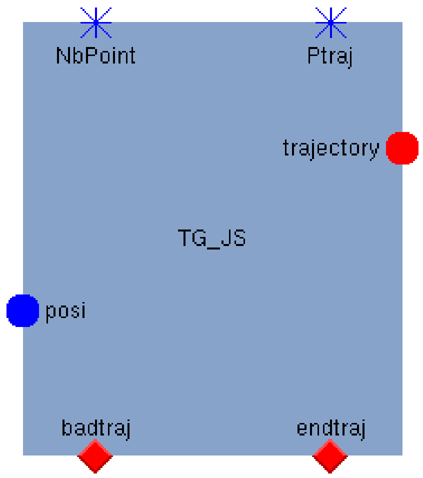

FIG. 2.2 – *Exemple de modules générés*

L'intéret d'utiliser ce type de port pour la génération de trajectoire est de créer des modules qui ne sont pas dédiés à une trajectoire unique. Ils nous permettent également de spécifier et de lancer une nouvelle trajectoire avant la fin de l'exécution de la précédente. Le mécanisme alors utilisé est décrit dans la figure 2.3, il s'agit d'un test sur le port paramètre *NbPoint*: si ce-dernier est différent de zéro, c'est que l'exécution d'une nouvelle trajectoire est demandée, il faut par conséquent refaire les initialisations.

2.4 Interface homme-machine

L'interface graphique d'ORCCAD est basée sur les classes C++ d'Ilog Views. Ces classes permettent la construction d'interface plus ou moins complexe. La technique est de construire graphiquement son interface avec l'utilitaire *ivstudio* (fourni par Ilog), de générer une classe de base correspondante et de dériver cette classe afin de pouvoir développer les méthodes désirées.

Les interfaces créées pendant le stage sont illustrées dans le manuel utilisateur (chapitre 3).

2.5 Les classes matrices

Le calcul des vitesses aux points de passage pour la méthode d'interpolation par fonctions splines et le calcul d'une consigne dans l'espace opérationnel nécessite la manipulation de matrice. Il nous était donc nécessaire de posséder une classe *matrice* qui réponde à nos besoin. Comme un grand nombre de classe manipulant les matrices sont actuellement disponible en *en freeware* sur Internet, nous en avons sélectionnée une, la classe *doubleMatrix* (de l'université d'Ucla au Etats Unis).

Pour manipuler les matrices homogènes, nous avions besoin d'un ensemble de méthodes qui n'étaient pas disponible dans la classe *doubleMatrix*. J'ai donc dérivée cette classe pour créer la classe *HMatrix* (figure 2.4), qui redéfinit les opérateurs de multiplication et d'inversion et implémente des méthodes permettant, entre autre, d'accéder à la matrice d'orientation et au vecteur de position.

```
// Debut algo
si (NbPoint != 0)  //nouvelle trajectoire demandee
     init();          // reinitialisation de la TM

switch( methode )
      // algo interpolation lineaire
      // algo interpolation polynome de degre 3
      ...
      // algo interpolation par fonctions spline
// fin de switch

// verification fin de trajectoire
// incrementation variable time

// fin algo
```

FIG. 2.3 – *Algorithme simplifié des fichiers* compute.c.

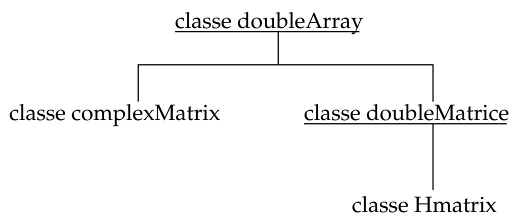

FIG. 2.4 – *Héritage des classes manipulant les matrices.*

La classe *HMatrix* est aussi utilisée dans la partie ressource physique (**PR**) pour stocker les matrices homogènes correspondant aux différents repères.

2.6 Conclusion

Après avoir présenté la partie développement logicielle de mon sujet de stage, le dernier chapitre expose la facon d'utiliser cette partieJe me suis contenté dans ce chapitre de présenter les techniques utilisées pour créer les interfaces graphiques et pour générer les modules ORCCAD. , il s'agit en fait du manuel utilisateur. Un autre manuel concernant les hiérarchies spécialisées est disponible en annexe, il s'agit du manuel de référence (ou manuel programmeur) dans lequel sont présentées les classes développées pendant mon stage.

Chapitre 3

Manuel utilisateur

3.1 Introduction

Ce chapitre est le manuel utilisateur des parties PR et TG de la hiérarchie spécialisée pour les robots du type bras manipulateur. Il est illustré par les interfaces graphiques qui permettent maintenant de générer des modules TG et de sauver les paramètres physiques du robot.

3.2 Panneau hiérarchie spécialisée

Une grande partie des techniques utilisées dans les domainess d'application comme les bras manipulateurs et les robots mobiles sont bien connues et fixées. Il parait donc profitable de représenter un certain nombre d'algorithmes sous une vue hiérarchique et réutilisable.

Le panneau (figure 3.1) est accessible à partir de l'éditeur de module soit par l'icone *module hierarchy* soit par le menu *Tools/hierarchy of TMs...*. Il permet d'accéder aux hiérarchies spécialisées.

La barre de menu propose quatre choix :

- **System :** permet de retourner à l'éditeur de modules ;
- **Domain :** permet de choisir la hiérarchie spécialisée (robot manipulateur ou mobile pour l'instant) ;
- **Config :** permet de configurer l'affichage des arbres ;
- **Help :** permet d'accéder à l'aide en ligne d'ORCCAD.

La figure 3.1 montre la sélection de l'arbre de génération de trajectoire pour les robots manipulateurs. Pour accéder au panneau suivant, il faut cliquer sur une des feuilles terminales de cet arbre. Les zones de textes situées en bas du panneau renseignent l'utilisateur sur les choix qu'il a effectu'e et sur l'action à venir.

3.3 Panneau ressource physique

Ce panneau appartient à la hiérarchie pour les robots manipulateurs, le but (figure 3.2) est de permettre la saisie de tous les paramètres physiques du robot et de les sauvegarder dans un fichier de

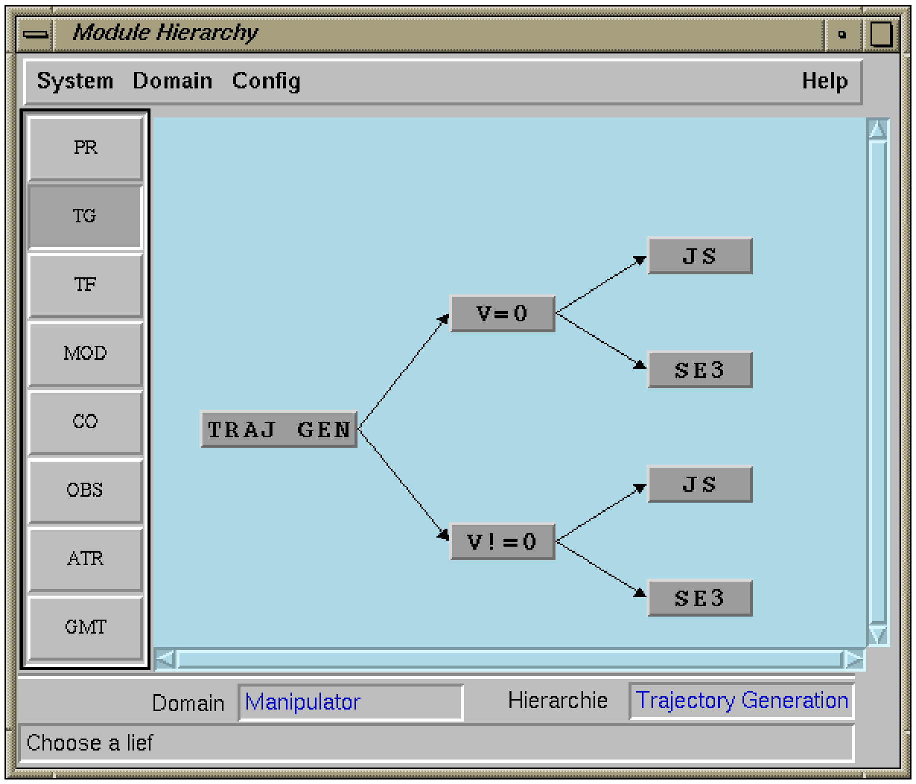

FIG. 3.1 – *Le panneau hiérarchie spécialisée.*

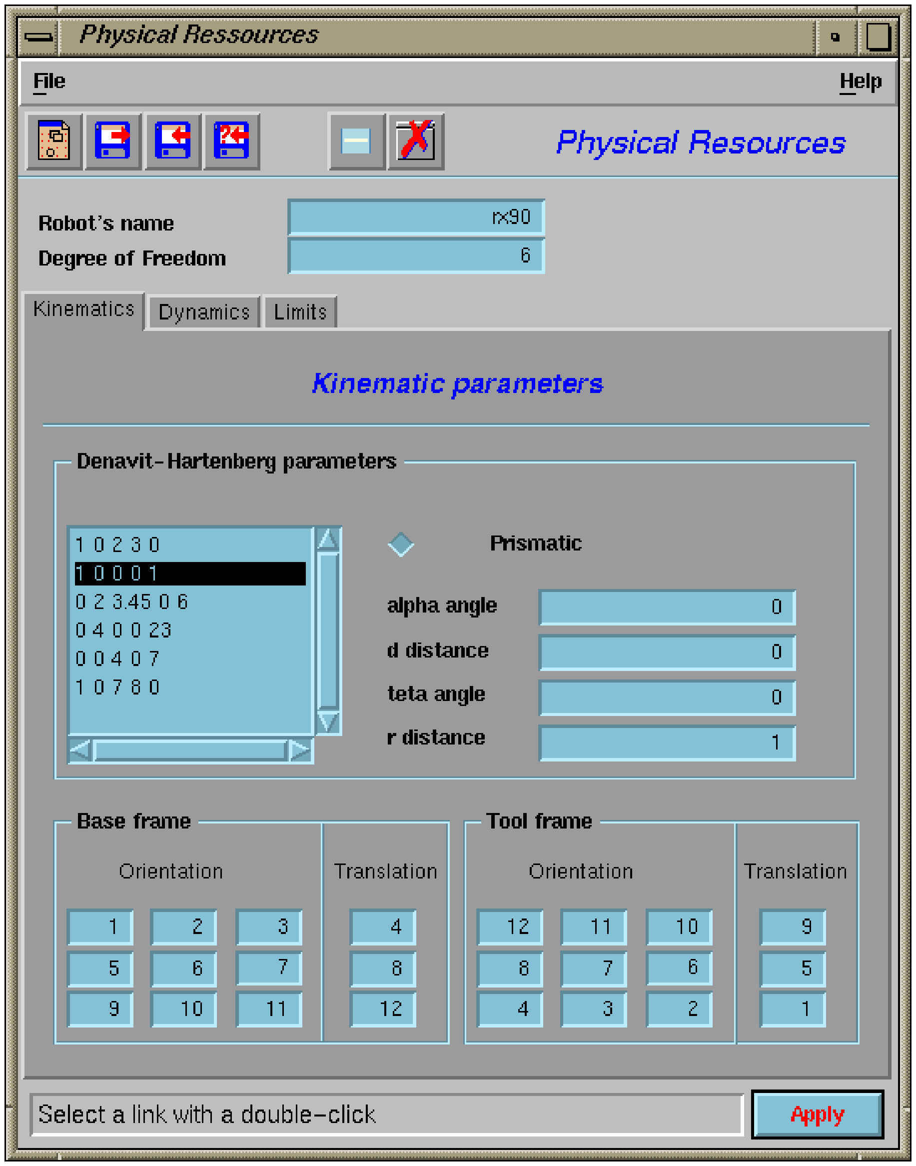

FIG. 3.2 – *Le panneau ressource physique*

configuration (description donnée en annexe). Il permet également la re-lecture de ces fichiers pour la correction des erreurs éventuelles.

Il est accéssible en cliquant sur l'unique feuille de la hiérarchie PR.

Le menu déroulant obtenu en cliquant sur *file* donne accès aux fonctions *new, Open, Save, SaveAs, Delete* et *Quit*. Les mêmes fonctions sont disponible par la barre d'icône horizontale. Le panneau se compose de deux champs (*Robot's name et Degree of freedom*) et d'un *notebook*. Le *notebook* dispose de trois pages :

- La page *Kinematics* : elle permet de rentrer les paramètres de Denavit-Hartenberg et les matrices homogènes correspondant au repère de base et au repère de l'outil (figure 3.2) ;
- La page *Dynamics* : elle permet la saisie des paramètres dynamiques du robot, c'est à dire de la masse du lien sélectionné, de la matrice d'inertie de ce lien et du repère associé à cette matrice d'inertie ;
- La page *Limits* : elle permet la saisie des valeurs limites caractéristiques du robot (position, vitesse, accélération et couple).

Remarques :

- chaque page du *notebook* contient une liste dans laquelle sont affichés les paramètres du lien (ou corps) sélectionné ;
- pour sélectionner un lien du robot, il faut double-cliquer sur la ligne correspondante dans une des trois listes. Les paramètres du corps sélectionné sont alors affichés dans les trois pages ;
- pour valider la modification des paramètres d'un corps, il faut cliquer sur *Apply* ;
- un exemple de fichier de configuration se trouve en annexe.

Pour créer un fichier de configuration, les paramètres du robot sont au préalables stockés dans un objet de la classe Robot (décrite en annexe).

3.4 Panneaux génération de trajectoire

3.4.1 Génération de trajectoire dans l'espace articulaire

Ce panneau est accessible en cliquant sur les feuilles terminales JS (*Joint Space*) de l'arbre TG (voir figure 3.1).

La création d'un module de génération de trajectoire dans l'espace articulaire nécessite deux types principaux de paramètres :

- les paramètres concernant la trajectoire (méthode d'interpolation, lissage (*Smoothless*) et nombre de point intermédiaire) ;
- et les paramètres physiques du robot.

Le panneau (figure 3.3) permet de saisir ces paramètres. Il permet également la re-lecture d'un fichier de configuration, si l'utilisateur n'a pas construit ce fichier, une option *NO-ROBOT* est disponible dans le menu déroulant *Select a robot* pour lui permettre de les rentrer à partir du clavier.

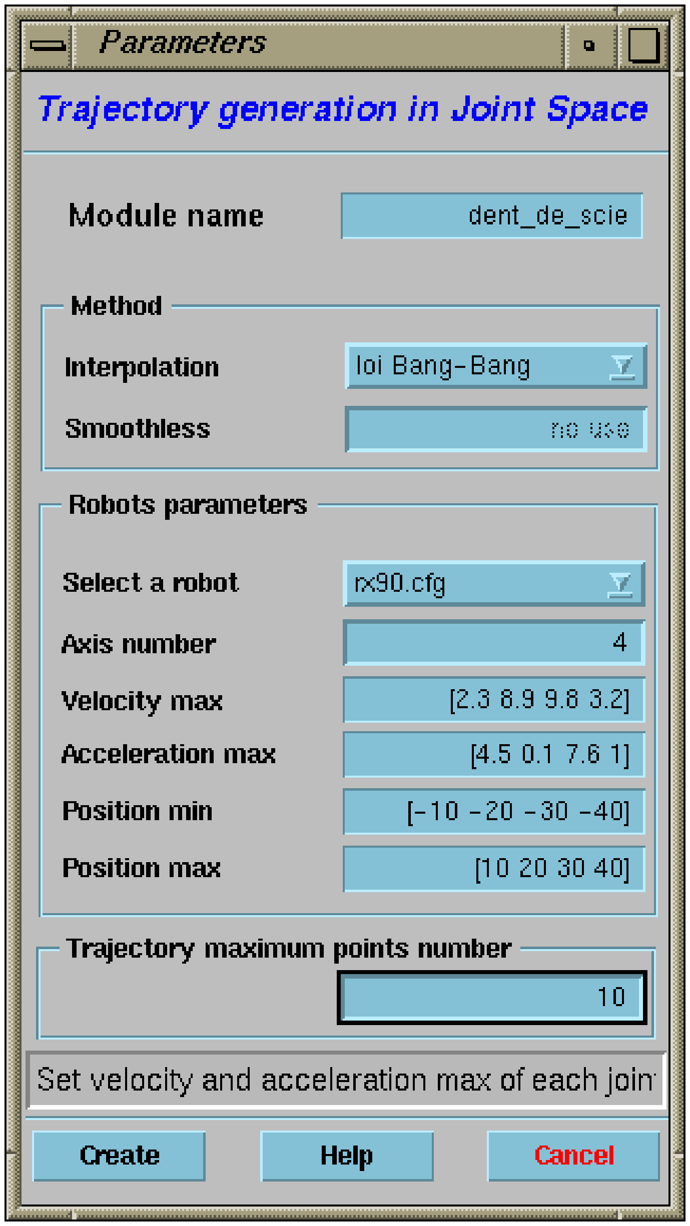

FIG. 3.3 – *Le panneau génération de trajectoire dans l'espace articulaire*

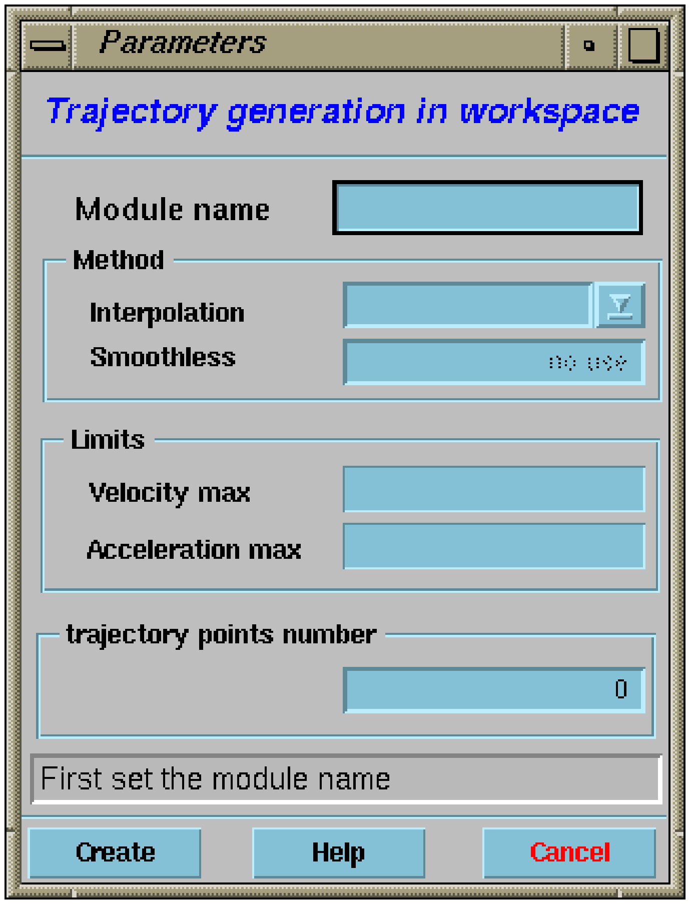

FIG. 3.4 – *Le panneau génération de trajectoire dans l'espace opérationnel*

Remarques :

- le champ *Smoothless* est le coefficient de lissage de la trajectoire pour la méthode qui implémente une trajectoire composé dínterpolations linéaires et de transitions paraboliques. Il correspond à la durée τ de la transition (figure1.8) ;
- le champs *Trajectory maximun points number* sert à fixer la taille du *buffer* dans lequel seront entrés les points de la trajectoire.

3.4.2 Génération de trajectoire dans l'espace opérationnel

Les mêmes algorithmes étant implémentés dans l'espace articulaire et dans l'espace opérationnel, on retrouve, dans le panneau (figure 3.4), la même structure que dans le panneau 3.3.

La principale différence réside dans les champs contenant les valeurs limites. Dans l'espace opérationnel, on ne connaît pas a priori les positions minimum et maximum accessibles par le robot, de même pour les vitesse et accélération limites. Pourtant, le panneau 3.4 présente deux champs pour les vitesses et les accélérations maximums. Ces valeurs étant nécessaires au calcul de la trajectoire, l'utilisateur devra les choisir de facon à ne pas dépasser les contraintes imposées par les articulations. On retrouve ici un des inconvénients de la génération de trajectoire dans l'espace opérationnel.

Remarque : les champs *Velocity max* et *Acceleration max* contiennent chacun deux valeurs, la vitesse ou l'accélération “maximun” en translation et la vitesse ou l'accélération “maximun” en rotation.

Conclusion

Le travail effectué pendant ce stage peut se décomposer en deux parties, la première a permis le développement d'actions robotiques pour le robot pendule. Le but de ce développement est, dans un premier temps, de donner un regard neuf d'utilisation du logiciel et dans un deuxième temps, de me familiariser avec la méthodologie de programmation ORCCAD. Le travail de la seconde partie est une phase de développement logiciel, il permet actuellement la saisie et la sauvegarde des caractéristiques physiques d'un robot manipulateur et la création automatique de modules générateur de trajectoire. La ré-utilisation des paramètres physiques d'un robot est réalisée par la fonction *RobotNameCB* de la classe *OrcPanelTGJS*, cette fonction fait le lien entre les hiérarchies PR et TG, elle pourra être utilisée pour les hiérarchies TF, MOD, CO, OBS et ATR qui restent à développer. La plupart des algorithmes de ces parties ont déjà été implémentés, il suffit donc de les intégrer dans ORCCAD et de réaliser les interfaces graphiques correspondantes. Il reste également à ajouter ou à modifier les lignes de code dans lesquelles se trouvent des appels systèmes, ceci afin de pouvoir porter le logiciel sur d'autres systèmes d'exploitation (Windows NT par exemple).

Sur plan personnel, j'ai apprécié le travail que l'on m'a confié pendant ce stage et ce pour diverses raisons.

Tout d'abord le sujet du stage m'a contraint à employer de nouveaux outils (environnemt temps réel de *Tornado*, classes graphiques d'*IlogViews*,..) qui sont relativement utilisés dans le monde de l'informatique et je suis persuadé que ce sera pour moi un plus de les connaître.

J'ai aussi apprécié de travailler dans le domaine, nouveau pour moi, de la robotique. C'est un domaine très exigent qui demande des compétences en automatique, en informatique mais aussi en électronique et en électrotechnique. De plus, il permet de travailler sur des projets très intéressant, par exemple le bras manipulateur Staubli rx90 de l'INRIA Rhône-Alpes est utilisé, en coopération avec un autre robot, dans une manipulation de poursuite de cible.

Et enfin, la dernière raison qui fait que mon travail a été valorisant, c'est d'avoir adopter une méthodologie de travail en projet, c'est à dire en étroite coopération avec les autres personnes qui développent le logiciel ORCCAD.

Annexe A

Terminologie et définitions associées à la robotique

Le but de cet annexe est de familiariser le lecteur avec les termes robotiques employés dans la suite de ce document. Les définitions, explications et autres illustrations présentées ici sont volontairement limitées aux robots du type bras manipulteurs.

Avant toutes choses, la première définition qu'il convient de rappeler est celle du robot, telle que l'Association Francaise de NORmalisation (AFNOR) l'a décrite: “Un robot est un manipulateur commandé en position, reprogrammable, polyvalent, à plusieurs degrés de liberté, capable de manipuler des matériaux, des pièces, des outils et des dispositifs spécialisés, au cours de mouvements variables et programmés pour l'exécution d'une variété de tâches. Il a souvent l'apparence d'un ou plusieurs bras se terminant par un poignet. Son unité de commande utilise, notament, un dispositif de mémoire et éventuellement de perception et d'adaptation à l'environnement et aux circonstances. Ces machines polyvalentes sont généralement étudiées pour effectuer la même fonction de facon cyclique et peuvent être adaptées à d'autres fonctions sans modifications permanente du matériel”.

Dans cette définition apparaîssent les termes *degrés de liberté* et *poignet*, ils seront explicités dans le paragraphe A.1. D'autres termes seront également présentés comme par exemple *espace articulaire*, *espace opérationnel* ou encore *paramètres de Denavit-Hartenberg*.

A.1 Articulations et corps

Une articulation lie deux corps successifs entre eux en limitant le nombre de degrés de liberté de l'un par rapport à l'autres. Le degré de liberté d'une articulation est donc la mobilité que celle-ci possède dans l'espace. Il existe deux types d'articulation :

- les articulations prismatiques ;
- les articulations rotoïdes.

A.1.1 Articulation prismatique

Il s'agit d'une articulation de type glissière, limitant le mouvement à une translation entre deux corps le long d'un axe commun. Elle est représentée par le symbole figure A.1. La situation relative entre les deux corps est mesurée par la distance le long de cet axe.

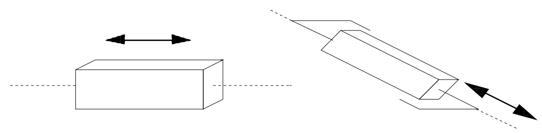

FIG. A.1 – *Représentation d'une articulation prismatique*

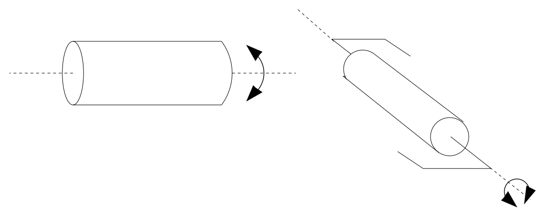

FIG. A.2 – *Représentation d'une articulation rotoïde*

A.1.2 Articulation rotoïde

Il s'agit d'une articulation de type pivot, réduisant le mouvement entre deux corps à une rotation autour d'un axe qui leurs est commun. La situation relative entre les deux corps est donnée par l'angle autour de cet axe. L'articulation est représentée par le symbole figure A.2.

A.1.3 Bras manipulateur

Les bras manipulateurs a six degrés de liberté (ddl) sont les structures de robot manipulateur les plus classiques et les plus répandues. Les 3 premiers *ddl*s forment ce que l'on appelle le *porteur du robot*, les trois autres forment le *poignet*. Le poignet se caractérise par trois articulations d'axes concourants et des dimensions et des masses plus faibles (figure A.3).

Le manipulateur utilisé à l'INRIA Rhône-alpes et sur lequel sont testés les modules (ou codes) générés par les classes de hiérarchie spécialisée d'ORCCAD est un robot manipulateur Staubli RX90 à 6 *ddl*s dont la modélisation est donnée figure A.5

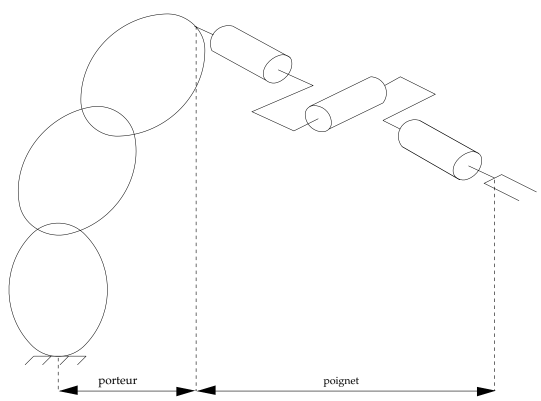

FIG. A.3 – *Structure générale d'un robot*

A.2 Espace articulaire

On appelle *configuration articulaire* ou *coordonnées articulaires* d'un robot manipulateur, l'état du robot représentant la situation de ses différents corps. La solution la plus classique pour décrire cet état est l'utilisation des variables, ou coordonnées articulaires (angle pour une articulation rotoïde, distance pour une articulation prismatique). Pour un robot a structure ouverte (simple ou arborescente), le nombre de variables articulaires correspond au nombre de degré de liberté.

A.3 Espace opérationnel

L'*espace opérationnel* ou le *repère outils* est l'espace dans lequel est représenté la situation de l'organe terminal (l'outils). Il est le plus souvent décrit par des matrices 4x4 appellées *matrices de transformation homogène T*. Ces matrices identifient la position de l'origine du repère et son orientation dans l'espace. Elles se composent d'une sous-matrice 3x3 d'orientation **A** (de l'anglais *Attitude*) et d'un vecteur position **P** et s'expriment par rapport à un repère référence du poste de travail.

$$T = \begin{pmatrix} s_x & n_x & a_x & p_x \\ s_y & n_y & a_y & p_y \\ s_z & n_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} A & P \\ 0 & 1 \end{pmatrix}$$

$$\text{où} \quad A = \begin{pmatrix} s_x & n_x & a_x \\ s_y & n_y & a_y \\ s_z & n_z & a_z \end{pmatrix} \quad \text{et} \quad P = \begin{pmatrix} P_x & P_y & P_z \end{pmatrix}$$

Remarques :

- L'outils est considéré comme un solide indéformable et libre dans l'espace possédant 6 degrés de liberté, il est donc nécéssaire de posséder 6 paramètres (3 vecteurs de **A** et 3 composantes de **P**) pour le positionner et l'orienter dans l'espace.
- Pour passer de l'espace opérationnel à l'espace articulaire, il faut utiliser un changeur de coordonnées qui se calcul à partir du modèle géométrique inverse (MGI : voir [3]). Le passage en coordonnées articulaires n'est pas toujours possible, on peut tomber dans une configuration dite singulière (pas de solution).

A.4 Les paramètres de Denavit-Hartenberg

Les paramètres de Denavit-Hartenberg sont utilisés pour décrire la structure géométrique des robots. Ils sont à la base de la mise en équations de tous les modèles de robots. La méthode présentée ici est la plus répandue et aussi la mieux adaptée au robot à structure ouverte simple. Ce paragraphe introduit les notations utilisées pour la description des robots à chaine ouverte simple et les illustre avec l'exemple de la modélisation du robot Staubli RX90 (figure A.5).

La méthode est basée sur les règles et conventions suivantes :

- la variable de l'articulation j est notée q_j ;
- le corps j est noté C_j ;

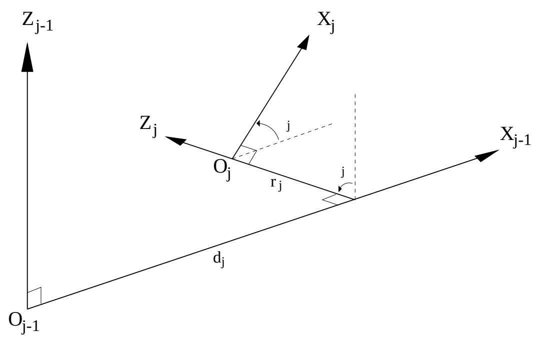

FIG. A.4 – *Représentation des paramètres de Denavit-Hartenberg.*

- les corps sont supposés parfaitement rigide et les articulations idéales ;
- le repère R_j est lié à C_j ;
- l'articulation j connecte le corps C_j au corps C_{j-1} ;
- l'axe z_j de R_j est porté par l'articulation j ;
- l'axe x_j est porté par la perpendiculaire commune aux axes z_j et z_{j+1}.

Le passage de R_{j-1} à R_j s'exprime en fonction des quatre paramètres suivants :

- α_j : angle entre les axes z_{j-1} et z_j, correspondant à une rotation autour de x_{j-1} ;
- d_j : distance entre z_{j-1} et z_j le long de x_{j-1} ;
- θ_j : angle entre les axes x_{j-1} et x_j, correspondant à une rotation autour de z_{j-1} ;
- r_j : distance entre x_{j-1} et x_j le long de z_{j-1}.

Ces quatre paramètres sont les paramètres de Denavit-Hartenberg (DH). Ils sont représentés figure A.4.

La variable articulaire q_j associée à la j^{ieme} articulation est soit θ_j, soit r_j, selon que l'articulation est de type rotoïde ou prismatique, ce qui se traduit par la relation :

$$q_j = \overline{\sigma}_j\theta_j + \sigma_j rj$$

avec $\sigma_j = 0$ si l'articulation j est rotoïde, 1 sinon.

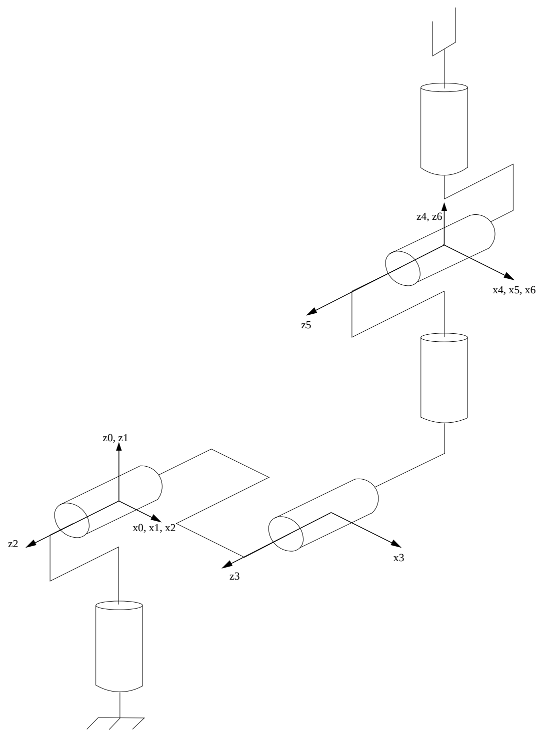

FIG. A.5 – *Modélisation du robot Staubli rx90.*

Annexe B

Exemple de fichier de configuration

L'exemple suivant montre le fichier de configuration pour un robot à deux articulations, les valeurs du repère de base, du repère de l'outil et des liens sont fausses.

```
%**********************************************
%**          ROBOT CONFIGURATION FILE        **
%**********************************************

ROBOT_NAME robotXXX
DOF         2
BASE_FRAME 1.01 22 33 10
           44 55 66 20
           77 88 99 30
TOOL_FRAME 10 20 30 110
           40 50 60 220
           70 80 90 330

%***************** LINK 0 ****************
MASS 1
INERTIA 2 3 4
        5 6 7
        8 9 10
ATTITUDE 11 12 13
         14 15 16
         17 18 19
POSITION 20 21 22
MDH_PARAMETERS 1 1 2 3 4
JOINTS_LIMITS -12 12
VELOC_LIMIT 56
ACCEL_LIMITS 78
TAU_LIMITS 90
```

```
%***************** LINK 1 ****************
MASS 10
INERTIA 20 30 40
        50 60 70
        80 90 100
ATTITUDE 110 120 130
         140 150 160
         170 180 190
POSITION 200 210 220
MDH_PARAMETERS 0 5 6 7 8
JOINTS_LIMITS -9 9
VELOC_LIMITS 65
ACCEL_LIMITS 43
TAU_LIMITS 21
```

Annexe C

Manuel de référence

CLASS orcPanelModel

Inheritance Path

orcPanelModelB

Description

Use to display the specialised hierarchy panel

Header File

"Inc/Ihm/orcPanelModel.h"

Synopsis

```
class OrcPanelModel : public OrcPanelModelB{
friend void addManipulatorTree(IlvMatrix*, IlvUShort, IlvUShort, IlvAny);
friend void addMobileTree(IlvMatrix*, IlvUShort, IlvUShort , IlvAny );
public:
    OrcPanelModel(// OrcPanelEdmod *sed,
                IlvDisplay* d, const char* , const char* ,
                IlvRect* size = 0 ,
                IlvBoolean useAccelerators=IlvFalse ,
                IlvBoolean visible = IlvFalse);

    void changeEditInteractor(IlvManagerViewInteractor* mi);

    void createTrees();

    void addPrTree();
    void addTgTree();
    void addTfTree();
    void addModTree();
    void addCoTree();
```

```
        void addObsTree();
        void addAtrTree();
        void addGmtTree();

        void removeTree();

        IlvInt getPretty_x();
        IlvInt getPretty_y();

        IlvMatrix *getMatrix();
        IlvGrapher *getGrapher();
        IlvView *getView();

        void CreateManipulatorInteractors();
        void CreateMobileInteractors();

        void MenuManipulatorCB(IlvGraphic*);
        void MenuPretyXCB(IlvGraphic*);
        void matrixCB(IlvGraphic*);
        void TextMessagesCB(IlvGraphic*);
        void TextDomainCB(IlvGraphic*);
        void MenuQuitCB(IlvGraphic*);
        void MenuMobileCB(IlvGraphic*);
        void MenuPrettyYCB(IlvGraphic*);
        void MenuRessourcesCB(IlvGraphic*);
        void TextHierarchieCB(IlvGraphic*);
        void MenuHelpCB(IlvGraphic*);
};
```

Constructor

```
orcPanelModel::OrcPanelModel(
    // OrcPanelEdmod *sed,
    IlvDisplay* d, const char* name, const char* title,
    IlvRect* size = 0 ,
    IlvBoolean useAccelerators=IlvFalse ,
    IlvBoolean visible = IlvFalse):
orcPanelModelB(d, name, title, size, useAccelerators, visible);
```

CLASS orcPanelPR

Inheritance Path

orcPanelPRB

Description

Use to display the Physical Resource panel

Header File

"Inc/Ihm/orcPanelPR.h"

Synopsis

```
class OrcPanelPR : public OrcPanelPRB{
public:
        OrcPanelPR(ExploreInteractor* _h,
                        IlvDisplay* display,
                        const char* name,
                        const char* title,
                        IlvRect* size = 0,
                        IlvBoolean useAccelerators = IlvFalse
                        IlvBoolean visible = IlvFalse);
        void choicePrismRotCB(IlvGraphic*);
        void OpenCB(IlvGraphic*);
        void ListParamDHCB(IlvGraphic*);
        void ListLinkCB(IlvGraphic*);
        void ListLimitCB(IlvGraphic*);
        void QuitCB(IlvGraphic*);
        void NewCB(IlvGraphic*);
        void SaveCB(IlvGraphic*);
        void SaveAsCB(IlvGraphic*);
        void DeleteCB(IlvGraphic*);
        void HelpPRCB(IlvGraphic*);
        void ApplyCB(IlvGraphic*);
        void RobotNameCB(IlvGraphic*);
        void DoFCB(IlvGraphic*);

        void ClearLists();
        void SetCB();

        Robot* getRobot();
        IlvUShort getSelectedPos();
        void putSelectedPos(IlvUShort pos);
};
```

Constructor

orcPanelPR::OrcPanelPR(

```
    ExploreInteractor* _h,
    IlvDisplay* display,
    const char* name,
    const char* title,
    IlvRect* size,
    IlvBoolean useAccelerators,
    IlvBoolean visible):
OrcPanelPRB(display,name,title,size,useAccelerators,visible);
```

CLASS orcPanelTGJS

Inheritance Path

orcPanelTGJSB

Description

Use to display the TG in JS generation module panel

Header File

"Inc/Ihm/orcPanelTGJS.h"

Synopsis

```
class OrcPanelTGJS : public OrcPanelTGJSB{
public:
        OrcPanelTGJS(ExploreInteractor* _h,
                        IlvInt typeTraj,
                        IlvDisplay* display,
                        const char* name,
                        const char* title,
                        IlvRect* size = 0,
                        IlvBoolean useAccelerators = IlvFalse
                        IlvBoolean visible = IlvFalse);
        void PosMinCB(IlvGraphic*);
        void ButtonCancelCB(IlvGraphic*);
        void ButtonCreateCB(IlvGraphic*);
        void AccelMaxCB(IlvGraphic*);
        void ButtonHelpCB(IlvGraphic*);
        void ModuleNameCB(IlvGraphic*);
        void PosMaxCB(IlvGraphic*);
        void nbAxeCB(IlvGraphic*);
        void InterpolationCB(IlvGraphic*);
        void VelocityMaxCB(IlvGraphic*);
        void nbPointCB(IlvGraphic*);
        void SmoothlessCB(IlvGraphic*);
        void RobotNameCB(IlvGraphic*);

        IlvInt createModule();
        IlvInt createDATAFile();
        IlvInt createTMFile();
        IlvInt createINITFile();
        void Alert(const char*);
        void Assign(IlvInt, const char*);
};
```

Constructor

OrcPanelTGJS::OrcPanelTGJS(

ExploreInteractor* _h,
IlvInt typeTraj,
IlvDisplay* display,
const char* name,
const char* title,
IlvRect* size,
IlvBoolean useAccelerators,
IlvBoolean visible):
OrcPanelTGJSB(display,name,title,size,useAccelerators,visible);

CLASS orcPanelTGSE3

Inheritance Path

orcPanelTGSE3B

Description

Use to display the TG in SE3 generation module panel

Header File

"Inc/Ihm/orcPanelTGSE3.h"

Synopsis

```
class OrcPanelTGSE3 : public OrcPanelTGSE3B{
public:
        OrcPanelTGSE3(ExploreInteractor* _h,
                      int typeTraj,
                      IlvDisplay* display,
                      const char* name,
                      const char* title,
                      IlvRect* size = 0,
                      IlvBoolean useAccelerators = IlvFalse
                      IlvBoolean visible = IlvFalse);
        void VelocityMaxCB(IlvGraphic*);
        void InterpolationCB(IlvGraphic*);
        void ModuleNameCB(IlvGraphic*);
        void AccelMaxCB(IlvGraphic*);
        void nbPointCB(IlvGraphic*);
        void ButtonCancelCB(IlvGraphic*);
        void ButtonCreateCB(IlvGraphic*);
        void ButtonHelpCB(IlvGraphic*);

        void Assign(IlvInt, const char*);
        IlvInt createModule();
        IlvInt createDATAFile();
        IlvInt createTMFile();
        IlvInt createINITFile();
        void Alert(const char* msg);
};
```

Constructor

```
OrcPanelTGSE3::OrcPanelTGSE3(
    ExploreInteractor* _h,
    int typeTraj,
    IlvDisplay* display,
    const char* name,
    const char* title,
```

```
    IlvRect* size,
    IlvBoolean useAccelerators,
    IlvBoolean visible):
OrcPanelTGSE3B(display,name,title,size,useAccelerators,visible);
```

CLASS HMatrix

Inheritance Path

doubleMatrix

Description

Implement methods to manipulate homogeneous matrix, inherite from doubleMatrix class. We have picked the doubleMatrix class on the web (university of UCLA).

Header File

"Inc/Ihm/HMatrix.h"

Synopsis

```
class HMatrix : public doubleMatrix{
friend HMatrix operator % (HMatrix t1,HMatrix t2);
friend double* UTeta(HMatrix ti, HMatrix tf);
public:
        HMatrix();
        HMatrix(double *a, double *p);
        HMatrix(int AE_RTL, double phi, double teta, double ksi, double *p);
        HMatrix(double a, double d, double o, double r);
         HMatrix();

        HMatrix a(double* a); // putA
        HMatrix p(double* p); // putP
        doubleArray a(); // getA
        doubleArray p(); //getP
        double* Euler();
        double* rtl();
        void putPERot(double phi, double teta, double ksi);
        void putRTLRot(double phi, double teta, double ksi);
        void getDHParam(double &a, double &d, double &o, double &r);

        HMatrix i(); //inversion
};
```

CLASS orcLink

Inheritance Path

Base class

Description

what is your mission on earth?

Header File

"Inc/Ihm/orcLink.h"

Synopsis

```
class Link {
friend ostream& operator ≪ (ostream&, Link&);
friend FILE* operator ≫ (FILE*, Link&);
public:
        Link();
        Link(FILE *cfg_ptr);
        void PutMass(double mass);
        void PutInertia(CartesianTensor inertia);
        void PutIfAttitude(RotationMatrix ifa);
        void PutIfPos(CartesianVector ifp);
        void PutDenavitHartenberg(double, double,double,double,double);

        void PutPosLimits(double PosMin, double PosMax);
        void PutVelLimit(double velLimit);
        void PutAccLimit(double accLimit);
        void PutTauLimit(double tauLimit);

        double GetMass();
        void GetDenavitHartenberg(DenavitHartenberg&);
        void GetInertia(CartesianTensor&);
        void GetIfAttitude(RotationMatrix&);
        void GetIfPos(CartesianVector&);

        double GetMinJointLimit();
        double GetMaxJointLimit();
        double GetVelLimit();
        double GetAccLimit();
        double GetTauLimit();
};
```

Annexe D

Résultats expérimentaux

Les courbes présentées dans les figures D.1 et D.2 sont les résultats d'une procédure de test qui simule l'exécution normale de l'application. Ils vont être expérimentés sur le robot Staubli RX90 (figure D.3).

D.1 Exemple de trajectoire d'une articulation

Les trajectoires sont rentrées par le moyen des ports paramètres à partir de l'èditeur de tâche robot, pour une la figure D.1, les valeurs des points intermdiaires sont les suivantes (pour un robot à 6 articulations rotoïdes):

dur ee articulation1	articulation2	articulation3	articulation4	articulation5	articulation6
3.0-5.0	-35.0	-50.0	18.0	5.0	-90.0
4.00.0	0.0	0.0	0.0	0.0	0.0
6.05.0	35.0	50.0	-18.0	-5.0	90.0
4.00.0	0.0	0.0	0.0	0.0	0.0
5.05.0	-35.0	-50.0	18.0	5.0	-90.0
6.00.0	0.0	0.0	0.0	0.0	0.0
6.05.0	35.0	50.0	-18.0	-5.0	90.0

D.2 Valeurs des points intermédiaires d'une trajectoire dans l'espace opérationnel

Cette trajectoire est celle avec laquelle les modules générateurs de trajectoire dans l'espace opérationnel ont été testés, elle comporte six points intermédiaires. Ces points sont entrés à partir du port paraètre Ptraj avec le format suivant :

durée – matrice de rotation – vecteur de translation

La durée est le temps désirée pour accomplir le parcours entre la position courante du robot et le point saisie sur la même ligne que la durée.

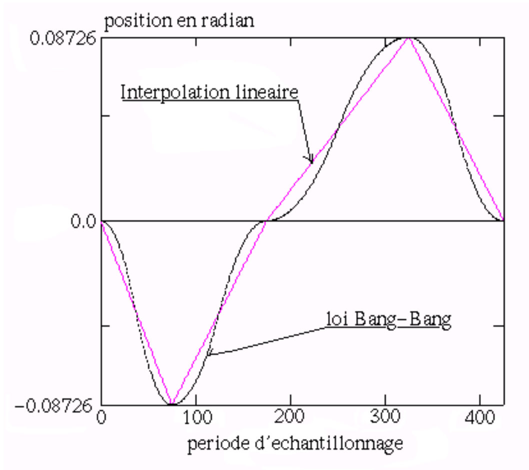

FIG. D.1 – *Exemple de courbes obtenues par les algorithmes : interpolation lineaire et loi Bang-Bang.*

duration	—A1—	—A2—	—A3—	—A4—	—A5—	—A6—	—A8—	—A9—	—A10—	—P1—	—P2—	—P3—
5	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.707	0.707	0.0	-0.707	0.707	22	2	22
5	0.0	-0.707	0.707	1.0	0.0	0.0	0.0	0.707	0.707	22	2	22
3	0.0	-0.707	0.707	0.0	-0.707	-0.707	1.0	0.0	0.0	12	2	5
5	-0.707	0.0	0.707	-0.707	0.0	-0.707	0.0	-1.0	0.0	10	1	10
3	-0.5	0.0	-0.866	0.866	0.0	-0.5	0.0	-1.0	0.0	6	16	6

où A est la matrice de rotation (*Attitude*) et P le vecteur de position. Les trois lignes de la matrice de rotation A sont mise à la suite les unes des autres.

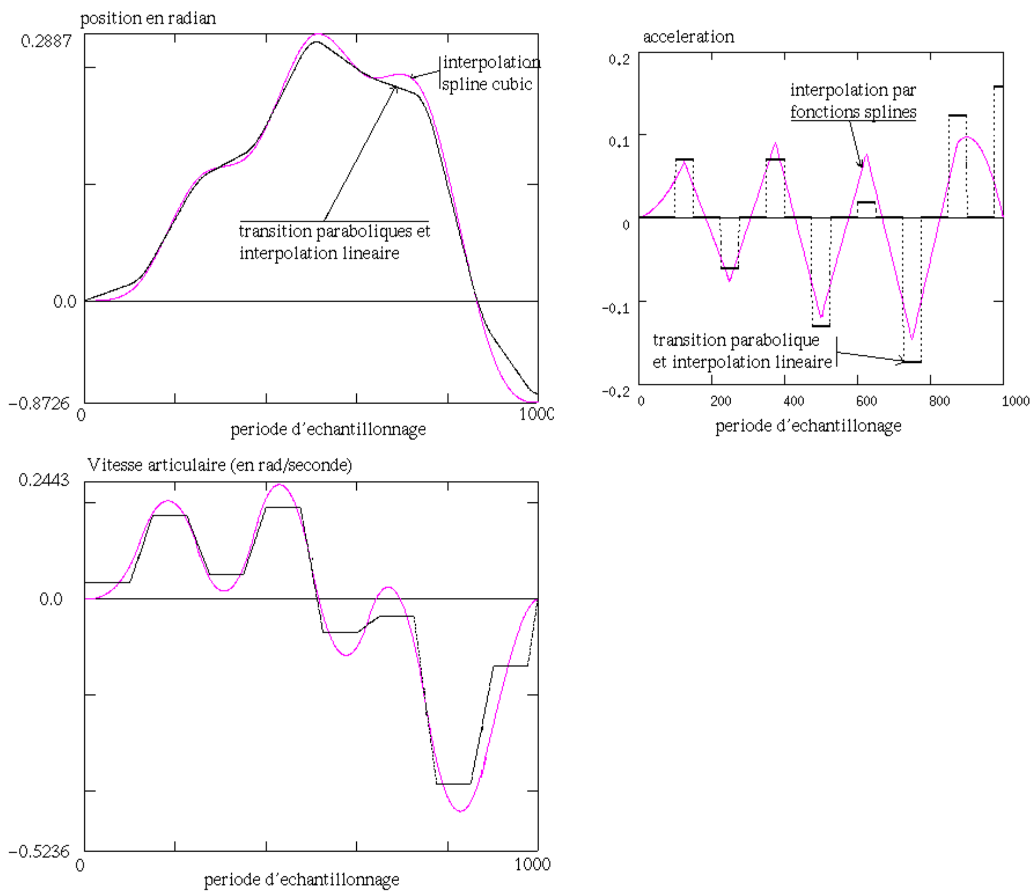

FIG. D.2 – *Exemple de courbes obtenues par les algorithmes : interpolation lineaire et transition parabolique et interpolation par fonctions splines.*

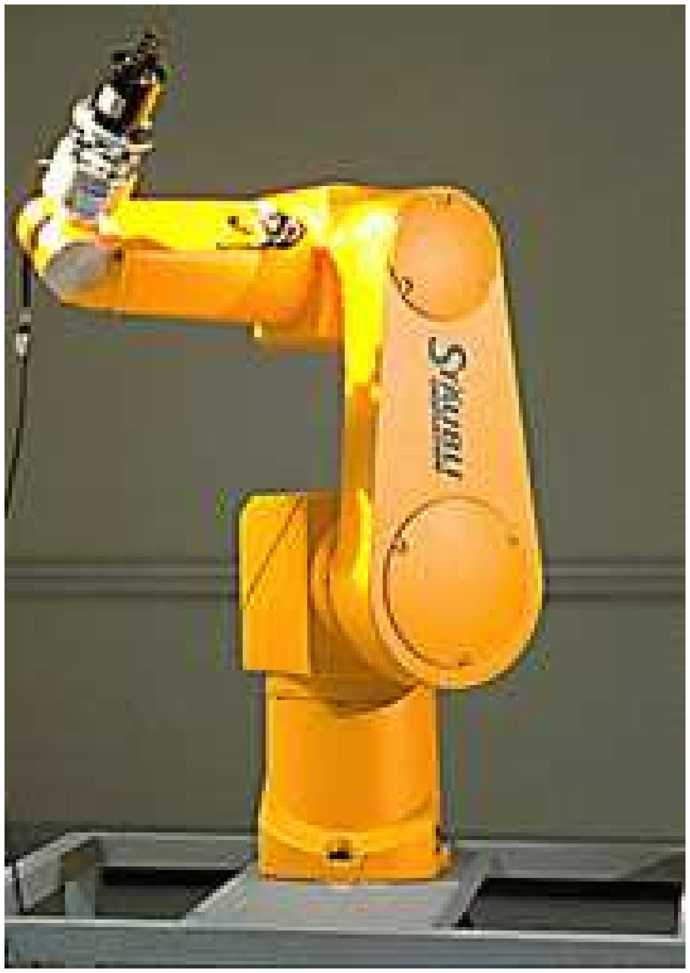

FIG. D.3 – *Le robot Staubli rx90.*

Annexe E

Partie économique

E.1 Raisons pour lesquelles l'INRIA développe ORCCAD

Le service robotique de l'INRIA participe au développement du logiciel ORCCAD pour disposer d'un environnement logiciel permettant la mise au point de contrôleur de robot pour les systèmes utilisés par les projets de robotique et vision.

E.2 Investissement total consacré au projet

E.2.1 Coût de l'investissement depuis le début du projet

Investissement en matériel

Le logiciel ORCCAD est un projet de recherche qui n'a nécessité aucun investissement en matériel. Les stations de travail utilisées sont les mêmes que celles mises à disposition des chercheurs et des ingénieurs. Les systèmes robotiques utilisés pour tester l'efficacité du logiciel appartiennent aux projets de recherche en robotique et vision, ils n'entrent pas dans l'investissement du projet ORCCAD.

Coût du travail des ingénieurs intervenant

Le développement du logiciel ORCCAD est le fruit d'un travail de recherche dont les premières réflexions ont commencées depuis 10 ans environ.

A mon arrivée à l'INRIA, sept personnes différentes ont travaillées sur le logiciel. La liste suivante présente la fonction et le coût de ces personnes :

- quatre thésards avec un salaire annuel estimé à 200KF pendant trois ans ;
- deux ingénieurs de recherche à 100% de leurs temps avec un salaire de 400KF/an pendant 2 ans ;
- 3 chercheurs à 25% de leurs temps avec un salaire identique à un ingénieur pendant 5 ans.

Le fait que les deux ingénieurs de recherche travaillent sur des sites différents (Montbonnot-Saint Martin et Sophia Antipolis) implique des frais de déplacement évalué à environ 50KF par an pendant 2 ans.

E.2.2 Coût de mon propre travail

Durant ce stage, mon salaire mensuel net était de 4081 francs. Le somme totale payée par l'INRIA est de 6681 francs par mois (1600 francs de charge) soit pour la durée totale du stage : 40 086 francs.

Ma présence à l'INRIA n'a pas nécessitée d'investissement particulier ni de coût de fonctionnement supplémentaire.

Le coût de l'encadrement a été évalué à 4% du temps de travail d'un ingénieur de recherche soit environ 8 heures par mois (4% de 200 heures). Ce coût s'élève donc à 4% de 400 KF multiplipar la durée du stage (en mois) soit un coût total de 24KF.

E.3 Impact économique du projet ORCCAD

Pour l'instant, le logiciel est mise à disposition pour une utilisation non commerciale, l'impact économique est donc nul.

E.4 Moyens mis en oeuvre pour la vente du logiciel ORCCAD

Il y a trois moyens qui ont été mis en oeuvre pour faire connaître le logiciel ORCCAD, ses trois moyens sont :

- des plaquettes de présentation ;
- des exposés ou des *posters* à des conférences robotiques internationales : SIROCCO 97 (Nantes) et IROS 97 (Grenoble) ;
- des publications dans des journaux spécialisés en robotique : *Robotica* et *International Journal of Robotics Research* ;
- un site internet dont l'adresse est : *http://www.inrialpes.fr/iramr/pub/Orccad/*

Le coût de ces moyens est négligeable par rapport à l'investissement en moyen humain.

La stratégie adoptée par les gens qui développent le logiciel est :

- mise à disposition *freeware* pour usage non commercial ;
- recherche de partenariats avec les *startups* de l'INRIA, notamment : Aleph Technologie, Verilog et Simulog.

E.5 Tableau récapitulatif

On remarque que le chiffre correspondant à l'investissement total (voir tableau E.1) s'élève à 4.64 millions de francs. Si l'INRIA décidait de lancer une *startup* pour commercialiser le logiciel ORCCAD, avec un prix de vente estimé à 10 KF, il lui faudrait trouver environ 275 acheteurs potentiels pour espérer rentrer dans ses frais. Elle pourrait alors faire des bénéfices sur la vente annuelle de la mise à jour, un prix de 5 KF lui apporterait un bénéfice d'environ 1.4 MF. Il faut enlever à ce chiffre les différents frais que doit payer une entreprise. il restera alors bien peu pour le salaire du ou des employés.

cout de l'investissement	materiel	humain	TOTAL
thesards		2.4 MF	
ingenieur		1.6 MF	
chercheurs		1.5 MF	
stagiaire		40 KF	
Diverse	100 KF		
TOTAL	100 KF	4.54 MF	4.64 MF

FIG. E.1 – *Tableau récapitulatif*

Bibliographie

[1] J. J. Borrelly and D. Simon, “Propositions d'architecture de contrôleur ouvert pour la robotique,” Research Report 1304, INRIA, Sophia-Antipolis, 1990.

[2] E. Coste-Manière, *Synchronisme et asynchronisme dans la programmation des systèmes robotiques: apport du langage* ESTEREL *et de concepts objets.* PhD thesis, Ecole de Mines de Paris, Juillet 1991.

[3] E. Dombre and W. Khalil, *Modélisation et commande des robots*, Hermès, Paris 1988.

[4] Edwall C.W., Pottinger H.J., Ho C.Y., *Trajectory generation and Control of robot using spline functions*, Robots conference, Detroit-Michigan, March 1982.

[5] Ilog Views Version 2.2, ILOG, *Programmer's guide and Reference manual.*

[6] A. Joubert, D. Simon : ORCCAD : *towards an Open Robot Controller Computer Aided Design system*, Rapport interne INRIA, Sophia Antipolis, Février 1991.

[7] K. Kapellos, *Environnement de Programmation des Applications Robotiques Réactives.* PhD thesis, Ecole des Mines de Paris, November 1994.

[8] K. Kapellos, R. Pissard-Gibollet: *Spécification - Vérification - Simulation d'une PrR*, Rapport interne INRIA, Mars 1997.

[9] Lin C.S., Chang P.R., Luh J.Y.S., *Formulation and optimization of cubic polynomial joint trajectories for industrial robots*, IEEE Transactions on automatic control, Vol. AC-28(12), december 1983

[10] R. Pissard-Gibollet, K. Kapellos: projet ORCCAD: *Environnement logiciel pour le contrôle-commande robotique, manuel d'utilisation d'ORCCAD version 3.0 alpha*, INRIA, Mars 1997.

[11] C. Samson, M. Le Borgne, B. Espiau, "*Robot Control: The Task Function Approach*". Oxford Engineering Science Series, Oxford University Press. Oxford, UK, 1991.

[12] D. Simon, B. Espiau, E. Castillo, and K. Kapellos, *Computer-aided design of generic robot controller h andling reactivity and real-time control issues*, IEEE Transactions on Control Systems and Technology, Fall 1993.

[13] VxWorks Version 5.3.2, Wind River, *Programmer's guide and Reference manual.*