



HAL
open science

Conception et implantation d'un système de programmation de robots

Jean-François Miribel

► **To cite this version:**

Jean-François Miribel. Conception et implantation d'un système de programmation de robots. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1984. Français. NNT : . tel-00312030

HAL Id: tel-00312030

<https://theses.hal.science/tel-00312030>

Submitted on 25 Aug 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Informatique»

par

Jean-François MIRIBEL



**CONCEPTION ET IMPLANTATION D'UN SYSTEME
DE PROGRAMMATION DE ROBOTS.**



Thèse soutenue le 27 octobre 1984 devant la commission d'examen.

G. VEILLON	Président
Mr. FORESTIER	} Examineurs
Mr. LATOMBE	
Mr. MAZER	
Mr. MUNTEAN	

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIÈRE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNÝ François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert
VACHAUD Georges

Directeur de Recherche
Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIEB Maurice
VINCENDON Marc

C.E.N.G. (STT)
C.E.N.G. (LETI)
C.E.N.G. (LETI)
C.E.N.G. (LETI)
C.E.N.G.
C.E.N.G.
C.E.N.G. (LETI)
C.E.N.G.
C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric
DEVINE
GERBER Roland
MERCKEL Gérard
PAULEAU Yves
GAUBERT C.

C.N.E.T.
C.N.E.T. (R.A.B.)
C.N.E.T.
C.N.E.T.
C.N.E.T.
I.N.S.A. Lyon

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

**Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES**

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M. MERMET
Directeur des Etudes et de la formation : Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie
---------	--------	-------------

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle
----------	--------------	--

REMERCIEMENTS

Pour moi, cette page, qui est la première, est aussi la plus agréable à écrire: c'est la dernière. Sans les personnes que j'ai le plaisir de remercier ci-dessous, cette page n'aurait pas lieu d'être: le reste n'existerait pas non plus.

Je tiens à remercier

Monsieur Gerard VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui a encouragé ces travaux, favorisé le développement de la Robotique au sein de l'ENSIMAG, et qui me fait l'honneur de présider le jury de cette thèse,

Monsieur Jean-Pierre FORESTIER, Directeur du CERT-DERA de Toulouse, pour l'intérêt qu'il a porté à ces travaux, et pour la collaboration qui s'est instaurée entre son Laboratoire et le Laboratoire LIFIA sur le projet LM,

Monsieur Jean-Claude LATOMBE, Professeur à l'Institut National Polytechnique de Grenoble, pour l'ensemble des connaissances que j'ai acquises grâce à lui, et pour la part prépondérante qu'il a prise dans ce travail,

Monsieur Trajan MUNTEAN, Maître-Assistant à l'Institut National Polytechnique de Grenoble, pour l'intérêt qu'il porte à ces travaux, et pour avoir bien voulu faire partie du jury de cette thèse,

Monsieur S.M. Emmanuel MAZER, Attaché de Recherches au CNRS, qui a ouvert la voie Intergalactique le 12 Janvier 1981, et sans lequel je ne saurais vraisemblablement pas encore ce qu'est réellement un robot.

Je remercie également Monsieur Georges GIRALT, Directeur de Recherches au LAAS de Toulouse et responsable du projet ARA, et Messieurs H.G.SAULET, Directeur de la société SCEMI, et J.SALEUR Directeur de la société MATRA-ROBOTRONICS, pour la part active

qu'ils ont eue dans le succès du projet LM, et pour la confiance qu'ils ont accordée à notre équipe.

Je tiens à exprimer ma gratitude envers tous ceux qui ont participé à ces travaux, et en particulier

Monsieur Michel DELAUNAY pour sa patience légendaire et sa disponibilité,

Messieurs Daniel BOISSON, Olivier FLACH, Emmanuel MAZER et Pierre MONTCUQUET pour les soirées fébriles passées à essayer de "faire tourner" le Number One,

Clarinettes, Calamity, Gaston Lagaffe, De Mesmaeker, Gus et Lolo pour la super ambiance de travail de l'équipe Spirou,

Madame Martine BLIN, Mademoiselle Sylvie LAFORGE, Messieurs Jean-Pierre BANSARD, Patrick BROYER, Bruno DUFAY, Jean-François GABARD, Christian GANDON, Pascal Di GIACOMO, François INGRAND, Jean-Marc LEFEBVRE, Patrick MEDIOLI, François MILIONI, et Marc PELTIER pour leur participation active à la réussite de ce projet LM.

Merci à Monsieur Christian LAUGIER pour son aide précieuse, et bonne chance ...

Enfin merci à mon épouse pour tout ce qu'elle a fait, à Brigitte et Manu HUMBERT-MAZER, à Catherine DUFAY et à Michel DESSIER pour m'avoir aidé à la réalisation de ce document.

Je dédie cette thèse à tibébé Julian et à sa mère.

Sommaire

INTRODUCTION: LA PROGRAMMATION DE ROBOTS

1. LE PROBLEME
 - 1.1. Objectif
 - 1.2. Nature du problème
 - 1.3. Illustration sur un exemple
2. DIFFERENTES APPROCHES POSSIBLES
 - 2.1. La programmation par l'exemple
 - 2.2. La programmation textuelle
 - 2.3. La programmation géométrique
 - 2.4. La programmation graphique
 - 2.5. La programmation automatique
3. TRAVAUX DU LIFIA
 - 3.1. Présentation
 - 3.2. Niveau manipulation
 - a. Programmation textuelle
 - b. Programmation par l'exemple
 - c. Simulation graphique
 - d. Programmation géométrique
 - e. Planification de trajectoires
 - 3.3. Le niveau tâche
4. NOTRE CONTRIBUTION ET PLAN DE LA THESE

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

1. INTRODUCTION
2. PRINCIPE DU LANGAGE LM
 - 2.1. LM est un nouveau langage
 - 2.2. LM repose sur un petit nombre de concepts de base
 - 2.3. LM est adapté à la technologie présente
 - 2.4. LM est un langage d'exécution
3. SUPPORT ALGORITHMIQUE
 - a) Objectif de la fonction

Sommaire

- b) Structures et instructions du langage LM
 - c) Discussion
4. MODELISATION DE L'UNIVERS
 - a) Objectif de la fonction
 - b) Structures et instructions du langage LM
 - c) Discussion
 5. COMMUNICATION AVEC L'ENVIRONNEMENT
 - a) Objectif de la fonction
 - b) Structures et instructions du langage LM
 - c) Discussion
 6. SPECIFICATION DES MOUVEMENTS
 - a) Objectif de la fonction
 - b) Structures et instructions du langage LM
 - c) Discussion
 7. SPECIFICATION DES OPERATIONS D'OUTILS TERMINAUX
 - a) Objectif de la fonction
 - b) Structures et instructions du langage LM
 - c) Discussion
 8. LE PARALLELISME ET LA SYNCHRONISATION
 - a) Objectif de la fonction
 - b) Structures et instructions du langage LM
 - c) Discussion
 9. EVOLUTIONS ACTUELLES DU LANGAGE
 - a) Les commandes gardées
 - b) Blocs parallèles

Sommaire

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

1. INTRODUCTION
2. COMPILATEUR ET EDITEUR DE LIENS
 - 2.1. Architecture du compilateur
 - 1- Un analyseur lexicographique
 - 2- Un module d'actions de compilation
 - 3- Un analyseur syntaxique
 - 4- Un module de recouvrement d'erreurs
 - 2.2. Format des instructions LM-E
 - 2.3. Editeur de liens
3. EXECUTIF LM
 - 3.1. Présentation
 - 3.2. Le séquenceur
 - 3.3. La tâche de fond
 - 3.4. Les tâches de surveillance de conditions d'arrêt
 - 3.5. Le chien de garde
4. PORTABILITE ET EXTENSIBILITE DU SYSTEME LM
 - 4.1. Définitions et présentation
 - 4.2. Portabilité du système
 1. Au niveau du calculateur hôte
 2. Au niveau du robot
 - a) RACORD niveau cartésien
 - b) RACORD niveau axes
 - 4.3. Extensibilité
 - a) L'interface MODFON
 - b) L'interface ESVOIE

Sommaire

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

1. INTRODUCTION
2. PRINCIPE DE L'ARCHITECTURE
3. MODULE DEPLACEMENT LIBRE
4. MODULE CHANGEUR DE COORDONNEES INVERSE NUMERO 1
5. MODULE TRAITEMENT VITESSE POSITION
6. MODULE CALCUL TEMPS DE PARCOURS
7. MODULE CALCUL PROFIL DE DEPLACEMENT
8. MODULE GENERATION DE TRAJECTOIRE
9. MODULE ANTICIPATION DYNAMIQUE
10. MODULE ASSERVISSEMENT
11. MODULE STOP LIBRE
12. MODULE DEPLACEMENT CARTESIEN
13. MODULE FORMATEUR DE TRAJECTOIRE
14. MODULE CHANGEUR DE COORDONNEES INVERSE NUMERO 2
15. MODULE CHANGEUR DE COORDONNEES DIRECT
16. MODULE STOP CARTESIEN
17. MODULE DEPLACEMENT VIA
18. MODULE SUIVI TRAJECTOIRE PLANIFIEE

CHAPITRE 4 : IMPLANTATIONS ET EXPERIMENTATIONS

1. INTRODUCTION
2. IMPLANTATIONS DE LM
 - 2.1. Laboratoires
 - 2.2. Industrie
3. EXPERIMENTATIONS
 - a) Montage d'un amortisseur
 - b) Insertion de composants électroniques
 - c) Réglage d'un moniteur de terminal alphanumérique
4. LECONS

Sommaire

CHAPITRE 5 : ENVIRONNEMENT DE LM

1. INTRODUCTION
2. INTERPRETEUR INTERACTIF
3. PROGRAMMATION PAR L'EXEMPLE
4. SIMULATION GRAPHIQUE
 - 4.1. Caractéristiques principales
 - 4.2. Composants du simulateur ISR
 - 4.3. Implantation
5. PROGRAMMATION GEOMETRIQUE
 - 5.1. Objectif
 - 5.2. Le langage LM-GEO
 - 5.3. L'interpréteur LM-GEO
6. PLANIFICATION DE MOUVEMENTS

CONCLUSION

BIBLIOGRAPHIE

ANNEXE

Exemple de programme LV

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

INTRODUCTION: LA PROGRAMMATION DE ROBOTS

1. LE PROBLEME

1.1. Objectif

Depuis 20 ans, l'industrie a développé des machines polyvalentes connues sous le nom de "robots industriels". Jusqu'à ces dernières années, ces machines étaient dépourvues de dispositifs sensoriels et n'utilisaient qu'un environnement informatique limité. Elles étaient de ce fait caractérisées par une quasi-incapacité d'adaptation aux variations de l'environnement.

La nécessité croissante d'automatiser des productions variées et complexes, l'abaissement du coût des matériels informatiques et l'apparition de besoins nouveaux tels que recherche spatiale, travaux sous-marins, sont à l'origine d'une nouvelle génération de robots, que l'on peut qualifier de "robots informatisés".

Un tel robot est une machine qui réunit:

- une mécanique motorisée (le manipulateur),
- une unité de contrôle (le calculateur),
- une interface avec l'univers physique (les capteurs).

Ces composants lui procurent une grande flexibilité d'utilisation:

- la généralité de la structure mécanique et des actionneurs le rend applicable à une large variété de tâches sans modifications matérielles majeures, ce qui permet d'envisager l'automatisation de productions en petite et moyenne série.
- l'équipement sensoriel, qui fournit des informations sur l'environnement réel, lui permet d'adapter son comportement à des variations de l'environnement relativement à un modèle donné, ce qui réduit le coût et la durée de la conception des équipements périphériques (alimentation des pièces, outillages, ...).

L'objectif de la programmation de robots est de permettre à son utilisateur de tirer le meilleur parti de cette flexibilité potentielle.

LE PROBLEME

1.2. Nature du problème

Programmer un robot consiste à spécifier les actions qui doivent être exécutées (mouvements des manipulateurs, opérations de l'outil terminal), l'utilisation des capteurs, et l'interaction entre les actions et la perception au sein d'un algorithme. Cet algorithme peut mettre en oeuvre des calculs complexes (typiquement des calculs géométriques), ainsi que des structures de contrôle élaborées incluant le parallélisme. En fait le problème de la programmation des robots est la réunion de plusieurs sous-problèmes très interdépendants. Deux sont particulièrement importants.

1) La spécification de positions sûres:

Un robot manipulateur est d'abord un dispositif de positionnement d'objets: il peut déplacer une pièce ou un outil d'une position à une autre. Ainsi, le coeur de la programmation des robots est de définir la séquence des positions du robot durant l'exécution d'une tâche. Ceci nécessite de définir les positions de prise et de lâcher des objets, les positions où l'on doit actionner des outils terminaux spécifiques, les trajectoires de transfert entre deux parties d'un poste de travail, etc...

Une difficulté de la spécification de positions est l'accessibilité: chaque position doit être accessible sans collision ni avec les objets de l'environnement, ni avec les butées des axes du robot, au moment où elle doit être atteinte.

2) La prise en compte de l'incertitude:

Ni l'environnement d'un robot, ni les actions de ce robot ne peuvent être prédites avec exactitude. L'incertitude géométrique n'est pas un problème tant qu'elle est inférieure à la précision requise par la tâche, par exemple le jeu entre deux pièces à assembler. Cependant, ceci nécessite un environnement soigneusement préparé, donc coûteux.

Une approche plus générale de la prise en compte de l'incertitude est l'interaction avec les capteurs. Ceci inclut par exemple l'inspection et la localisation de pièces avec un système de vision, et la localisation de points de contact par capteurs tactiles. Le programme de commande de robots devient alors l'implantation d'une stratégie faisant interagir l'action et la perception pour réduire

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

l'incertitude à une valeur inférieure à la précision requise.

Ces deux sous-problèmes interagissent de multiples façons. Par exemple, le calcul de trajectoires sûres nécessite de tenir compte de l'incertitude, notamment lorsque celle-ci est importante ou que l'environnement est fortement encombré d'obstacles. Réciproquement, le choix de trajectoires appropriées, en particulier la direction d'approche pour réaliser le montage de deux pièces, peut faciliter l'interprétation de données tactiles si un contact non souhaité se manifeste.

Un programme de robot doit constituer une solution à ces deux sous-problèmes, ainsi qu'à plusieurs autres: répartition des opérations entre plusieurs robots et coordination, maintien de l'équilibre des objets, ... De plus cette solution doit être acceptable relativement à des considérations de temps (durée d'exécution) et de fiabilité.

1.3. Illustration sur un exemple

A titre d'exemple, nous allons étudier la cellule d'assemblage expérimentale mise en place dans le cadre du Projet National ARA (Automatisation et Robotique Avancées). Cette cellule (Figure 1) réunit plusieurs systèmes d'amenée et d'évacuation de pièces, plusieurs systèmes de vision (noir et blanc, couleur, vision tridimensionnelle avec laser), des capteurs de force, de proximité, de contact, avec deux robots industriels à 6 degrés de liberté: un robot Renault-ACMA TH8 et un robot SCEMI.

La cellule est destinée à réaliser l'assemblage d'une grande variété de pièces. L'une des manipulations, prévue également dans le programme ARA, consiste à assembler un contacteur électrique (cf. Figure 2). Un scénario pour réaliser une partie de cet assemblage peut être:

- Détecter l'arrivée d'une pièce sur le tapis roulant, la reconnaître, contrôler sa qualité, calculer sa position et son orientation.
- Prendre la pièce au vol sur le tapis avec l'un des deux robots, s'assurer que la prise est bonne et qu'il n'y a pas de glissements.

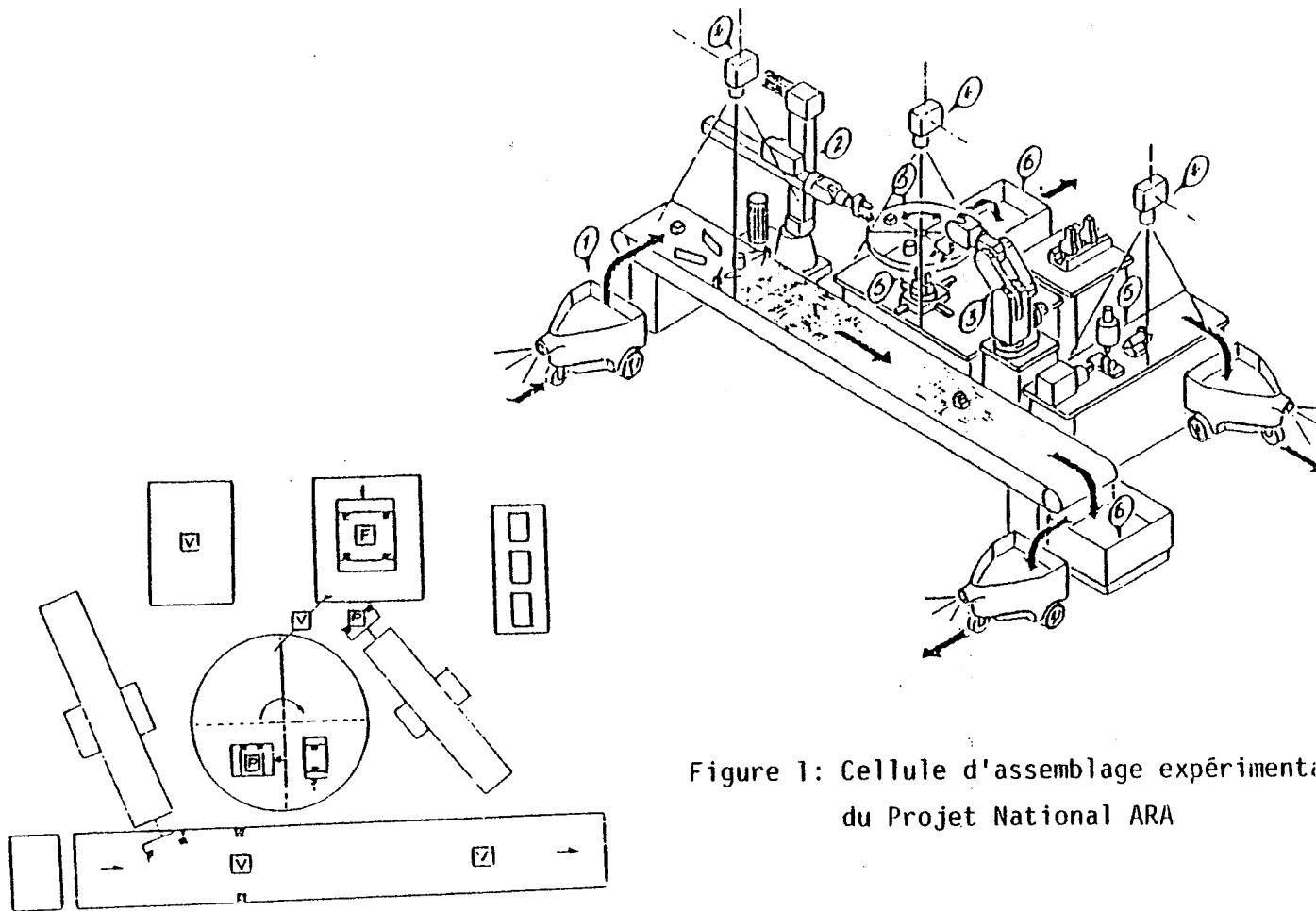
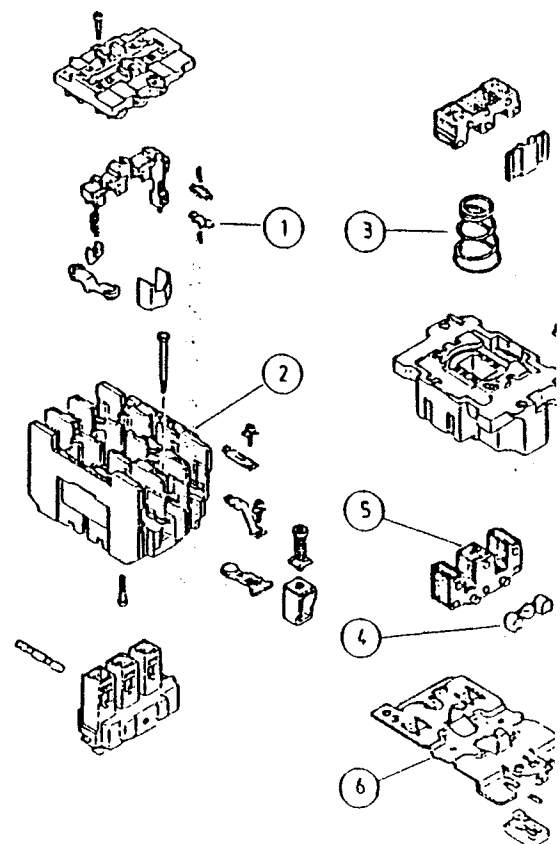


Figure 1: Cellule d'assemblage expérimentale du Projet National ARA

Figure 2: Eclaté d'assemblage d'un contacteur



INTRODUCTION : LA PROGRAMMATION DE ROBOTS

- Ranger la pièce sur la table d'assemblage ou, si c'est possible, assembler directement la pièce sur un sous-assemblage en cours (suivant le degré de difficulté, l'opération d'assemblage pourra se faire avec capteur de force ou de vision).
- Une fois un assemblage terminé, le diriger vers l'un des dispositifs d'évacuation.

Cet exemple met en évidence quelques difficultés que l'on rencontre pour programmer ce type d'application:

- Le système de commande des robots doit être relié au système de vision et mettre à jour les paramètres transmis par la vision en fonction des référentiels de travail des robots et de l'avancement du tapis roulant (nécessité de calibrage et de synchronisation).
- "S'assurer que la prise est bonne" implique la présence d'un capteur de pression ou d'une peau sensible sur les mors de la pièce, ainsi que des algorithmes et une électronique rapides pour réagir à des glissements d'objets.
- Le positionnement d'une pièce sur un sous-assemblage avec exactitude, même à l'aide d'un capteur de force, est difficile également à programmer: il faut un algorithme sûr, convergent et rapide.
- Le processus de décision doit apporter la flexibilité voulue en choisissant le robot le plus apte à réaliser une opération, en fonction de leurs travaux respectifs, et en prenant en compte les risques de collision.

2. DIFFERENTES APPROCHES POSSIBLES

La définition d'un système de programmation de robots doit prendre en compte deux exigences:

- (1) Le système doit être facile à utiliser , de façon à être accessible à des opérateurs sans longue formation.
- (2) Il doit posséder un pouvoir d'expression élevé de façon à permettre la description de tâches complexes.

DIFFERENTES APPROCHES POSSIBLES

Ces deux exigences, en apparence contradictoires, ont motivé plusieurs approches de la programmation des robots qui ont débouché sur différentes classes de systèmes.

2.1. La programmation par l'exemple

La méthode de programmation la plus ancienne, et aujourd'hui la plus répandue, est appelée programmation "par l'exemple" ou "par apprentissage". Elle consiste à exécuter une fois la tâche en contrôlant le robot à l'aide d'un dispositif de télécommande (palonnier, pantin). Les mouvements de tous les actionneurs sont enregistrés et peuvent être ensuite reproduits automatiquement. Les aides aux programmeurs incluent des outils de télécommande avec retour d'effort, et des fonctions spécialisées pour exécuter des trajectoires rectilignes ou circulaires.

Cette méthode de programmation est attrayante dans son principe:

- elle exige peu de connaissances spécialisées de l'opérateur (mais toutefois une certaine habileté),
- elle permet à l'opérateur de voir immédiatement les effets de sa programmation,
- elle ne nécessite qu'un support informatique léger.

Elle présente cependant des inconvénients importants:

- il est impossible d'utiliser des données en provenance de capteurs (pas d'adaptation à l'environnement),
- il est impossible de coordonner plusieurs robots,
- il est difficile de modifier les "programmes" enregistrés,
- il n'y a pas ou peu de possibilités d'itérations, ni d'instructions conditionnelles,
- le poste de travail est immobilisé durant la phase de programmation.

Ces inconvénients résultent de ce que la programmation par l'exemple ne donne accès qu'à un sous-ensemble des fonctionnalités du robot. Autrement dit, elle privilégie la facilité de la programmation au détriment du pouvoir d'expression.

La programmation par l'exemple est de ce fait restreinte à des tâches où répétabilité et tolérance sont compatibles, où la diversité des opérations est faible, et où les cycles d'opérations sont courts. Son domaine d'application inclut donc assez naturellement la projection d'enduits (peinture

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

notamment) et le chargement-déchargement de pièces. Par contre, il est pratiquement impossible de réaliser la programmation d'assemblages mécaniques par cette méthode. En effet, pour ces tâches, il est souvent préférable de faire dépendre les actions des robots de données sensorielles et de faire travailler plusieurs robots simultanément.

Quelques systèmes industriels récents de programmation par l'exemple visent à réduire les inconvénients ci-dessus par l'introduction de facilités nouvelles (utilisation de capteurs, possibilité de branchement conditionnel). Ces facilités, empruntées à l'approche des langages textuels (cf. paragraphe suivant), restent très limitées.

2.2. La programmation textuelle

Les limitations de la programmation "gestuelle", c'est-à-dire par l'exemple, ont conduit au développement d'une programmation dite "textuelle" faisant appel à des langages symboliques. Contrairement à la programmation par l'exemple, la programmation textuelle s'est d'abord développée dans des laboratoires (cf. WAVE [Paul 77] et AL [Finkel 74]), et a fait son entrée dans l'industrie plus récemment.

Les langages de programmation développés combinent les facilités algorithmiques de langages tels que BASIC, PASCAL ou LISP dans une syntaxe analogue avec des constructions appropriées à la Robotique. Ces constructions concernent notamment:

- la modélisation de l'univers en termes de repères cartésiens, indépendants de l'architecture mécanique des robots,
- la communication avec l'environnement des robots (perception notamment),
- la description des mouvements des robots et des actions de leurs outils terminaux,
- le contrôle de processus parallèles: coordination de plusieurs robots, synchronisation avec des dispositifs tels que systèmes d'alimentation de pièces.

DIFFERENTES APPROCHES POSSIBLES

L'intérêt de ce mode de programmation est de donner un large accès aux fonctions de base des robots, donc d'avoir un pouvoir expressif élevé. Toutefois, bien qu'il ait aujourd'hui un développement industriel important, on le considère encore souvent accessible seulement à des utilisateurs expérimentés. En fait, nous verrons que les langages textuels permettent assez simplement d'implanter des utilitaires de programmation par l'exemple utilisables par des opérateurs ignorant tout de la programmation symbolique. Plus généralement, l'emploi de langages textuels peut être largement facilité par des environnements de programmation appropriés. Il n'en reste pas moins que l'accès à certaines fonctions telles que l'interaction avec les capteurs et la coordination de robots continue à nécessiter une certaine expertise en programmation.

2.3. La programmation géométrique

La programmation géométrique peut être considérée comme une extension de la programmation textuelle introduite au paragraphe précédent. Son but principal est de faciliter les raisonnements du programmeur sur l'environnement tridimensionnel des robots.

La description de l'environnement en termes de repères cartésiens a de nombreux avantages quant à l'implantation des langages de programmation: concision des représentations, simplicité des algorithmes [Paul 81]. Cependant, pour un humain, il n'est pas toujours facile de conduire son raisonnement sur des entités aussi abstraites que les repères, que ce soit pour écrire ou pour comprendre des programmes.

La programmation géométrique permet à l'utilisateur de décrire les positions des robots et des pièces par des relations géométriques explicites entre entités (par exemple: faces, arêtes) des objets concernés. L'interpréteur du langage utilise une base CAO de modèles géométriques pour traduire ces relations en termes de transformations entre repères.

La programmation géométrique, introduite par le langage RAPT [Popplestone et al 78], n'a connu qu'un assez faible développement, et n'a pas encore fait son entrée dans l'industrie. L'existence de plus en plus répandue de systèmes de CAO pourrait modifier cette situation.

2.4. La programmation graphique

En première approximation la programmation graphique n'est qu'une variante de la programmation par l'exemple: l'opérateur guide un robot simulé sur terminal graphique à travers une séquence de mouvements et d'opérations. Cette séquence est ensuite reproduite automatiquement soit sur le robot simulé (pour contrôle), soit sur le robot réel [Liégeois et al 82], [Laugier 84b].

La difficulté de guider interactivement un robot simulé sur un écran bi-dimensionnel, que ce soit à l'aide d'un boîtier de commande ou avec des touches de fonctions, a conduit à incorporer à la programmation graphique des notions rappelant la programmation géométrique. On ne guide plus alors le robot simulé comme un robot réel, mais on définit les positions clés par des relations géométriques simples [Liégeois, Borrel et Dombre 84]. L'opérateur dispose d'un jeu de commandes élémentaires telles que "déplacer le POINT A d'un objet attaché au robot SUR le POINT B d'un objet fixe". Ces commandes sont données à l'aide d'un menu et d'un dispositif d'interaction graphique du type tablette à digitaliser qui permet de désigner les entités géométriques concernées (point, ligne,...).

La programmation graphique possède les qualités de la programmation par l'exemple: l'opérateur voit immédiatement les effets de sa programmation et n'a à manipuler que des concepts de programmation simples. Elle offre en plus quelques avantages: plus grande sécurité pour l'opérateur et le matériel, possibilité de travailler avec des objets (robots, pièces manipulées, environnements) non encore existants, non immobilisation du matériel de production. Elle nécessite toutefois l'existence d'un système de CAO, ou tout au moins d'un système de modélisation. La programmation graphique présente également toutes les limitations, quant au pouvoir d'expression, inhérentes à la programmation par l'exemple.

DIFFERENTES APPROCHES POSSIBLES

2.5. La programmation automatique

Les approches de la programmation des robots présentées dans les paragraphes précédents conduisent à des systèmes dits de niveau "manipulation" ou "effecteur" [Latombe 79], car ils nécessitent de décrire explicitement les mouvements et les opérations des robots, ainsi que l'interaction avec les capteurs. Ces approches ont tendance soit à favoriser la simplicité de la programmation au détriment du pouvoir d'expression, soit l'inverse. Bien entendu cette vue est un peu schématique, et, comme nous le verrons, il est possible de combiner certains avantages des différentes approches. Il n'en reste pas moins que la programmation de niveau manipulation reste souvent délicate.

La programmation automatique des robots a pour objectif de traduire une description de haut niveau d'une tâche en un programme de niveau manipulation. La description fournie en entrée spécifie une séquence de situations "objectifs" exprimées indépendamment des robots. Par exemple, dans le cas de l'assemblage mécanique, elle est typiquement constituée des modèles CAO des objets et des relations d'assemblage entre ces objets. Cette approche de la programmation est souvent dite de niveau "tâche" ou "objet" [Latombe 79]. Elle vise à combiner simplicité et pouvoir d'expression, en augmentant considérablement la tâche de l'interpréteur du langage, qui doit alors faire le travail du programmeur. Plusieurs systèmes de niveau tâche ont été proposés [Lieberman et Wesley 77], [Lozano-Perez 76], [Lozano-Perez et Brooks 84] mais ils n'ont donné lieu au mieux qu'à des implantations partielles. La recherche sur ce sujet est très active [Latombe 83] [Lozano-Perez 83] et devrait fournir rapidement soit des outils interactifs d'aide à la programmation de robots, soit des systèmes plus complets dans des domaines limités (par exemple la fabrication de cartes électroniques).

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

3. TRAVAUX DU LIFIA

3.1. Présentation

L'équipe Intelligence Artificielle et Robotique du LIFIA a commencé ses travaux en Robotique en 1977. La motivation initiale était de développer un support expérimental de recherche en Intelligence Artificielle sur des sujets tels que la génération de plans d'actions, l'interaction de la perception et de l'action, l'apprentissage, le raisonnement spatial [Mazer 81]. Le principal objectif de ces travaux était l'automatisation de la programmation des robots, c'est-à-dire le développement d'un système de niveau tâche [Latombe 82a]. Pour l'atteindre, il était nécessaire de disposer d'un langage de programmation cible, de niveau manipulation. Aucun n'étant disponible à cette époque, l'équipe a décidé de développer son propre langage de niveau manipulation. Les travaux de l'équipe du LIFIA dans le domaine de la programmation des robots concernent donc les deux niveaux introduits au paragraphe précédent: le niveau manipulation et le niveau tâche.

3.2. Niveau manipulation

L'objectif initial de l'équipe était seulement de développer un langage cible pour la programmation automatique. Les besoins de la recherche et des coopérations avec plusieurs industriels ont conduit à réviser considérablement cet objectif [Latombe et al 84]. D'un langage cible, on est passé à un système complet combinant un langage de programmation textuel et un environnement de développement et de mise au point de programmes. Les principaux éléments de ce système sont les suivants :

a. Programmation textuelle

Le coeur du système est le langage de programmation LM, qui est un langage textuel. Il combine des constructions algorithmiques conventionnelles avec des facilités spécifiques à la programmation de robots: modélisation de l'univers du robot, communication avec l'environnement (incluant les capteurs), spécification de mouvements du manipulateur et d'opérations des outils, et outils de

TRAVAUX DU LIFIA

contrôle de processus.

L'exécutif du langage fonctionne soit en mode interactif (les instructions données au terminal sont exécutables immédiatement), soit en mode compilé (tous les programmes sont compilés dans un code intermédiaire et reliés ensemble avant l'exécution) [Mazer 81] [Latombe et Mazer 81c] [Latombe, Mazer et Miribel 81b] [Miribel et Mazer 82] [Mazer et Miribel 84a] [Latombe et al 84].

b. Programmation par l'exemple

Le langage LM a été utilisé pour développer de nombreux utilitaires de programmation par l'exemple. L'un d'eux, LM-EX, est destiné à l'assistance à la programmation pour des tâches d'assemblage complexes. Il génère des programmes LM textuels dans des fichiers, qui peuvent par la suite être édités, modifiés, et combinés avec d'autres programmes LM [Bansard 83].

c. Simulation graphique

L'exécutif du langage LM peut contrôler soit des robots réels, soit des robots simulés. Le simulateur graphique ISR inclut des fonctions permettant de se rapprocher du monde réel (incertitudes, collisions). Il peut être utilisé soit pour valider des programmes existants, soit pour en générer de nouveaux en utilisant les utilitaires de programmation par l'exemple [Laugier 83], [Laugier et Pertin 84c], [Laugier, Evieux et Pertin 84a].

d. Programmation géométrique

Pour programmer en langage LM, on modélise l'espace en termes de repères cartésiens. LM-GEO est une extension de LM, qui utilise des modèles géométriques d'objets issus de systèmes de CAO. Il permet au programmeur de spécifier les positions des objets par des relations géométriques explicites. Les instructions en LM-GEO sont converties en LM par un logiciel qui déduit les positions des repères à partir des relations géométriques données [Miribel 81] [Mazer 82a] [Mazer 82b] [Mazer 83].

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

e. Planification de trajectoires

L'exécutif LM calcule et échantillonne des mouvements droits et circulaires en temps réel. Un module séparé accepte des descriptions de trajectoires plus complexes, définies par des équations. Il génère des tables de points en coordonnées cartésiennes relatives, sous une forme directement utilisable dans des programmes LM.

La Figure 3 illustre l'utilisation typique de ces facilités. Le programmeur développe un nouveau programme en utilisant l'interpréteur du langage LM en mode interactif, avec l'assistance d'outils tels que les utilitaires de programmation par l'exemple, le simulateur graphique, LM-GEO et le planificateur de mouvement. Une fois créé et mis au point sur simulateur, le programme est compilé, puis exécuté en utilisant l'interpréteur en mode compilation.

Le système LM (ou des parties de ce système) ont fait l'objet de multiples expérimentations, certaines peu conventionnelles: montage d'un mécanisme d'essuie-glaces, montage d'un amortisseur de Peugeot 505, insertion de composants électroniques non standard sur une carte, réglage de l'électronique d'un moniteur de terminal alphanumérique. Depuis 1983, l'état conceptuel du système s'est stabilisé. L'effort de recherche le concernant au sein du LIFIA s'est réduit (une partie du système ayant été transféré dans l'industrie) au profit des travaux sur la programmation de niveau tâche.

3.3. Le niveau tâche

L'élaboration d'un système de niveau tâche correspondait en 1977 à un objectif à long terme. Plûtôt que d'aborder de front tous les problèmes posés par la conception d'un système complet, l'équipe Intelligence Artificielle et Robotique a concentré ses efforts sur des problèmes mieux délimités, à savoir:

- le choix des prises d'un objet.

Un programme de calcul automatique des prises d'un objet en fonction de la géométrie de cet objet et de son environnement a été conçu et réalisé. La méthode utilisée

TRAVAUX DU LIFIA

dans ce programme opère en deux phases: d'abord calcul de prises possibles par utilisation d'indices morphologiques locaux, puis élimination de certaines prises par l'application de contraintes globales d'accessibilité. La méthode utilise des indices morphologiques relativement nombreux et élaborés, ce qui lui permet de traiter des objets de complexité moyenne.

Le programme implantant cette méthode a été écrit en FORTRAN (phase 1) et en LISP (phase 2). Il a été expérimenté en simulation sur des objets composés de polyèdres et de cylindres [Laugier 81] [Laugier 83] [Laugier et Pertin 83].

- la génération de trajectoires sans collision.

Le problème du calcul de trajectoires sans collision a été abordé en reprenant la méthode dite "par grossissement d'obstacles" de Lozano-Perez. Un algorithme pour appliquer cette méthode dans l'espace tridimensionnel a été défini et implanté, les objets ne se déplaçant qu'en translation. L'expérimentation de cet algorithme a conduit à jeter les bases d'une méthode nouvelle adaptée au traitement des manipulateurs à articulations rotoïdes [Germain 84].

- la synthèse des mouvements fins d'assemblage.

Une méthode de synthèse automatique des mouvements fins fondée sur l'apprentissage par induction à partir d'exemples a été développée. Cette méthode opère en deux phases. Lors de la première phase, un programme contrôle le robot et décide de ses actions en interaction directe avec les données transmises par les capteurs. Ce programme, qui prend ses décisions sur la base de connaissances expertes fournies par l'utilisateur, engendre ainsi plusieurs traces d'exécution pour une même opération de montage. Chaque trace correspond à une exécution différente, et deux traces différent généralement à cause des variations de l'imprécision d'une exécution à une autre. La seconde phase de la méthode travaille hors ligne. Elle prend en entrée des traces d'exécution, compare et combine ces traces, en généralise certaines parties par des règles d'induction, et fournit en sortie un programme en LM adapté à l'opération de montage. La méthode complète a été implantée en LISP sur un calculateur CII-HB-70 connecté à un micro-calculateur LSI-11/23 pilotant un robot ROBITRON à 4 degrés de liberté muni de capteurs de forces. Elle a été expérimentée avec succès sur plusieurs exemples [Dufay 83a] [Dufay et Latombe 83b] [Dufay et Latombe 83c].

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

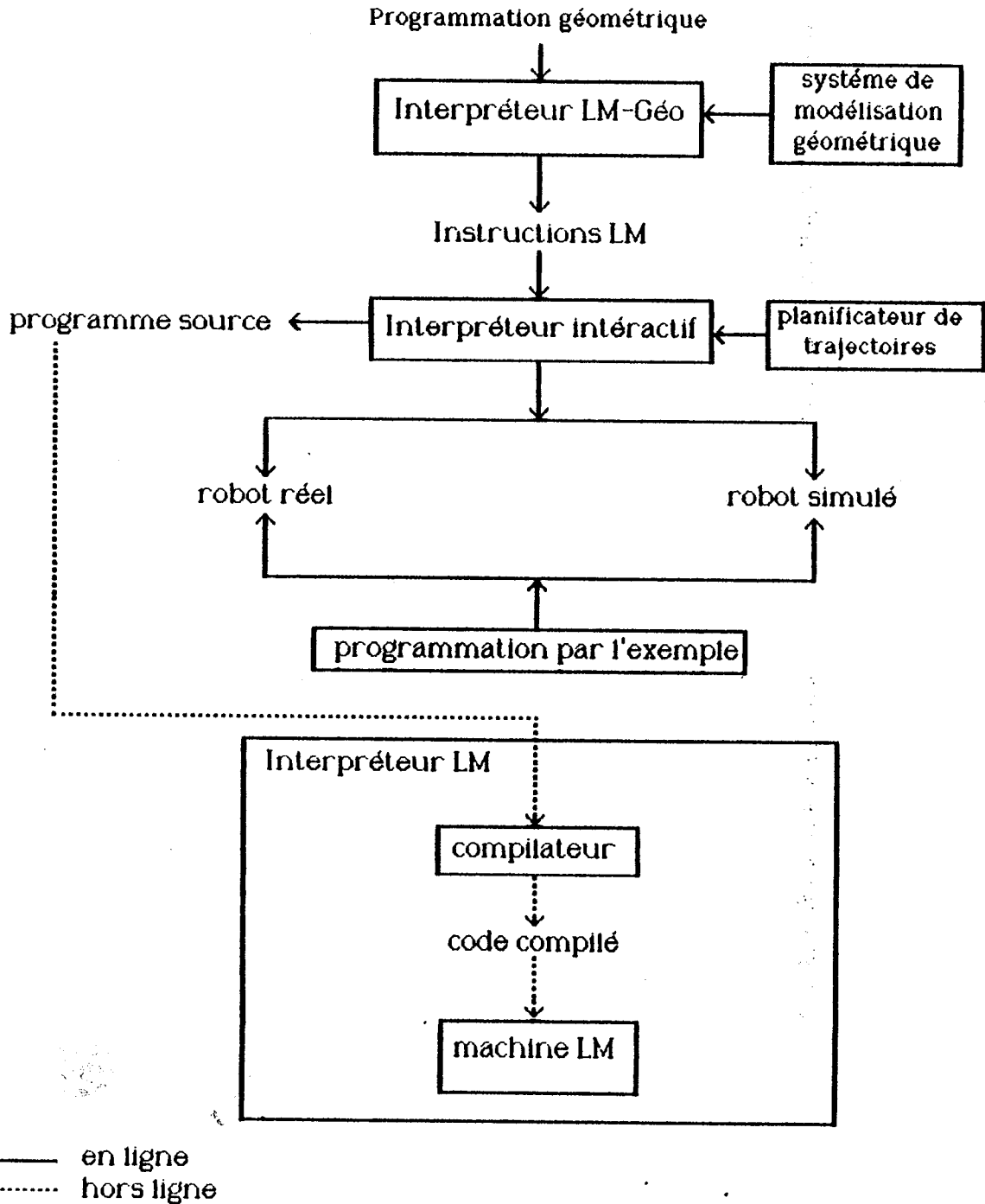


Figure 3: Présentation générale du système LM

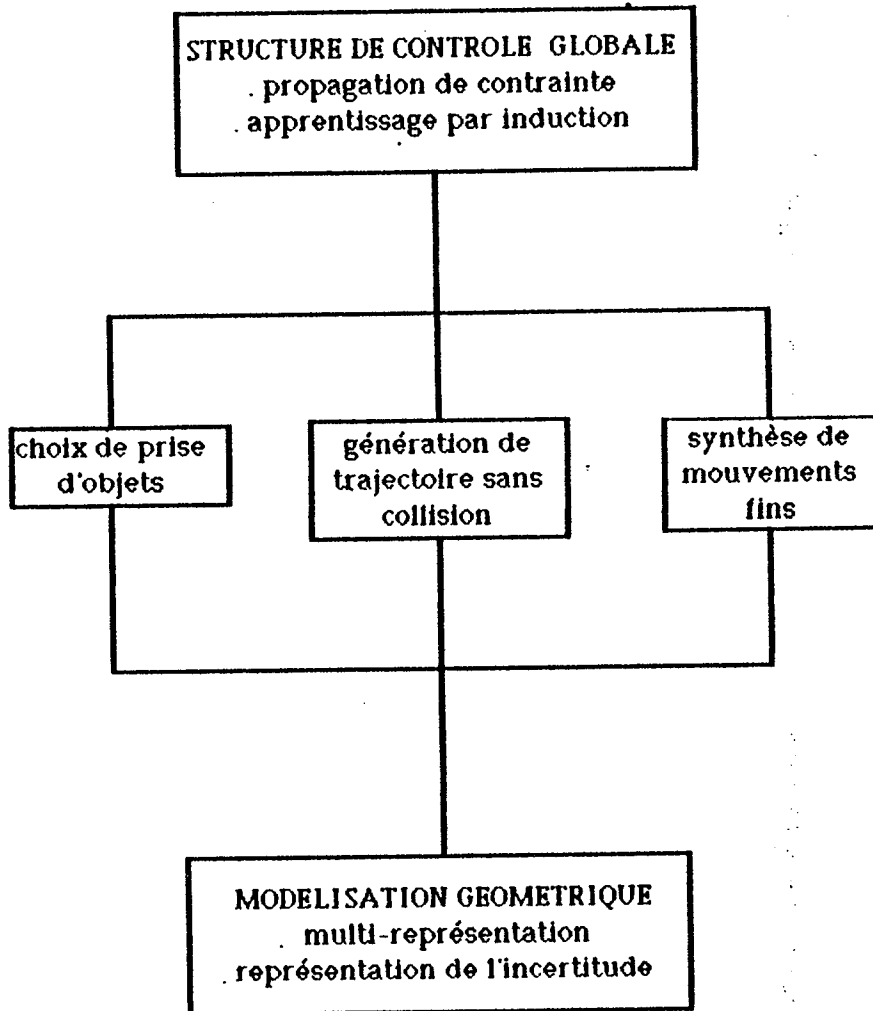


figure 4: structure du Système prototype niveau tâche

INTRODUCTION : LA PROGRAMMATION DE ROBOTS

Aujourd'hui les travaux de l'équipe visent à intégrer les trois modules ainsi développés au sein d'un système prototype de niveau tâche, dont la structure est montrée à la Figure 4. Cet effort d'intégration a déjà conduit à l'étude et à la réalisation partielle d'un support de modélisation géométrique répondant aux besoins des trois modules: modélisation d'objets articulés, représentations multiples d'un même objet, représentation de l'incertitude [Pertin-Troccaz 84].

4. NOTRE CONTRIBUTION ET PLAN DE LA THESE

Nous avons commencé nos travaux dans l'équipe du LIFIA en 1979. Le sujet de notre DEA a porté sur la définition et la réalisation d'un langage de programmation de robots à partir de relations géométriques, le langage LRG [Miribel 81], implanté en LISP. C'est à la suite de ce DEA, c'est-à-dire en 1980, qu'a commencé notre participation au "projet LM".

Les premières spécifications de LM avaient été réalisées dès 1978 [Latombe 79], ainsi qu'une première version de l'exécutif [Mazer 81]. Notre premier travail a consisté en l'écriture du compilateur et de l'éditeur de liens du langage ; ceci a nécessité la reprise partielle et l'ajout de constructions syntaxiques au langage. Nous avons également été amenés à réaliser une nouvelle version de l'exécutif LM pour prendre en compte ces modifications.

Ces réalisations ont été faites initialement dans le cadre d'un contrat avec la société SCEMI, par laquelle nous avons été employés durant deux ans (1980-1982), et ont donné lieu à la première implantation industrielle de LM. Depuis fin 1982, nous sommes employés par la société ITMI, qui a pris en charge la commercialisation et la maintenance du système LM.

Depuis nos débuts dans le projet LM, le langage et son environnement ont considérablement évolué (cf. paragraphe précédent). De par nos travaux sur le coeur du système qu'est l'interpréteur, nous avons collaboré de près ou de loin à tous les travaux réalisés autour de LM au sein de l'équipe durant cette période.

Cette thèse, qui présente notre contribution au développement de LM, comporte 5 chapitres:

NOTRE CONTRIBUTION ET PLAN DE LA THESE

- Le premier chapitre présente LM à travers les fonctions d'un langage de programmation de robot.
- Le second chapitre détaille la structure de l'interpréteur LM, et des différents modules qui le composent, hormis les modules de génération des mouvements.
- Le troisième chapitre montre la structure multi-tâches que nous préconisons pour gérer les calculs et contrôles nécessaires à la génération de mouvements d'un manipulateur.
- Le quatrième chapitre détaille les différentes implantations de LM et quelques expérimentations réalisées en LM.
- Le dernier chapitre, enfin, présente l'environnement actuel du langage et les développements en cours pour parfaire cet environnement.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

1. INTRODUCTION

Dans ce chapitre, nous présentons les principaux concepts du langage LM. Notre but n'est pas de faire une description exhaustive et détaillée du langage. Le lecteur pourra trouver une telle description dans le Manuel de Référence [Mazer et Miribel 84a].

Nous commençons (paragraphe 2) par énoncer les grands principes qui ont présidé à la définition du langage LM. Ensuite (paragraphe 3 à 8), notre présentation est structurée par grandes fonctions.

Les fonctions considérées recouvrent l'ensemble des facilités offertes par les langages de programmation de robots actuels, ce qui ne signifie pas que LM regroupe toutes ces facilités, mais qu'il apporte une solution (au moins partielle) à chaque fonction.

Ces fonctions sont [Lozano-Perez 82]:

- le support algorithmique,
- la modélisation de l'univers,
- la communication avec l'environnement,
- la spécification des mouvements de robots,
- la spécification des opérations des outils,
- le parallélisme et la synchronisation.

Pour chaque fonction, nous adoptons le même plan: nous introduisons d'abord l'objectif de la fonction; ensuite nous présentons les structures et instructions LM qui lui correspondent; enfin, nous discutons les solutions apportées par LM, dans l'absolu et en relation avec d'autres langages de programmation de robots.

Un nombre assez impressionnant de langages de programmation de robots ont été conçus. Nous pouvons citer notamment AL [Finkel 74], AML [Taylor 82], VAL [Unimation 79], VAL-II [Shimano 84], RAIL [Automatix], Sigla [Olivetti], T3 [Cincinatti], LPR [LPR 81], SRL [Blume 82], PASRO [Biomatik], ROBEX, Lucifer [Giraud], RAPT [Popplestone 78], HELP, RCL, ROL [Gixi], PAL, LAMA_S [Falek 80], MAL [Gini 79], RCCL [Hayward 84], ... Cependant, relativement peu de ces langages ont donné lieu à une implantation sérieuse, ou

INTRODUCTION

même à une implantation tout court. De plus, plusieurs langages n'ont été développés qu'à des fins expérimentales, et ont aujourd'hui disparu, dans certains cas après avoir donné lieu à un successeur. C'est pourquoi nos discussions des solutions apportées par LM en relation avec d'autres langages sont volontairement concises et, pour la plupart, limitées aux langages actuels les plus importants (typiquement AL, AML et VAL-II).

2. PRINCIPE DU LANGAGE LM

Nous exposons dans ce paragraphe les grands principes qui ont présidé à la conception du langage LM:

2.1. LM est un nouveau langage

Il y a deux approches principales à la conception d'un langage textuel de programmation de robots. L'une consiste à étendre un langage existant par une bibliothèque de procédures (par exemple: RCCL est une extension du langage C; PASRO est une extension de PASCAL). L'autre est de développer un nouveau langage; par exemple: AL, AML, même si ces derniers s'inspirent pour une part de langages traditionnels (par exemple AL est inspiré d'ALGOL).

Dans LM nous avons choisi la seconde approche. Un avantage potentiel est de rendre les programmes plus lisibles grâce à une syntaxe appropriée. Un autre avantage est de permettre la détection d'erreurs de programmation lors de la phase d'analyse syntaxique, ce qui procure une plus grande sécurité lors de l'expérimentation des programmes.

Une approche intermédiaire consiste à utiliser un langage extensible, par exemple LMAC [Rousseau 83]. L'expérience montre toutefois qu'un tel langage ne permet pas d'engendrer toutes les constructions syntaxiques souhaitées.

2.2. LM repose sur un petit nombre de concepts de base

Pour faire des avantages potentiels énoncés au paragraphe ci-dessus une réalité, nous avons choisi de bâtir LM sur un relativement petit nombre de concepts de base, des

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

constructions syntaxiques expressives permettant de combiner ces concepts de multiples façons. Ainsi, le programmeur débutant peut rapidement réaliser des applications simples; ensuite, au fur et à mesure que ses compétences augmentent, il peut trouver dans le langage les options adaptées à ses nouveaux besoins.

Ce second principe démarque assez nettement LM de langages tels que VAL-II et AML, qui tendent à fournir de nombreuses instructions adaptées chacune à un concept relativement élaboré.

2.3. LM est adapté à la technologie présente

LM a été défini en vue d'être complètement implanté. De ce fait, les choix effectués l'ont été en considérant les possibilités de la technologie du matériel informatique du début des années 80 en relation avec les contraintes de temps réel imposées par la commande des robots. Ceci explique par exemple que LM n'ait pas de constructions permettant d'exprimer directement un mouvement "compliant".

La technologie évoluant, de nouvelles composantes seront rajoutées au langage. Ainsi ont été définies des facilités algorithmiques permettant un parallélisme d'exécution plus grand dans les programmes LM [Lefebvre 82]; ces facilités sont actuellement en cours d'implantation sur le système multi-processeur CLEOPATRE du CERT (multi-68000 avec bus VME modifié).

2.4. LM est un langage d'exécution

Nous pouvions concevoir LM soit comme un langage d'exécution (c'est-à-dire uniquement pour définir des programmes de commande de robots), soit comme un langage d'exécution et d'implantation (c'est-à-dire aussi pour réaliser les modules de programmation automatique nécessaires à un système de programmation de niveau tâche).

Nous avons choisi de faire de LM uniquement un langage d'exécution. La réalisation d'un système de niveau tâche est donc indépendante de LM. Seul le code engendré par ce système est composé d'instructions LM. Ce choix a pour intérêt de permettre une certaine simplification des structures de données du langage LM. Il est cohérent avec le principe 3.

SUPPORT ALGORITHMIQUE

3. SUPPORT ALGORITHMIQUE

a) Objectif de la fonction

En général un programme de commande de robot est loin d'être une simple séquence d'instructions décrivant les mouvements et opérations du robot. Il a souvent à effectuer des traitements complexes d'informations, dont la nature n'est pas spécifique de la Robotique.

Par exemple, dans certains exemples d'applications présentés au chapitre IV, environ 80 pour cent des programmes réalisés sont consacrés à des calculs et/ou du contrôle algorithmique. Une application réalisant l'insertion de composants électroniques non standard en utilisant un capteur de vision tridimensionnelle a nécessité la mise en oeuvre de calculs géométriques importants. Une autre application -- le réglage de l'électronique d'un terminal de visualisation alphanumérique en utilisant un système de vision noir-et-blanc -- a conduit à la réalisation d'une procédure de diagnostic écrite en LM pour déterminer les actions de réglage (par exemple la rotation d'un potentiomètre) en fonction des défauts des "mires" (images de test).

La syntaxe du support algorithmique est très important, car elle définit le cadre dans lequel seront insérées les autres instructions du langage. Elle a une influence directe sur la lisibilité des programmes et la facilité de les modifier, donc sur l'acceptation du langage par les usagers potentiels.

b) Structures et instructions du langage LM

Le support algorithmique de LM s'inspire directement des langages PASCAL et ALGOL. Il inclut:

- la possibilité de déclarer et d'utiliser des variables symboliques de différents types (booléen, entier, réel, ...) indicées ou non, au sein d'expressions parenthésées,

- des instructions de contrôle: SI ... ALORS ... SINON, TANTQUE ..., POUR ..., CAS ...,

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

- la possibilité de définir des procédures et des fonctions, et de les appeler avec deux modes de passage de paramètres (adresse et valeur).

Il n'incorpore cependant pas la notion de structure que l'on trouve en PASCAL. De plus, par souci de portabilité, LM n'inclut que des fonctions simples de manipulation de fichiers (LIRE, ECRIRE, REWIND), et il ne comporte aucune fonction d'appel du système d'exploitation. Toutefois, de tels appels, ainsi que des programmes écrits dans d'autres langages que LM peuvent être ajoutés à l'interpréteur et appelés depuis des programmes en LM comme des procédures afin d'étendre ces facilités.

Exemple:

Nous illustrons le support algorithmique de LM sur un exemple de palétisation développé par la société SCEMI. La tâche consiste à saisir chaque rotor contenu dans une palette 1 (cf. Figure I.1), le déposer sur un posage intermédiaire, lui ajouter un roulement approvisionné par une ligne d'amenée, et ranger l'ensemble rotor-roulement dans une palette. Les deux palettes sont identiques. Le rôle du posage intermédiaire est de permettre un positionnement précis (par gravité) des robots avant l'insertion des roulements, la précision dans la palette 1 étant insuffisante. La Figure I.2 visualise les relations entre le posage, le rotor, et le roulement.

Le programme mis en oeuvre fait appel à deux procédures PRENDRE et DEPOSER. La procédure PRENDRE demande trois paramètres: la position de prise d'un objet, la distance d'approche de cette prise suivant la verticale, et la distance de dégagement de cette prise suivant la verticale. De façon semblable, la procédure DEPOSER demande la position de dépose, ainsi que les distances d'approche et de dégagement de cette position suivant la verticale. Il faut également noter qu'en LM, une position est représentée par un repère cartésien (cf. paragraphe 4.b).

SUPPORT ALGORITHMIQUE

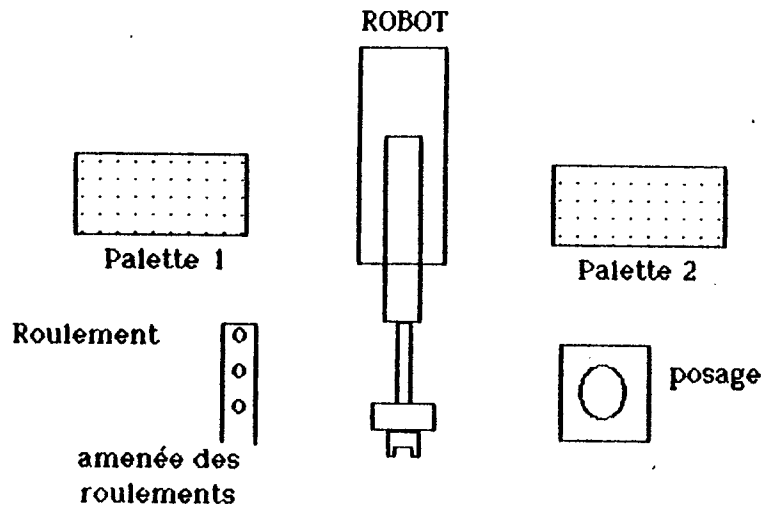


Figure 1.1 : Vue de dessus du poste de montage des roulements

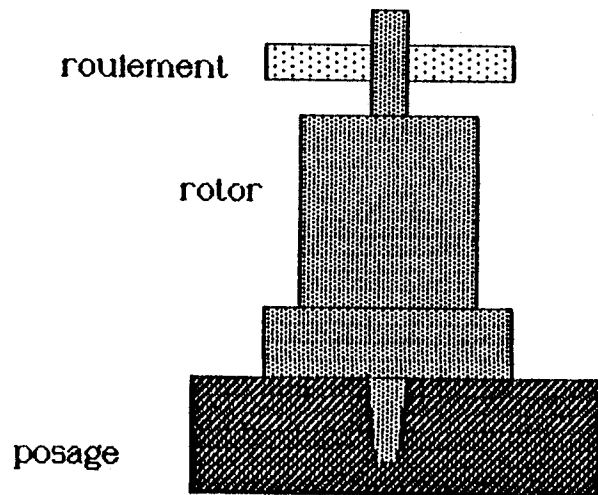


figure 1.2:

Le rotor et son roulement sur le posage intermediaire.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

Le programme principal est alors le suivant:

```
PROGRAMME DEMO;
CO programme de la manipulation présentée sur le stand
  de la société SCEMI à l'occasion de la biennale
  de la Machine-Outil Paris 1982;
REPERE PALETTE_1,PALETTE_2,PRISE,POS,POSAGE;
REPERE POS_INITIALE;
REPERE POSAGE1,ROULEMENT,POSEROULE;
ENTIER LGN,COL;
REEL INCR_LGN,INCR_COL;
FICHER ENTREE PALETT;
CO fichier dans lequel on a mémorisé les points
  en phase d'apprentissage avec un palonnier;
EXT PROCEDURE PRENDRE( REPERE; REEL; REEL );
EXT PROCEDURE DEPOSE( REPERE; REEL; REEL );
DEBUT
POS_INITIALE:=ROBOT; COL:=0; INCR_LGN:=70.; INCR_COL:=70.;
FIXER VITESSE ROBOT A 0.7;
LIRE PALETTE_1,PALETTE_2,POSAGE,POSAGE1,POSEROULE
  ,ROULEMENT DANS PALETT;
CO boucle de dépalétisation ;
TANTQUE COL< 5 FAIRE
  LGN:=0;
  TANTQUE LGN< 4 FAIRE
    CO calcul nouvelle position;
    PRISE:=PALETTE_1*TRANSLAT(VX,REE(LGN)*INCR_LGN)
      *TRANSLAT(-VY,REE(COL)*INCR_COL);
    PRENDRE(PRISE,50.,90.);
    DEPOSE(POSAGE,50.,50.);
    PRENDRE(ROULEMENT,10.,10.);
    DEPOSE(POSEROULE,15.0,0.0);
    PRENDRE(POSAGE1,0.,40.);
    CO calcul nouvelle position de dépose;
    POS:=PALETTE_2*TRANSLAT(-VX,REE(LGN)*INCR_LGN)
      *TRANSLAT(VY,REE(COL)*INCR_COL);
    DEPOSE(POS,90.,50.);
    LGN:=LGN+1;
  FINFAIRE;
  COL:=COL+1;
FINFAIRE;
CO retour en position initiale;
DEPLACER ROBOT A POS_INITIALE;
ECRIRE "CYCLE TERMINE";
FIN;
```

SUPPORT ALGORITHMIQUE

Remarque:

Certaines variables d'un programme en LM permettent d'exprimer des grandeurs dans les unités de son choix, indépendamment des unités utilisées par les interfaces de LM avec les dispositifs matériels (actionneurs et capteurs). Il existe des constantes prédéfinies telles que M (mètre), N (newton), etc Ainsi:

$DIST = 0.1 * M$

donne à DIST la valeur 100 (car l'unité de longueur de LM est le millimètre).

De même:

$F = 2.5 * N$

donne à F la valeur 2.5 (car l'unité de force de LM est le newton), l'intérêt étant de rendre le programme plus lisible.

c) Discussion

Le langage AL a été le premier langage de programmation de robots à proposer un support algorithmique évolué. Depuis, la nécessité d'un tel support est largement reconnue, et la plupart des langages récents, tels que AML, VAL-II et RAIL, offrent des facilités comparables à celles de LM. Il faut signaler que VAL-II est le successeur de VAL, qui lui n'avait qu'un support algorithmique réduit.

Le choix d'une syntaxe de type ALGOL/PASCAL pour LM vise à favoriser la construction de programmes structurés, supposés plus faciles à écrire et à comprendre. Des choix analogues ont été faits dans d'autres langages tels que AL ou AML, avec cependant des variantes (par exemple AML s'inspire assez largement de LISP). Il existe également des choix radicalement différents. Par exemple, une autre approche, illustrée par les langages MCL et ROBEX, considère que la programmation des robots sera effectuée (du moins partiellement) par des personnes ayant déjà une expérience des machines-outils à commande numérique; ainsi les langages MCL et ROBEX ont une structure syntaxique comparable à celle du langage APT, avec des instructions de contrôle limitées. Encore, une autre approche est celle du langage LPR qui est basé sur la notion de graphe d'états inspirée du GRAFCET (langage d'automates programmables) [Blanchard]. Ces approches ont toutes leurs justifications. Pour notre part, nous pensons que l'évolution rapide de la Robotique conduit à un type de machine trop informatisée pour que l'on puisse

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

à terme se contenter de constructions syntaxiques simplifiées.

Le contenu du support algorithmique de LM est semblable à celui des langages de commande de robots récents. On pourrait certes juger qu'il manque de structures de données sophistiquées (listes, structures, ...) mais il est cohérent avec les principes 3 et 4 énoncés au paragraphe précédent, en particulier la possibilité de manipuler en ligne des structures (bien entendu, la situation aurait été différente si nous avions choisi de faire de LM, non pas seulement un langage d'exécution, mais aussi un langage d'implantation). Il faut cependant noter que les langages qui résultent de l'extension de langages existants par des bibliothèques de programmes (par exemple RCCL) sont susceptibles, par construction, d'hériter de supports algorithmiques plus riches que celui de LM.

4. MODELISATION DE L'UNIVERS

a) Objectif de la fonction

Une partie des instructions d'un programme de robot a pour effet de modifier l'état de l'univers (1) en cours. Certains objets changent de position; des relations entre objet peuvent être également transformées: par exemple, la saisie d'une pièce (2) crée une liaison rigide entre la pièce et la pince du robot concerné.

Il est important qu'un robot mémorise où il a déposé une pièce s'il a besoin de la réutiliser ultérieurement (c'est même impératif si aucun capteur ne peut lui redonner cette information). Il peut également être utile de mémoriser une information fournie par un capteur, si cette information doit être réutilisée plus tard, et qu'elle n'est pas

(1) Par "univers", nous entendons le (ou les) robots et son (leur) environnement.

(2) Le mot "objet" désigne indifféremment un composant de robot ou un composant de l'environnement. Le mot "pièce" est réservé aux objets manipulés par les robots.

MODELISATION DE L UNIVERS

susceptible d'avoir été modifiée; on évite ainsi des appels redondants, et parfois coûteux en temps, aux fonctions de perception. La mémorisation d'informations sur l'univers est aussi essentielle lorsque plusieurs instructions du programme s'exécutent en parallèle de façon asynchrone.

L'objectif de la modélisation de l'univers est donc de maintenir à jour une base de données qui représente, à chaque instant de l'exécution d'un programme, l'état courant de l'univers. Cette base sert d'interface entre différentes instructions du programme: certaines y prennent de l'information, d'autres en apportent, etc ...

Cet objectif peut même aller au delà. L'interpréteur du langage de programmation peut utiliser la structure du modèle de l'univers pour effectuer des mises à jour automatiques lors de l'exécution d'instructions (effets de bord); par exemple, le déplacement d'un objet provoque le déplacement de tous les objets qui lui sont physiquement attachés. Les mises à jour automatiques effectuées par l'interpréteur peuvent considérablement simplifier la programmation d'une tâche, et réduire les risques d'erreur.

b) Structures et instructions du langage LM

La modélisation de l'univers est réalisée en LM principalement à l'aide de repères cartésiens et de liaisons entre ces repères. Bien entendu toute variable symbolique, quel que soit son type, peut également servir à modéliser un aspect de l'univers (par exemple, un réel peut mémoriser la distance entre deux objets, ou le poids d'une pièce).

Chaque repère d'un tel modèle représente un objet ou une partie d'objet. Plusieurs repères peuvent être associés à un même objet pour représenter différents aspects de cet objet: par exemple, un repère sera utile pour spécifier la position de l'objet sur une table de travail, un second repère définira une position de prise de cet objet (position à laquelle on doit amener une pince pour saisir l'objet), et un troisième repère déterminera la position d'un trou dans l'objet.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

Une liaison entre deux repères signifie que les deux repères sont solidaires. Ce sera par exemple les trois repères associés ci-dessus au même objet. Ce sera aussi le cas pour les deux repères associés, l'un à une pièce, l'autre à une pince, lorsque la pince tient la pièce.

La Figure 1.3 illustre la modélisation de l'univers de LM en combinant les deux exemples précédents. Comme nous allons le préciser, un tel état de l'univers peut être décrit par les instructions suivantes:

```
1. PINCE_1:= ROBOT(i)*TRANSLAT(VZ,-50.);
2. LIER PINCE_1 A ROBOT(i);
   ....
3. CUBE:= STATION*TRANSLAT(VX,-100.)*TRANSLAT(VY,-200.);
4. LIER TROU-CUBE A CUBE PAR
   TRANSLAT(VX,50.)*TRANSLAT(VZ,20.)
   *ROT(VX,PI/2)*ROT(VZ,PI/2.);
5. PRISE-CUBE:= CUBE * TRANSLAT(VZ,100.) * ROT(VX,-PI/2);
   ...
6. LIER PRISE-CUBE A CUBE;
   ...
7. LIER CYLINDRE A PINCE_1 PAR TRANSLAT(VY,30.);
```

Un repère est créé en LM par la déclaration d'une variable symbolique de type "repère". Cette variable est une structure qui regroupe deux sortes d'informations: la position du repère relativement à un repère prédéfini, appelé STATION, et les liaisons du repère avec d'autres repères. Le repère STATION est le repère de référence dont la position est connue a priori du programmeur et de l'interpréteur du langage (en fait, le programmeur peut souvent travailler "en relatif" et ignorer la position de STATION dans l'espace).

La position d'un repère est définie en appliquant une transformation (combinaison de rotations et de translations) à un repère de position connue. Ainsi le programmeur peut définir ses propres repères de travail sans avoir à toujours se ramener au repère STATION (cf. ligne 5 de l'exemple ci-dessus).

MODELISATION DE L UNIVERS

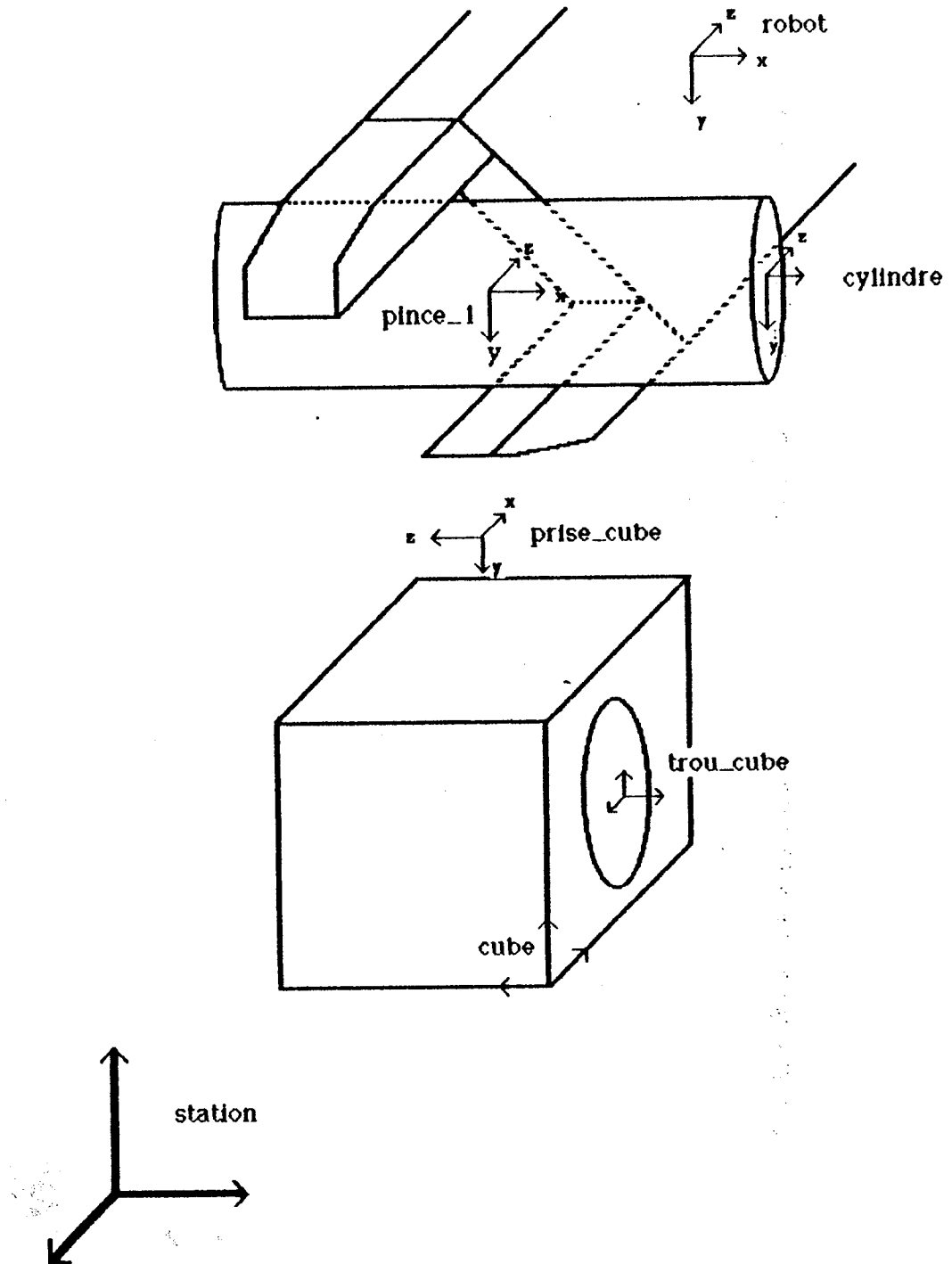


figure 1.3: Exemple de modélisation de l'univers en LM.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

Chaque rotation et translation composant une transformation est décrite à l'aide d'un vecteur (cf. lignes 3, 4, 5 et 7 de l'exemple). Pour faciliter l'écriture des programmes, LM permet de déclarer et de manipuler des variables symboliques de types "transformation" et "vecteur". Plusieurs opérations sur ces variables (composition de transformations, génération de transformation, addition de vecteurs, ...) sont définies dans le langage (cf. Manuel de Référence).

Plusieurs repères sont connus a priori de l'interpréteur du langage LM. Outre STATION c'est aussi le cas des repères ROBOT(i), i=1,2,... Chaque ROBOT(i) est un repère attaché à l'extrémité utile du robot numéro i (les robots sous le contrôle du même interpréteur sont numérotés 1,2,...). La position de ROBOT(i) n'est pas sous le contrôle direct du programmeur; elle est calculée par l'interpréteur chaque fois que c'est nécessaire, en fonction des valeurs transmises par les capteurs de position internes du robot. Comme nous le verrons au paragraphe 5, le tableau ROBOT(i) appartient à une classe de variables, appelées variables d'état.

Une liaison entre deux repères est créée (resp. détruite) par l'instruction LIER (resp. DELIER). L'instruction LIER permet également de spécifier la position du premier repère relativement au second (cf. lignes 4 et 7 de l'exemple ci-dessus). A chaque instant, l'ensemble des repères et de leurs liaisons définit un graphe dont chaque composante connexe est appelée "solide". Toute modification de la position d'un repère par une instruction d'affectation ou par une instruction de déplacement entraîne automatiquement la modification correspondante (en mémoire) des positions des repères appartenant au même solide. Ainsi, dans l'exemple précédent, toute modification de la position du repère ROBOT(i) entraîne la mise à jour correspondante des positions des repères PINCE_1 et CYLINDRE.

Dans un même solide, il ne peut figurer au plus qu'un seul repère d'état. Tous les repères d'un solide contenant un repère ROBOT(i) sont dits déplaçables; les instructions de mouvement (cf. paragraphe 6) permettent de décrire le déplacement de n'importe lequel de ces repères. Par exemple, compte tenu des liaisons créées dans l'exemple ci-dessus, l'instruction:

```
DEPLACER CYLINDRE A TROU-CUBE;
```

provoque un mouvement du robot numéro 2 conduisant à superposer le repère CYLINDRE sur TROU-CUBE.

MODELISATION DE L UNIVERS

L'interpréteur ne vérifie pas (il ne possède d'ailleurs pas les informations qui lui permettraient de le faire) que deux repères liés entre eux correspondent à deux objets physiquement solidaires.

Remarque:

La notion de liaison pose un problème de définition sémantique en relation avec les procédures (et fonctions). En effet, considérons trois repères R1, R2 et R3 et une procédure P. Les repères R1 et R2 sont déclarés dans la même unité de programme que la procédure P (variables globales); le repère R3 est déclaré localement dans P (variable locale). Supposons de plus que P contienne les deux instructions LIER R1 A R3 et LIER R2 A R3. Au retour de P, R3 "n'existe plus"; mais on pourrait faire en sorte que la liaison indirecte entre R1 et R2 subsiste. Le choix que nous avons effectué dans LM est plus net: toutes les liaisons de R3 sont effacées, si bien que R1 et R2 se retrouvent déliés au retour de P.

c) Discussion

Les concepts de modélisation de l'univers de LM sont directement inspirés du langage AL. L'intérêt majeur de la modélisation par des repères cartésiens est de combiner concision de la représentation et simplicité des algorithmes [Paul 81]. De plus elle fournit une qualité de description qui est supérieure aux représentations utilisées par de nombreux langages de programmation de robots. En effet, dans ces langages, on représente souvent la position et l'orientation d'un objet par 6 paramètres x,y,z,phi,thêta,et psi, (trois coordonnées cartésiennes pour la position, et trois angles d'Euler pour l'orientation). Cette représentation est plus contraignante pour l'utilisateur.

La représentation de l'univers par des repères reste cependant assez rudimentaire. En effet, il n'est pas toujours facile de raisonner sur les repères, et on souhaiterait disposer de modèles géométriques explicitant les formes des objets. De tels modèles sont toutefois actuellement incompatibles avec les contraintes de la commande des robots en temps réel. On notera cependant qu'une facilité de ce type est fournie au sein du système LM par le langage LM-GEO (cf. chapitre V paragraphe 5); mais l'interpréteur de LM-GEO opère hors-ligne, et engendre du

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

code LM.

Les opérations sur les repères définies dans LM imposent que la position d'un repère soit complètement définie. Cela peut conduire le programmeur à être obligé de sur-contraindre artificiellement la position d'un objet, notamment lorsque celui-ci présente des symétries (par exemple, l'orientation d'un objet cylindrique autour de son axe est sans intérêt). La mise en oeuvre d'opérations sur des repères à positions incomplètement spécifiées nécessiterait de disposer de facilités de calcul symbolique au sein de l'interpréteur du langage. Aucun langage de robotique ne permet aujourd'hui de telles facilités en raison de leur coût algorithmique.

LM est le seul langage, avec AL, à disposer des instructions LIER et DELIER. On notera que, dans LM, elles manipulent les seules structures de données du langage ayant un caractère dynamique. Cette exception a été motivée d'une part par l'intérêt majeur des liaisons pour simplifier la programmation (tous les utilisateurs un peu expérimentés de LM en font un usage intensif), et d'autre part par la mise au point d'algorithmes performants pour traiter les structures correspondantes (cf. [Mazer 81]).

5. COMMUNICATION AVEC L'ENVIRONNEMENT

a) Objectif de la fonction

L'expression "communication avec l'environnement" recouvre deux aspects distincts:

- D'une part, la perception au sens traditionnel du terme. Il s'agit d'obtenir des informations sur les objets de l'environnement des robots par le biais de capteurs. Ces capteurs sont typiquement des systèmes de vision qui identifient et localisent des objets, ou des capteurs de force qui mesurent les efforts exercés par des robots.

- D'autre part, la lecture d'informations provenant de dispositifs périphériques qui ne sont pas sous le contrôle de l'interpréteur du langage LM (dispositifs d'amenée de pièces, machines spéciales, ...). Une telle information peut par exemple caractériser l'état opératoire d'un dispositif.

COMMUNICATION AVEC L ENVIRONNEMENT

Le but de la communication avec l'environnement est de permettre au programme de commande de robots d'adapter les traitements et les actions des robots à l'environnement. Par exemple, une donnée d'un système de vision permettra de calculer la destination d'une trajectoire, ou de lancer un sous-programme de mise au rebut d'une pièce si l'inspection visuelle a montré qu'elle était défectueuse. Un capteur de force permet d'identifier les contacts entre pièces lors d'une opération de montage, pour décider de mouvements de correction afin d'éliminer les erreurs de positionnement. Les informations sur l'état opératoire d'une machine périphérique peuvent servir à coordonner des processus parallèles, etc ...

La prise en compte de l'environnement dans un programme de commande de robots a une importance croissante car il est reconnu qu'elle permet de réduire considérablement les coûts de mise en oeuvre des robots, notamment en ce qui concerne les outillages spécialisés. Dans certains cas elle est même indispensable à l'automatisation d'une opération.

Sans justifier totalement à elle seule la programmation textuelle des robots, la fonction de communication avec l'environnement la motive fortement. En effet, les autres modes de programmation de niveau manipulation, la programmation par l'exemple et la programmation graphique, ne l'incluent au mieux que de façon très limitée.

La principale difficulté de l'insertion de la fonction de communication avec l'environnement au sein d'un langage textuel résulte de la variété potentielle des capteurs et des dispositifs périphériques. Il est essentiel que le système présente de ce point de vue une certaine adaptabilité pour qu'un programmeur puisse faire usage d'un nouveau capteur sans qu'il soit nécessaire de modifier l'interpréteur du langage.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

b) Structures et instructions du langage LM

Le langage LM offre deux constructions principales pour la communication avec l'environnement: les variables d'état et les fonctions capteurs. En première approximation, on peut voir les variables d'état comme des fonctions capteurs sans paramètre.

Une fonction capteur (ou une variable d'état) s'évalue à une valeur de type entier, booléen, réel, vecteur, ou transformation. Cette valeur n'est pas sous le contrôle direct du programmeur; c'est par exemple un réel mesurant la force exercée par un robot le long d'une direction donnée, une transformation définissant la position d'un objet déterminée par un système de vision relativement à un repère connu, un booléen donnant l'état opératoire d'un dispositif périphérique, un vecteur définissant l'orientation d'une force, etc ...

Exemples de fonctions capteurs:

- VISION(NOBJ,K): fonction capteur s'évaluant à la transformation qui définit la position d'un repère attaché à l'objet désigné par l'entier NOBJ dans le repère STATION; K est un entier qui désigne le système de vision qui effectuera la reconnaissance de l'objet et sa localisation.

- FORCE(i,V,REP): fonction capteur s'évaluant à la composante de force exercée par le robot numéro i le long du vecteur V dans le torseur du repère REP.

Une variable d'état peut aussi désigner un repère, dit alors repère d'état. La position de ce repère n'est pas sous le contrôle direct du programmeur, mais les liaisons éventuelles de ce repère avec d'autres repères du programme le sont.

Exemples de variables d'états:

- ROBOT(i): variable d'état désignant le repère attaché à l'extrémité du robot numéro i.

- TEMPS-DEPL(i): variable d'état indicée par le numéro d'un robot mesurant le temps écoulé depuis le début du dernier déplacement du robot numéro i.

COMMUNICATION AVEC L ENVIRONNEMENT

L'indice *i* d'une variable d'état peut être omis s'il est égal à 1. Ainsi ROBOT (1) et ROBOT sont équivalents (cette remarque est destinée à faciliter la lecture d'exemples de programmes donnés plus loin).

Une fonction capteur, ou une variable d'état, peut être utilisée dans toute expression. Par exemple:

```
DEPLACER PINCE_1 A STATION*VISION(3,1)*TRANSLAT(VZ,50);
```

peut commander un déplacement du repère PINCE_1 d'un robot à la position de prise d'un objet localisé à l'aide du système de vision 1. VISION(3,1) définit la position de l'objet relativement à STATION, la translation suivant VZ définit la position de prise relativement à cette position.

Les variables d'état peuvent être passées comme paramètres d'une procédure (ou d'une fonction) en modes "valeur" et "état". En mode "état", le paramètre formel dans la procédure est traité comme s'il était la variable d'état elle-même. Il est ainsi possible d'écrire une procédure utilisant des variables d'état inconnues lors de l'écriture de la procédure.

Deux instructions:

```
ECRIRE v SUR VOIE x1,x2,...xn;
```

```
LIRE w SUR VOIE y1,y2,...,yp;
```

donnent accès à une interface normalisée "ESVOIE" de l'interpréteur (cf. chapitre II, paragraphe 4) qui permet d'intégrer au système de nouveaux capteurs et de nouveaux dispositifs périphériques.

Exemple:

Nous illustrons ici l'utilisation de données transmises par des capteurs dans un programme en LM. Pour cela nous reprenons l'exemple du paragraphe 3 (palétisation).

L'utilisation d'un posage intermédiaire pour monter les roulements résultait de la trop grande imprécision de positionnement des rotors sur la palette 1. Le rôle du posage était d'assurer un recentrage automatique des rotors lors du lâcher. Si l'on munit le robot d'un capteur de force, il est possible de supprimer le posage intermédiaire, et même le transfert d'une palette à une autre.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

La boucle principale devient:

```
TANTQUE COL< 5 FAIRE
  LGN:=0;
  TANTQUE LGN< 4 FAIRE
    PRENDRE(ROULEMENT,10.,10.);
    CO calcul nouvelle position de dépose;
    POS:=PALETTE_1*TRANSLAT(-VX,REE(LGN)*INCR_LGN)
      *TRANSLAT(VY,REE(COL)*INCR_COL);
    INSERER(POS,90.,50.,20,ECHEC);
    SI ECHEC ALORS ALLERA ETIQ_ECHEC; FINSI;
    LGN:=LGN+1;
  FINFAIRE;
  COL:=COL+1;
FINFAIRE;
```

La procédure INSERER utilise des données du capteur de force pour réduire l'imprécision sur les positions. Cette procédure s'écrit:

```
PROCEDURE INSERER (REPERE POS_FIN, REEL APPROCHE,
  REEL DEGAGE, REEL DIAMETRE, BOOLEEN ECHEC);
ENTIER N; VECTEUR V; REPERE TROU_1;
REEL DIAMETRE, EPS;
CO FX< FY< FZ sont trois variables d'état mesurant
les forces exercées par le robot suivant
les axes du repere ROBOT;
DEBUT
  N := 0; TROU_1 := POS_FIN * TRANSLAT(-VZ, 10);
  EPS := DIAMETRE/2;
  TANTQUE N< 5 FAIRE
    DEPLACER ROBOT A TROU_1 JUSQUA FZ>15;
    SI DISTANCE (ROBOT, POSFIN)>2.
      ALORS RETOUR VRAI;
    SINON
      DEPLACER ROBOT DE TRANSLAT(VECT(FX,FY,0),EPS);
      EPS := EPS/2; N := N + 1;
    FINSI;
  FINFAIRE;
  RETOUR FAUX;
FIN;
```


c) Discussion

L'évaluation d'une fonction capteur ou d'une variable d'état peut nécessiter des calculs plus ou moins importants. A un extrême, il peut s'agir d'une simple opération de lecture sur un port d'entrée-sortie. A l'autre extrême (analyse d'image par exemple), les calculs peuvent être particulièrement volumineux. En général, ils ne sont pas alors effectués par l'interpréteur LM, mais par le capteur lui-même qui possède son propre processeur (par capteur nous entendons donc non seulement le récepteur physique mais aussi un système de traitement). Il n'en reste pas moins que le temps nécessaire à l'évaluation d'une fonction capteur ou d'une variable d'état dépend fortement de la fonction (resp. de la variable). Il faut en tenir compte si ces évaluations se font en parallèle avec l'exécution d'un mouvement de robot ou d'une opération d'outil, surtout si elles peuvent interagir avec cette exécution (cf. par exemple les clauses JUSQUA des paragraphes 6 et 7).

De façon générale, la communication avec les capteurs au sein des langages les plus récents est comparable à celle mise en oeuvre dans LM. Toutefois, nous pensons que l'accès que donne LM à l'interface ESVOIE est une facilité originale.

Certains langages incluent des extensions destinées à la programmation d'un système de vision; il est alors possible de spécifier dans un même programme les opérations d'un robot et les traitements (prise d'image, binarisation, analyse de convexité, ...) sur des images. Les langages qui offrent de telles facilités sont AML/V, une extension du langage AML, et le langage RAIL. Un langage ayant une syntaxe de même type que LM, le langage LV (langage de Vision), a été défini par les sociétés ITMI et MATRA-ROBOTRONICS. Le langage LMV, union de LM et LV, est en cours de réalisation. Un exemple de programme LV est donné en annexe 1.

6. SPECIFICATION DES MOUVEMENTS

a) Objectif de la fonction

L'objectif est de donner au programmeur les constructions symboliques pour décrire les mouvements que doivent réaliser les robots pendant l'exécution d'une tâche.

La notion de mouvement recouvre plusieurs concepts qu'il convient de distinguer au niveau du langage de programmation. Il s'agit de:

- La géométrie du mouvement (c'est-à-dire la trajectoire). La façon la plus naturelle de décrire une trajectoire est en général de l'exprimer directement dans l'espace cartésien indépendamment de la structure mécanique du robot qui l'exécute. Le programmeur doit pouvoir choisir entre diverses géométries afin de minimiser la longueur du chemin à parcourir tout en évitant les collisions.

- La cinématique du mouvement. Elle concerne l'évolution du robot dans le temps le long de la trajectoire. La vitesse d'exécution peut avoir un effet sur la précision et être choisie en conséquence, notamment en fonction du degré d'encombrement de l'espace. Elle peut avoir été imposée par l'opération effectuée par l'outil terminal (par exemple celui-ci peut déposer un cordon de colle le long d'un joint).

- L'interaction du mouvement avec les capteurs. De nombreux mouvements sont contrôlés uniquement en position. Cela suppose toutefois d'avoir une connaissance de l'environnement qu'il n'est pas toujours possible d'avoir, y compris au moment où le mouvement commence à s'exécuter. Il est donc parfois utile d'adapter les réactions d'un robot en mouvement à des données sensorielles (typiquement force, vision, toucher). Une réaction simple consiste à stopper le mouvement lorsqu'une condition devient vraie. Une réaction plus sophistiquée consiste à adapter la géométrie du mouvement à des données transmises par des capteurs (mouvements dits "à compliance").

La coordination et la synchronisation de plusieurs mouvements et opérations de robots est un autre aspect à prendre en considération. Nous le traiterons au paragraphe 8.

SPECIFICATION DES MOUVEMENTS

b) Structures et instructions du langage LM

LM possède une seule instruction dont l'exécution provoque un mouvement de robot. Il s'agit de l'instruction DEPLACER. Sa sémantique fait intervenir les trois concepts introduits au paragraphe précédent. Nous présentons les constructions correspondant à chacun d'eux, puis nous donnons des exemples complets pour illustrer comment ces constructions peuvent être combinées.

- Trajectoire:

La trajectoire d'un robot peut être décrite soit dans l'espace cartésien, soit dans l'espace des coordonnées propres du robot.

* Dans l'espace cartésien:

La trajectoire est définie par le déplacement d'un repère déplaçable (cf. paragraphe 4.b). Elle comporte un ou plusieurs "segments". Chaque segment est soit un "segment libre" (l'option par défaut), soit un "segment cartésien", soit un "segment circulaire", soit un "segment planifié".

Dans le cas d'un segment libre, seule la position d'arrivée du repère à déplacer est spécifiée. Les degrés de liberté seront coordonnés par l'interpréteur du langage de telle sorte qu'ils démarrent et s'arrêtent en même temps. La trajectoire suivie entre la position de départ et la position d'arrivée dépend donc de la géométrie du robot. Elle convient en général aux mouvements rapides ne présentant pas de risque de collision.

Dans le cas d'un segment cartésien (resp. circulaire), la position d'arrivée du repère à déplacer est également spécifiée; de plus, l'origine du repère se déplacera le long d'une droite (resp. d'un arc de cercle), le repère subissant une rotation autour d'un axe de direction constante.

Dans le cas d'un segment planifié, le repère est déplacé suivant une séquence de positions enregistrées dans une table. Cette table peut par exemple avoir été construite par le module de planification de mouvements présenté au paragraphe 6 du chapitre V.

Le programmeur peut enchaîner plusieurs segments de même type ou de types différents. L'interpréteur calcule des transitions entre segments pour assurer une continuité de la vitesse.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

* Dans l'espace des coordonnées propres du robot:

On spécifie les degrés de liberté du robot qui seront déplacés et la position d'arrivée de chacun d'eux (la trajectoire ne comporte alors qu'un seul segment). L'interpréteur coordonne les degrés de liberté en mouvement.

- Cinématique:

Le programmeur peut contrôler la vitesse d'un déplacement en fixant la valeur d'un coefficient de vitesse à un nombre entre 0 et 1. Soit T_1 la durée minimale pour exécuter un mouvement suivant une trajectoire donnée aux vitesses et accélérations maximales (des degrés de liberté) connues de l'interpréteur. Soit T_C la durée pour exécuter un mouvement suivant la même trajectoire avec un coefficient de vitesse égal à C . L'interpréteur de LM calcule la vitesse de chaque degré de liberté de telle sorte que $T_C = T_1/C$.

Le programmeur peut fixer le coefficient de vitesse par des instructions séparées des instructions DEPLACER; il peut aussi la fixer segment par segment dans chaque instruction DEPLACER.

Il est également possible de contrôler indirectement la vitesse d'un déplacement en fixant la durée d'exécution de chaque segment.

- Interaction avec les capteurs:

La seule construction disponible est une clause d'arrêt qui, dans une instruction DEPLACER, spécifie une condition. Cette condition est évaluée périodiquement pendant le mouvement; lorsqu'elle devient vraie, le mouvement est stoppé. Le programmeur a la possibilité de fixer la fréquence d'évaluation de la condition. Si l'interpréteur n'arrive pas à suivre cette fréquence à l'exécution, il envoie un message le signalant.

SPECIFICATION DES MOUVEMENTS

Exemples:

1- FIXER VITESSE A 0.2;

DEPLACER ROBOT A POSITION;

Le repère ROBOT est déplacé à la position du repère POSITION suivant un segment libre, avec un coefficient de vitesse égal à 0.2.

2- FIXER VITESSE A 0.2;...

DEPLACER ROBOT A POSITION AVEC VITESSE =0.8;

Le repère ROBOT subit le même déplacement que dans l'exemple 1, mais à une vitesse supérieure.

3- DEPLACER PINCE_1 A PRISE-CUBE EN CARTESIEN;

FERMER PINCE; LIER CUBE A PINCE_1;

DEPLACER CUBE VIA CUBE*TRANSLAT(VZ,50.) EN CARTESIEN,

DESTINATION*TRANSLAT(VZ,50.) EN LIBRE

A DESTINATION EN CARTESIEN AVEC VITESSE=0.1;

La première instruction DEPLACER conduit à un déplacement de PINCE_1 à PRISE-CUBE suivant un segment cartésien. La seconde instruction DEPLACER spécifie trois segments de différents types. Le premier segment (cartésien) correspond typiquement au retrait d'un objet de la table de travail (d'où le mode cartésien qui permet d'éviter des collisions avec d'autres objets); le second segment (libre) correspond à un déplacement rapide à une altitude suffisante pour éviter toute collision; le troisième segment correspond à un dépôt de l'objet à la position DESTINATION (la vitesse lente permet d'éviter un heurt trop violent de l'objet contre la table).

4- DEPLACER CUBE VIA CUBE*TRANSLAT(VZ,50.) AVEC DUREE= 2*S

A DESTINATION EN CARTESIEN;

La vitesse du premier segment est fixée par la durée de son exécution, celle du second par le coefficient de vitesse courant. Le premier segment est un segment libre (option par défaut).

5- DEPLACER CUBE DE TRANSLAT (VZ,0.50*M) EN CARTESIEN

JUSQUA FZ(2)>5*N;

Le repère CUBE est déplacé en ligne droite le long de son axe VZ d'une distance au plus égale à 0.50 M. Le mouvement est arrêté si la force FZ mesurée sur le robot numéro 2 (sans doute ici celui qui tient le cube) dépasse 5 newtons.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

6- DEPLACER CUBE DE TRANSLAT (VZ,0.50*M) EN CARTESIEN
JUSQUA FZ(2)>5*N AVEC FREQUENCE =20;

Le mouvement est le même que dans l'exemple 5, mais on impose ici que la condition d'arrêt soit évaluée 20 fois par seconde.

7- DEPLACER CYLINDRE A TROU-CUBE EN CARTESIEN
JUSQUA FX>5. OU FY>5. OU FZ>5.;

8- DEPLACER OBJET A BUT
JUSQUA DISTANCE (OBJET,OBSTACLE)<DIST;

Ici le mouvement est effectué en mode libre, mais il y a un obstacle dont la position est déterminée par le repère OBSTACLE; la clause d'arrêt permet de stopper le mouvement si la trajectoire passe trop près de l'obstacle.

9- DEPLACER AXE 2 DE 25*DG;

L'axe 2 du manipulateur 1 va effectuer une rotation de 25 degrés (dans le sens trigonométrique) par rapport à sa position courante.

10- DEPLACER TORCHE SUIVANT CERCLE TANGENT(PI/2)
PAR [POS1,POS2];

Le repère est déplacé le long du cercle défini par sa position initiale et les deux repères POS1 et POS2. La longueur d'arc est de PI/2 (demi-cercle), et l'orientation de la torche gardera le même angle par rapport à la tangente au cercle durant tout le mouvement (option "TANGENT").

c) Discussion

L'instruction DEPLACER de LM illustre de façon typique le principe 2 énoncé au paragraphe 2 de ce chapitre. Nous avons essayé de définir des constructions syntaxiques riches pour combiner un nombre de concepts de base aussi petit que possible. LM se distingue ainsi de nombreux autres langages qui fournissent plusieurs instructions de mouvement correspondant à différents types de trajectoire. Par exemple RAIL comporte les instructions MOVE, APPROACH, DEPART, et ROTATE.

Les facilités offertes par LM restent aujourd'hui uniques. Ainsi aucun langage ne permet de combiner dans une même instruction plusieurs segments de trajectoire de différents types. Certains langages permettent d'enchaîner des segments équivalents à l'aide de plusieurs instructions de mouvement,

SPECIFICATION DES MOUVEMENTS

mais avec un temps d'arrêt entre deux segments. Plusieurs langages opérationnels ne disposent pas de possibilités générales de trajectoires rectilignes, par exemple HELP, RPL et AML (on notera toutefois que ce dernier a été conçu pour des robots dits cartésiens, ce qui permet d'engendrer des lignes droites par simple coordination des degrés de liberté; mais il ne doit pas y avoir alors de rotation du poignet). LM est également l'un des seuls langages avec VAL-II à permettre l'exécution de mouvements précalculés. On notera cependant qu'un langage tel que MCL offre directement des possibilités intéressantes, telles que la définition d'une trajectoire comme l'intersection de deux surfaces mathématiques, inexistantes en LM. D'autres langages ont des constructions spécifiques d'un robot privilégié (par exemple le langage VAL et le robot PUMA).

La description de trajectoires à l'aide de repères cartésiens est assez naturelle, car elle évite en principe d'avoir à raisonner sur la structure géométrique des robots. Toutefois elle présente quelques inconvénients:

- Elle ne permet pas de gérer explicitement et simplement des configurations d'un robot (un robot à 6 degrés de liberté peut avoir jusqu'à 8 configurations possibles lui permettant de réaliser une position et orientation de sa pince).

- La représentation interne des transformations qui définissent les positions des repères (par exemple: les coordonnées homogènes) ne spécifie pas complètement comment les rotations doivent être exécutées. Par exemple:
DEPLACER REP DE ROT(VZ, $3 \cdot \pi / 4$) EN CARTESIEN;
suggère de faire tourner le repère REP autour de son axe Z d'un angle égal à $3 \cdot \pi / 4$; les choix effectués dans l'interpréteur LM (minimisation de la longueur du déplacement) conduisent à une rotation autour de l'axe Z de $-\pi / 4$.

- Le fait que les positions des repères doivent être complètement spécifiées (cf. paragraphe 4.c) peut conduire à des trajectoires malheureuses (par exemple lors de la saisie d'un cylindre sur sa face cylindrique).

- Il n'est pas facile de juger de l'accessibilité d'une position cartésienne, en raison des butées mécaniques sur les degrés de liberté.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

L'expérience a montré que ces inconvénients peuvent être partiellement éliminés en fournissant au programmeur des facilités pour décrire les mouvements de degrés de liberté des robots. Ceci a conduit à rajouter à LM les trajectoires spécifiées dans l'espace des coordonnées propres d'un robot. Il s'est avéré que ces trajectoires sont également très utiles pour piloter des structures mécaniques plus simples à 2 ou 3 degrés de liberté, qui participent souvent à l'environnement de robots plus élaborés.

Malgré sa simplicité, le contrôle de la vitesse d'un robot à l'aide d'un coefficient s'est avérée suffisante dans presque tous les cas traités. Un meilleur contrôle est obtenu en faisant appel à des mouvements planifiés (cf. paragraphe 6 du chapitre V).

La seule interaction d'un mouvement avec les données transmises par les capteurs d'environnement est gérée par la clause JUSQUA, que l'on retrouve dans quelques autres langages tels que AL et AML. LM n'offre aucune construction pour exprimer directement la complaisance d'un mouvement à des données sensorielles d'environnement. Jusqu'à présent de tels mouvements nécessitaient la mise en oeuvre de matériels spécialisés permettant un échantillonnage suffisamment dense du mouvement, et seuls AL et PAL possèdent des instructions appropriées. En principe, il est possible d'engendrer des mouvements compliants en LM avec plusieurs instructions DEPLACER utilisant les clauses JUSQUA, SANS ATTENTE et IMMEDIAT (cf. paragraphe 8); en pratique cette possibilité conduit à des mouvements s'adaptant lentement à l'environnement. De nouvelles facilités sont en préparation (cf. paragraphe 9).

7. SPECIFICATION DES OPERATIONS D'OUTILS TERMINAUX

a) Objectif de la fonction

L'objectif de la spécification des opérations d'outils terminaux est conceptuellement voisin de celui de la spécification des mouvements de robots. Il s'agit de donner au programmeur les constructions symboliques qui lui permettent de décrire les actions que devront réaliser les outils terminaux au cours de l'exécution d'une tâche

SPECIFICATION DES OPERATIONS D OUTILS TERMINAUX

d'application. Un outil peut être une pince pneumatique à deux états, une pince asservie à deux mors parallèles, une torche à souder, etc ...

La définition d'un formalisme de programmation approprié à la commande d'outils terminaux se heurte à deux difficultés majeures:

- Il existe potentiellement une grande variété d'outils terminaux, les pinces de préhension étant certes les plus fréquentes. Ces outils peuvent être fondés sur des principes de fonctionnement différents.

- Il n'y a pas de limite à la complexité potentielle d'un outil. On peut très bien imaginer que dans un avenir raisonnablement proche il devienne classique de monter sur un robot une "main articulée" comportant plusieurs "doigts" indépendants, chacun constitué de plusieurs "phalanges" (cf. [Salisbury 82])

b) Structures et instructions du langage LM

Deux sortes d'instructions sont disponibles en LM pour décrire les actions de l'outil terminal d'un robot.

Une instruction générale est destinée à la commande d'un outil quelconque. Elle est de la forme:

ACTIONNER OUTIL j DU ROBOT AVEC [x1,x2,...,xn];
où x1,x2,...,xn sont des paramètres tels qu'une vitesse, une distance, un nombre de tours, ...; leur signification dépend de la nature de l'outil actionné.

Des instructions plus spécifiques, mais ayant l'avantage d'être plus lisibles, permettent de décrire les opérations de pinces à deux mors:

ECARTER PINCE DU ROBOT i A [ou DE] y;
OUVRIR PINCE DU ROBOT i;
FERMER PINCE DU ROBOT i;

L'instruction ECARTER permet de commander l'ouverture d'une pince à deux mors asservie en position, soit en absolu, soit en relatif. Les instructions OUVRIR et FERMER commandent l'ouverture complète ou la fermeture complète de pinces à deux mors. Ces pinces peuvent être asservies en position ou

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

à deux états (pincés pneumatiques par exemple).

La clause JUSQUA peut être utilisée dans chacune des instructions ci-dessus, par exemple:

```
ECARTER PINCE DU ROBOT 2 DE -4.0 JUSQUA PRESSION(2)>2.5;
```

c) Discussion

L'instruction ACTIONNER est destinée à permettre la commande d'une grande variété d'outils. Elle manque toutefois de lisibilité et serait sans doute insuffisante pour piloter un outil véritablement complexe tel qu'une "main articulée". L'hypothèse faite dans LM est qu'un tel outil sera muni de son propre processeur de commande, de la même façon qu'un capteur complexe; l'instruction ACTIONNER reste alors utilisable. Il est également possible de voir l'outil et son processeur comme un dispositif externe commandable par l'interface ESVOIE (cf. paragraphe 5).

L'ensemble des instructions spécifiques destinées à la commande de pincés de préhension à deux mors apparaît d'une certaine façon comme un sous-langage. On pourrait imaginer que d'autres sous-langages puissent être créés pour d'autres types d'outils. Ainsi RAIL inclut plusieurs instructions destinées à la commande d'un outil terminal de soudure. Nous verrons au chapitre II que la syntaxe de LM est assez facilement extensible. Il serait ainsi possible d'ajouter de nouvelles instructions dont l'interprétation ferait appel aux mêmes fonctions que l'interprétation de l'instruction ACTIONNER. On remarquera qu'il y a là une analogie avec la construction de LMV (cf. paragraphe 5).

Dans tous les cas, il y aurait dans le futur un avantage certain non seulement pour les concepteurs de systèmes de programmation de robots, mais aussi pour les utilisateurs, à envisager une certaine normalisation des outils terminaux.

Certains langages tels que AL incluent l'instruction CENTRER qui permet de saisir un objet de position partiellement inconnue sans modifier sa position, à l'aide d'une pince à deux mors munie de capteurs tactiles. En LM, cette opération nécessite l'emploi de plusieurs instructions ECARTER et DEPLACER.

LE PARALLELISME ET LA SYNCHRONISATION

8. LE PARALLELISME ET LA SYNCHRONISATION

a) Objectif de la fonction

De plus en plus un robot n'est plus une machine isolée. Il interagit avec d'autres dispositifs, qui peuvent être d'autres robots, sous le contrôle du même programme ou non; ces dispositifs peuvent aussi être des machines spécialisées, des robots mobiles d'alimentation de pièces, etc ...

Le parallélisme et la synchronisation ont pour objectif de gérer des dispositifs opérant simultanément, éventuellement de façon asynchrone. On voudra par exemple coordonner l'action d'un outil terminal et les mouvements du robot porteur de cet outil; deux robots pourront interagir pour réaliser ensemble une opération d'assemblage complexe mettant en jeu trois pièces distinctes; plusieurs robots pourront réaliser séparément et en parallèle des sous-assemblages au sein d'un même assemblage; un robot de manipulation devra réagir à un événement asynchrone, tel que par exemple la demande d'un robot mobile d'être déchargé de la pièce qu'il apporte. Comme on peut le voir, les exemples où parallélisme d'action et synchronisation sont nécessaires, sont nombreux en Robotique. La finesse avec laquelle il faut synchroniser des actions peut être différente et les outils à mettre en oeuvre sont de complexité variable.

b) Structures et instructions du langage LM

Les deux constructions principales offertes par LM pour permettre la gestion du parallélisme sont les clauses SANS ATTENTE et IMMEDIAT. Ces clauses peuvent apparaître dans les instructions DEPLACER et dans les instructions de commande d'outils.

La clause SANS ATTENTE permet de continuer l'exécution d'un programme pendant l'exécution d'un mouvement ou d'une action d'outil. Soit par exemple la séquence d'instructions LM suivante:

```
DEPLACER ROBOT A QUELQUE-PART SANS ATTENTE;  
ECARTER PINCE DU ROBOT 2 A 0.5;
```

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

La pince du robot 2 se fermera immédiatement après que le mouvement du robot 1 ait été lancé. L'exécution du programme sera alors suspendue jusqu'à ce que la pince ait atteint l'écartement de 0.5.

La clause IMMEDIAT rend possible la modification d'un mouvement (ou d'une action d'outil) en cours. Par exemple, considérons le morceau de programme suivant:

```
DEPLACER ROBOT DE TRANSLAT (V1,D1) SANS ATTENTE;  
...  
DEPLACER IMMEDIAT ROBOT DE TRANSLAT(V2,D2);
```

Si la seconde instruction DEPLACER est exécutée alors que le robot 1 est toujours en cours de mouvement le long du vecteur V1, alors la trajectoire est modifiée pour devenir une translation suivant le vecteur V2. L'interpréteur du langage engendre une transition entre les deux trajectoires destinée à limiter l'accélération.

La clause IMMEDIAT peut aussi apparaître dans une instruction fixant le coefficient de vitesse d'un robot pour modifier la vitesse de ce robot alors que celui-ci est en mouvement, par exemple:

```
DEPLACER ROBOT A DEST EN CARTESIEN AVEC VITESSE =0.9;  
...  
FIXER IMMEDIAT VITESSE DU ROBOT 1 A 0.4;
```

Diverses possibilités complémentaires liées au parallélisme et à la synchronisation sont offertes par le langage. Il s'agit notamment de l'instruction ARRETER qui permet de commander l'arrêt d'un mouvement ou d'une action d'outil en cours, de variables d'état telles que EN_ACT(i) et EN-DEPL(i) qui permettent de savoir si un robot ou son outil sont actifs ou non, et de l'instruction PAUSE-FINPAUSE.

Cette dernière, qui consiste en une liste de "paires" condition-action, est destinée à faciliter l'expression de la coordination entre plusieurs opérations parallèles. Les conditions sont évaluées jusqu'à ce que l'une d'elles devienne vraie; l'action correspondante est alors exécutée.

LE PARALLELISME ET LA SYNCHRONISATION

Exemple 1:

L'exemple ci-après montre comment la clause SANS ATTENTE et l'instruction PAUSE-FINPAUSE peuvent être utilisées pour construire un programme composé de sections de codes exécutées en parallèle, chaque section de code correspondant à un robot. Cet exemple a un caractère pédagogique visant à illustrer le fonctionnement de certaines constructions du langage. Nous verrons au paragraphe 9 qu'une nouvelle instruction CODEBUT-COFIN est en préparation pour faciliter la construction de programmes composés de sections parallèles.

```
PROGRAMME DEUX_ROBOTS;
ENTIER NB,PC1,PC2;
REPERE BUT;
DEBUT
NB:=1; PC1:=1;PC2:=1;
DEB:
PAUSE
NON EN_DEPL(1) ET NON EN_ACT(1)
-> DEBUT
  CAS
    PC1=1 -> SI DISTANCE(ROBOT(2),BUT)>10.
              ET NON EN_DEPL(2)
              ALORS DEPLACER ROBOT A BUT SANS ATTENTE;
              SINON PC1:=PC1-1;
              FINSI;
    PC1=2 -> OUVRIR PINCE SANS ATTENTE;
    PC1=3 -> DEPLACER ROBOT DE TRANSLAT(-VZ,50.)
              SANS ATTENTE;
    PC1=4 -> FERMER PINCE SANS ATTENTE;
    PC1=5 -> DEBUT
              DEPLACER ROBOT A REPOS_1 SANS ATTENTE;
              PC1:=0;
              FIN;
  FINCAS;
  PC1:=PC1+1;
  NB:=NB+1;
  FIN;
NON EN_DEPL(2) ET NON EN_ACT(2)
-> DEBUT
  CAS
    PC2=1 -> SI DISTANCE(ROBOT,BUT)>10.
              ET NON EN_DEPL(1)
              ALORS DEPLACER ROBOT A BUT SANS ATTENTE;
              SINON PC2:=PC2-1;
```

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

```
        FINSI;
PC2=2 -> OUVRIR PINCE DU ROBOT 2 SANS ATTENTE;
PC2=3 -> DEPLACER ROBOT(2) DE TRANSLAT(-VZ,50.)
        SANS ATTENTE;
PC2=4 -> FERMER PINCE DU ROBOT 2 SANS ATTENTE;
PC2=5 -> DEBUT
        DEPLACER ROBOT(2) A REPOS_2 SANS ATTENTE;
        PC2:=0;
        FIN;

FINCAS;
PC2:=PC2+1;
NB:=NB+1;
FIN;
FINPAUSE;
SI NB< 20 ALORS ALLERA DEB; FINSI;
FIN;
```

Exemple 2:

L'exemple ci-après montre l'intérêt de la clause IMMEDIAT pour adapter le mouvement d'un robot à des conditions d'environnement. L'application choisie est la soudure en continu. On suppose qu'un capteur (vision tridimensionnelle [Mezin 82] [Borianne 84], capteur tactile, ou autre) fournit des informations géométriques sur la forme du joint de soudure au fur et à mesure de la progression de la torche tenue par le robot.

```
LIER TORCHE A ROBOT PAR TRSF_TORCHE;
DEPLACER TORCHE A BUT CARTESIEN SANS ATTENTE;
ACTIONNER OUTIL SANS ATTENTE;
TANTQUE EN_DEPL(1) FAIRE
    V1:=VECT_TRSL(VISION(1));
    SI LONG(V1)>EPS1
    ALORS DEPLACER IMMEDIAT TORCHE A VISION(1)
        *TRANSLAT(VZ,500.) SANS ATTENTE;
        CO l'axe Z du repère déplacé
        donne la direction;
    SINON
        SI DISTANCE(ROBOT,BUT)< EPS2
        ALORS ARRETER OUTIL;
        ARRETER ROBOT;
    FINSI;
FINFAIRE;
```

LE PARALLELISME ET LA SYNCHRONISATION

c) Discussion

Clairement LM n'offre pas de facilités générales pour l'expression du parallélisme. Il y a deux raisons à cela: d'une part le parallélisme est très exigeant en puissance de calcul, surtout avec les contraintes de temps réel liées à la commande de mouvements; d'autre part les facilités nécessaires à la Robotique ne sont pas clairement déterminées, et nous avons toujours cherché à éviter dans LM les constructions superflues. On notera d'ailleurs que l'expression et la gestion du parallélisme restent encore un sujet d'actualité de la recherche en Informatique "classique" et que les problèmes sont loins d'être tous résolus.

Un autre facteur qui limite les facilités de parallélisme dans LM est la portabilité du langage.

Les solutions apportées dans LM à ces limitations sont de deux types. D'une part, il existe des interfaces normalisées (donc indépendantes de l'implantation) de l'interpréteur qui permettent de tirer parti de certaines facilités offertes par le système d'exploitation; par exemple les tâches d'interruption ne sont pas actuellement programmables en LM, mais peuvent être implantées dans un autre langage et communiquer avec un programme LM grâce à l'interface ESVOIE. D'autre part, de nouvelles instructions LM sont en préparation (cf. paragraphe 9); elles visent toutes à accroître l'expression du parallélisme. Toutefois la nouvelle version de LM incluant ces facilités est en cours d'implantation sur une structure multiprocesseurs particulière et sa portabilité risque d'être moins grande que celle de l'interpréteur actuel.

D'autres langages offrent des facilités non offertes actuellement par LM, par exemple le sémaphore (VAL-II, AML, AL, ...), l'instruction CODEBUT-COFIN (AL, MCL), l'interface avec un réseau (VAL-II). Il est toutefois difficile de savoir si ces facilités sont liées à une implantation de l'interpréteur ou non. Par exemple, l'implantation de LM sur calculateur HP-1000 permet aussi l'interfaçage avec un réseau et l'appel des nombreuses primitives de synchronisation dont dispose le système d'exploitation RTE_A. Réciproquement, la clause IMMEDIAT présentée ci-dessus n'est pas disponible dans toutes les implantations de LM, en général pour des raisons de puissance de calcul.

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

La tendance actuelle, tant pour LM (cf. la thèse de J.M.Lefebvre en préparation), que pour d'autres langages tels que VAL-II est de bâtir l'interpréteur autour d'un noyau multi-tâches orienté "temps réel". La portabilité d'un langage dépendra alors essentiellement de celle du noyau.

9. EVOLUTIONS ACTUELLES DU LANGAGE

Les évolutions actuelles du langage LM concernent principalement le parallélisme et la synchronisation. Deux nouvelles constructions, les commandes gardées et les blocs parallèles, ont été motivées par le besoin croissant en Robotique de contrôler un nombre croissant de processus parallèles, et par la disponibilité d'architectures multiprocesseurs de coût raisonnable. Ainsi la nouvelle version du langage incluant les extensions introduites dans ce paragraphe est en cours d'implantation [Lefebvre 85] sur système CESAR (processeurs 68 000 sur bus VME modifié) avec le moniteur multi-tâches CLEOPATRE. CESAR et CLEOPATRE sont deux réalisations du Laboratoire CERT/DERA.

a) Les commandes gardées

Un objectif est d'étendre les facilités actuellement offertes par LM pour spécifier des mouvements guidés par des données de capteurs d'environnement. L'autre objectif est de permettre au programmeur d'orchestrer les réactions du programme à des événements asynchrones, typiquement des interruptions externes.

Des commandes gardées de la forme <garde>=><commande> sont les constructions utilisées pour atteindre les deux objectifs ci-dessus. L'expression <garde> est une condition booléenne, et <commande> est une instruction LM. Chaque fois que l'interpréteur rencontre une commande gardée, il substitue aux variables de la garde (autres que les variables d'états) leurs valeurs courantes. La garde ainsi modifiée est alors périodiquement évaluée en parallèle avec l'exécution du reste du programme. Si elle devient vraie, alors l'instruction <commande> est exécutée.

EVOLUTIONS ACTUELLES DU LANGAGE

Les commandes gardées peuvent apparaître au sein d'instructions DEPLACER, ou comme instructions d'un programme:

- Une commande gardée dans une instruction DEPLACER devient active lorsque le mouvement est physiquement commencé. Elle est désactivée lorsque le mouvement est terminé. Elle permet de décrire des modifications à apporter à la trajectoire en fonction de données sensorielles. Des commandes gardées imbriquées permettent d'implanter des mouvements à compliance active avec différents modèles de compliance [Lefebvre 82].

- Une commande gardée comme instruction d'un programme peut être utilisée pour définir la réaction à des événements extérieurs, tels que des interruptions. Elle peut être étiquetée, et des instructions ACTIVER et DESACTIVER permettent de définir des sections de programme non interrompibles; par exemple, un robot ne doit pas être interrompu pour décharger un robot mobile qui l'alimente en pièces alors qu'il est en train d'exécuter une opération de dépôt de colle.

b) Blocs parallèles

L'objectif est d'étendre les facilités actuellement offertes par LM pour coordonner plusieurs robots sous le contrôle du même programme. La version actuelle du langage peut gérer les opérations parallèles de plusieurs robots en utilisant la clause SANS ATTENTE. Cette facilité est appropriée lorsque les robots interagissent étroitement (par exemple l'un tient une pièce, pendant que l'autre réalise une opération sur cette pièce). En principe, elle permet aussi au programmeur de définir des sections de code distinctes à exécuter en parallèle; mais elle n'est pas vraiment adaptée à cette utilisation.

Il a été défini une instruction de la forme suivante:

```
CODEBUT
  [etiq 1] <bloc 1>
  [etiq 2] <bloc 2>
  ...
  [etiq N] <bloc N>
COFIN;
```

CHAPITRE 1: LES CONCEPTS DU LANGAGE LM

Elle spécifie que N blocs de programme doivent être exécutés en parallèle. Typiquement, mais pas nécessairement, chaque bloc correspond à un robot distinct. Deux autres instructions:

```
ENVOYER <message> A <etiq-bloc>;  
RECEVOIR <message> DE <etiq-bloc>;
```

servent à la communication entre blocs. Un message peut être une valeur; celle-ci est alors rangée dans une pile par l'instruction ENVOYER. Ce peut être aussi le mot réservé SIGNAL; le bloc émetteur se met en attente de la réception du message par le bloc récepteur.

EVOLUTIONS ACTUELLES DU LANGAGE

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

1 INTRODUCTION

Lors des études préliminaires à la réalisation de l'interpréteur de LM, il a paru naturel de chercher à mettre en place une architecture logicielle évolutive et adaptable. De plus, les contraintes liées à la commande de bras articulés en temps réel, qui demande des calculs importants tels que produits matriciels et calculs trigonométriques, imposaient que cette architecture soit performante.

Pour réaliser ces objectifs, il a été décidé de découper l'interpréteur de LM en deux parties distinctes. L'une transforme le code source (les programmes écrits en LM) en un langage intermédiaire numérique, appelé LM-E, plus adapté à une exécution rapide, et l'autre partie exécute le code obtenu. Ce découpage est classique en informatique: la première partie correspond à la phase de compilation et d'édition de liens, et la seconde à la phase d'exécution du programme. Le langage intermédiaire est le code d'une machine virtuelle, appelée "exécutif LM". L'intérêt de ce langage intermédiaire est d'une part méthodique (il sépare nettement traitement syntaxique et traitement sémantique), et d'autre part il facilite le transport et l'évolution de l'interpréteur du langage [Mazer 81]. Il peut toutefois être légèrement plus lent en temps d'exécution que le code directement exécutable qui serait engendré par un compilateur adapté à un matériel particulier.

Le code intermédiaire LM-E correspond au niveau d'interface sur lequel sont à l'étude de nombreuses propositions de norme. En effet, tant au niveau national qu'international, des tentatives sont faites pour homogénéiser les liens entre les différentes machines d'un atelier; les armoires de commande de robots en font bien entendu partie. Cette évolution est normale, et même souhaitable, mais, comme nous le verrons au paragraphe 4, elle ne résoud pas tous les problèmes d'interface.

La structure et l'organisation du code intermédiaire et de son interprétation par l'exécutif jouent un rôle important. Nous avons choisi une structure sous forme de code

INTRODUCTION

post-fixé, l'exécutif LM (ou interprète de ce code) étant donc organisé autour d'une machine à pile. Cette architecture est adaptée à une réalisation ultérieure sous forme d'une machine cablée ou microprogrammée.

La Figure II.1 montre les phases de traitement d'un programme en LM par l'interpréteur. Cette Figure correspond à l'utilisation de LM en mode "compilation". Comme nous l'avons vu dans l'Introduction de ce rapport (paragraphe 3), le langage LM est aussi utilisable en mode "interactif". Les modules composant l'interpréteur sont à peu de détails près les mêmes dans les deux modes; seule change la structure de contrôle définissant leur séquence d'appel. Pour simplifier ce chapitre, nous ne considérerons donc l'interpréteur que dans le mode "compilation" (ce qui correspond à la première version réalisée de l'interpréteur). Nous aborderons le mode interactif seulement au chapitre V lors de la description de l'environnement de programmation en LM.

Nous décrivons d'abord le compilateur et l'éditeur de liens (paragraphe 2), puis l'exécutif LM (paragraphe 3). L'ensemble de l'interpréteur LM a été conçu de façon à pouvoir être transporté sur plusieurs matériels (calculateurs et robots); dans le paragraphe 4, nous analyserons sa portabilité vis-à-vis du matériel informatique et son adaptabilité à différentes installations de robotique.

CHAPITRE 2: ARCHITECTURE INTERNE DE L INTERPRETEUR LM

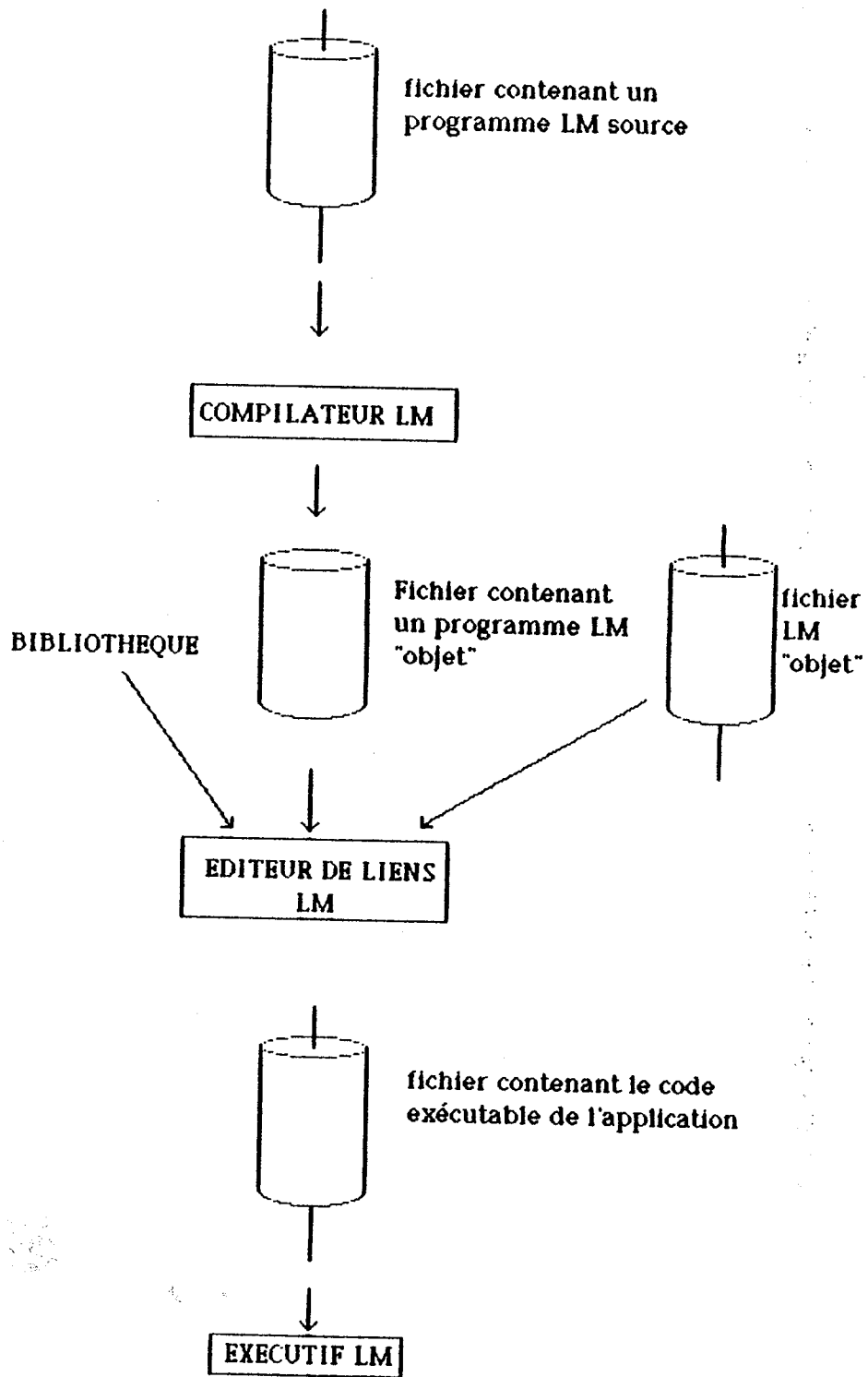


Figure II.1. Synoptique du traitement d'un programme par l'interpréteur LM.

2. COMPILATEUR ET EDITEUR DE LIENS

2.1. Architecture du compilateur

Le compilateur LM fait partie de la famille des compilateurs dirigés par la syntaxe. La grammaire du langage LM est une grammaire L1(1). Cette propriété permet une analyse syntaxique déterministe sans retour arrière, et descendante: pour déterminer la production de la grammaire à dériver, il faut et il suffit de connaître l'unité lexicale courante.

Néanmoins, une règle de la grammaire n'est pas de type L1(1): la règle <instruction-étiquetée> peut débuter par un identificateur suivi du symbole ":", (l'identificateur est alors une étiquette) ou, dans le cadre de l'instruction d'affectation, par un identificateur (cet identificateur est un identificateur de variable). Ce contexte de succession de symboles étant unique dans la grammaire LM, c'est l'analyseur lexical qui différencie une étiquette d'un identificateur.

L'analyseur syntaxique de LM a été produit grâce à l'utilisation d'un transformateur de grammaires L1(1) [Griffiths 69] [Delaunay 80]. Afin de prendre en considération les aspects contextuels et sémantiques du langage, ce transformateur de grammaires autorise l'insertion d'actions de compilation en tous points des règles de la grammaire. La grammaire est représentée en entrée sous la forme BNF-étendu (Bacchus-Naur forme) semblable à celle utilisée dans le rapport de définition du langage Pascal. Ces actions de compilation permettent en particulier la création et la consultation des tables d'identificateurs, le contrôle de l'utilisation des types et la génération d'un programme objet interprétable.

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

Exemple:

```
-----  
Soit la règle "instruction étiquetée":  
  instruction étiquetée::=  
  "étiquette" : <définition d'étiquette>  
  <instruction_étiquetée> <instruction_non_étiquetée>
```

L'action de compilation <définition d'étiquette> est appelée dans le contexte où une instruction est étiquetée. Cette action vérifie que l'étiquette correspondante est définie une fois et une seule. Elle doit également mémoriser le compteur de programme pour engendrer un branchement à chaque fois que l'on rencontrera une instruction ALLERA portant sur cette étiquette.

L'utilisation d'un transformateur de grammaire a grandement facilité la réalisation et la mise au point du compilateur. Cela a également permis de minimiser la taille de ce compilateur. En effet, le transformateur de grammaires utilisé produit des tables numériques de taille optimisée, contenant les règles de la grammaire LM et les références aux actions de compilation.

Le compilateur LM se compose ainsi de quatre modules principaux:

1- Un analyseur lexicographique

Son rôle est de reconnaître les éléments du vocabulaire terminal de la grammaire LM, en progressant dans le texte source.

2- Un module d'actions de compilation

Il contient l'ensemble des actions de compilation nécessaires à la réalisation des vérifications contextuelles et à la génération du programme "objet" en LM-E.

3- Un analyseur syntaxique

Son rôle est de vérifier la validité de la syntaxe du programme LM source en dérivant les règles de grammaire contenues dans les tables d'analyse. Il fait appel à l'analyseur lexicographique pour progresser dans le texte

COMPILATEUR ET EDITEUR DE LIENS

source et aux actions de compilation référencées dans les règles de grammaire utilisées. Il émet des messages d'erreurs lorsque les règles d'écriture ne sont pas respectées et appelle le module de recouvrement d'erreurs.

4- Un module de recouvrement d'erreurs

Ce module est appelé lorsqu'une erreur de syntaxe est détectée. Il permet de ne pas stopper l'analyse dès la détection d'une erreur, sauf en cas d'erreur fatale. L'algorithme de recouvrement d'erreur n'est pas très sophistiqué et peut être amélioré: actuellement, en cas d'erreur, le module permet de reprendre l'analyse à la première unité lexicale utilisable dans le contexte de dérivation courant.

La Figure II.2 décrit schématiquement le fonctionnement du compilateur. Le découpage en modules, facilité par l'utilisation du transformateur de grammaires, donne à l'ensemble une grande flexibilité. En effet, par exemple, la version anglaise du compilateur LM ne diffère de la version française que par la table des mots réservés. D'autre part, l'insertion de nouvelles instructions au langage ne requiert que l'écriture des nouvelles règles de grammaire et des actions de compilation correspondantes, et une éventuelle modification de l'analyseur lexicographique. Au cours du chapitre précédent nous avons vu l'intérêt d'une telle flexibilité pour la définition de nouvelles instructions de commande d'outils terminaux et de communication avec l'environnement. Les développements actuels de LM (commandes gardées, blocs parallèles) sont également facilités par cette flexibilité.

Le compilateur LM accepte en entrée des unités de compilation: programme principal, procédure, ou fonction. Diverses options de compilation permettent la sortie de la liste du programme compilé sur le terminal ou dans un fichier, la génération des numéros de ligne, l'utilisation de fichiers "include", etc ...

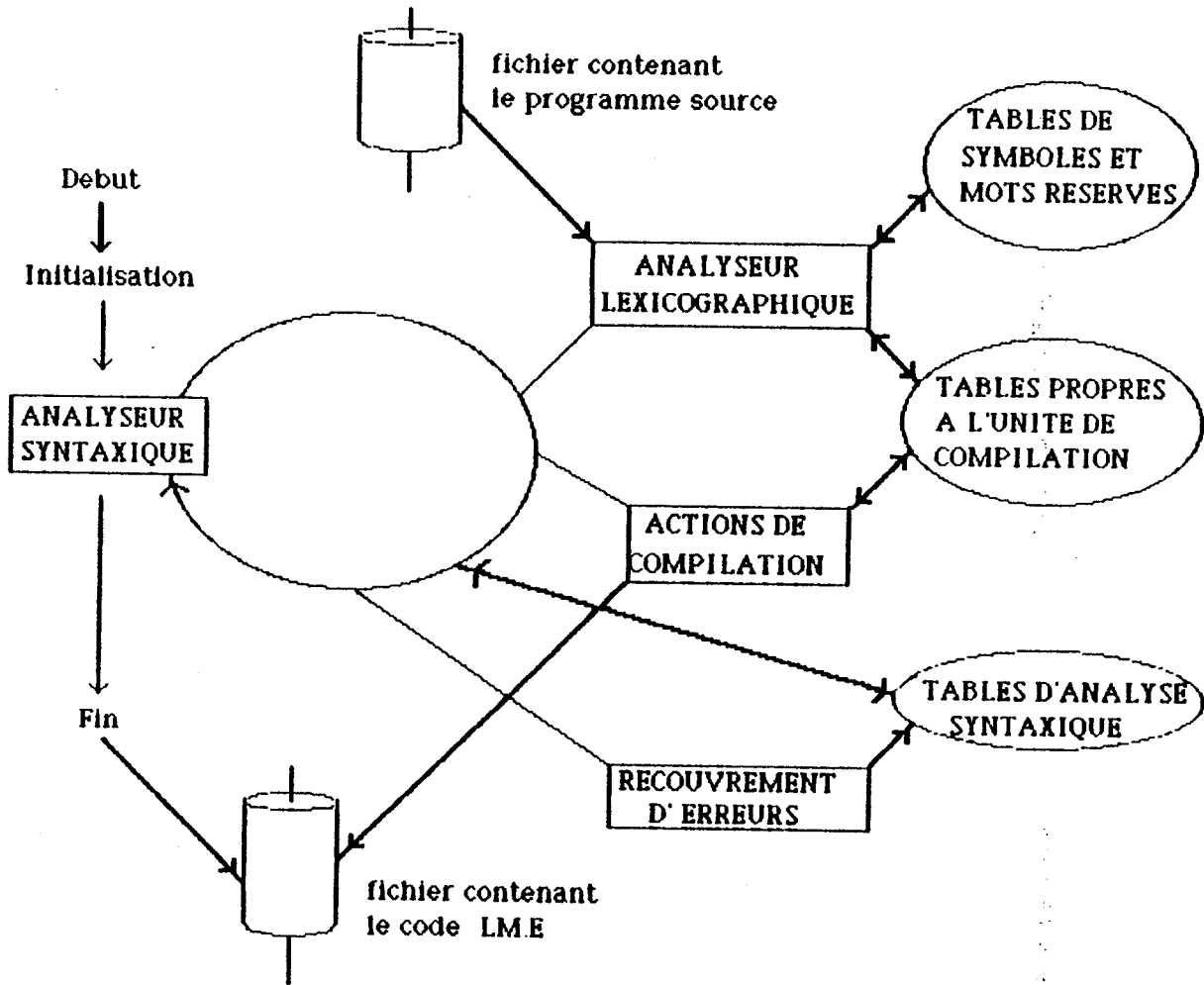
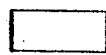



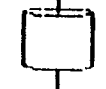


Figure 11.2: Synoptique de fonctionnement du compilateur LM

légende:

-  - code executable
-  - données
-  - activation possible
-  - liaison du code exécutable sur des données
-  - fichiers

COMPILATEUR ET EDITEUR DE LIENS

2.2. Format des instructions LM-E

Chaque instruction LM-E engendrée par le compilateur et interprétée par l'exécutif a une longueur fixe de 16 bits.

L'exécutif LM utilise une pile d'exécution. Nous avons choisi une évaluation post-fixée des expressions arithmétiques, qui s'accorde bien avec cette structure. Chaque type de donnée élémentaire possède son propre registre de base dans une unité de compilation. De même, les variables globales sont accessibles grâce à un groupe de registres globaux.

Les instructions peuvent être séparées en 7 grandes familles:

- instructions d'empilage,
- instructions de contrôle,
- instructions de déplacements, d'actions d'outils et de contrôle sur les déplacements et actions,
- instructions d'entrée/sortie sur terminal et fichier,
- instructions de communication avec l'environnement,
- instructions de test des conditions d'arrêt (clauses JUSQUA),
- instructions diverses.

Ces familles se reconnaissent à la valeur des quatre premiers bits de l'instruction. Le découpage et la signification des autres champs dépend de la famille reconnue.

Le fichier engendré par le compilateur (fichier "objet") contient toutes les instructions LM-E produites, les informations nécessaires à la construction des tableaux de bases d'adressage, les valeurs des constantes utilisées dans le programme, et les renseignements concernant les références externes au programme. Il est prêt pour l'édition des liens.

CHAPITRE 2: ARCHITECTURE INTERNE DE L INTERPRETEUR LM

Exemple:

Nous illustrons ci-dessous la génération de code LM-E sur un exemple simple de programme LM.

```
PROCEDURE PROC(TABLEAU ENTIER TAB);
```

```
ENTIER E1;
```

```
DEBUT
```

```
TAB(1);=E1;
```

Code LM-E engendré:

empiler valeur constante 1

empiler valeur entier paramètre TAB

empiler adresse élément de tableau paramètre

(cette instruction dépile 2 éléments et empile son résultat)

empiler valeur entier local E1

opération d'affectation sur entiers

(cette instruction dépile 2 éléments)

```
E1:=TAB(1)+2;
```

Code LM-E engendré:

empiler adresse entier local E1

empiler valeur constante 1

empiler valeur entier paramètre TAB

empiler valeur élément de tableau paramètre

empiler valeur constante 2

(cette instruction dépile 2 éléments)

opération d'addition sur entiers

(cette instruction dépile les 2 opérandes
et empile le résultat)

opération d'affectation sur entiers

(cette instruction dépile 2 éléments)

La génération des instructions, lors de l'appel de la procédure PROC, se fait de la façon suivante:

```
PROGRAMME MAIN;
```

```
EXT PROCEDURE PROC(TABLEAU ENTIER);
```

```
TABEAU(10) ENTIER TABENT;
```

```
DEBUT
```

```
PROC(TABENT);
```

Code LM-E engendré:

empiler adresse du descripteur de tableau TABENT

appel de la procédure PROC

...

COMPILATEUR ET EDITEUR DE LIENS

Remarques:

La longueur des instructions LM-E choisie (16 bits), conduit à trois remarques importantes:

1- Elle rend possible une implantation de l'exécutif LM sous forme d'une machine microprogrammée.

2- L'utilisation d'un registre de base pour chaque type permet de ne conserver que le numéro d'une variable, et non pas la valeur du déplacement par rapport à un registre de base unique. En effet, les éléments manipulés ont des dimensions différentes; par exemple un réel occupe 2 mots et une transformation entre 14 et 24 suivant la représentation choisie.

3- Les instructions de branchement utilisent un registre d'indirection pour pouvoir effectuer un branchement sur toute instruction du programme. Ceci nous a permis de réaliser un compilateur fonctionnant en un seul passage.

2.3 Editeur de liens

L'éditeur de liens a pour fonctions de vérifier les compatibilités des appels de procédures et fonctions, et de leurs paramètres, puis d'effectuer un préchargement et de produire un seul fichier, exécutable, qui contient directement la pile d'exécution initialisée pour l'application. Ce fichier peut ainsi être rapidement chargé par l'exécutif LM, et l'exécution du programme peut commencer.

La version de l'éditeur de liens que nous avons initialement réalisée présentait quelques restrictions importantes. Il était nécessaire de déclarer explicitement dans le programme principal toutes les en-têtes des procédures et fonctions externes utilisées dans l'application. De même, les variables globales devaient être déclarées dans le même ordre dans toutes les unités de compilation y faisant référence. Ces limitations ont été éliminées et de nouvelles fonctionnalités ont été ajoutées récemment qui, jointes à la création d'un bibliothécaire de procédures, simplifient considérablement la phase d'édition

de liens pour le programmeur.

3. EXECUTIF LM

3.1. Présentation

L'exécutif LM accepte en entrée un module exécutable sous forme de code numérique (LM-E). Cet exécutif remplit trois tâches principales:

- une tâche de fond, qui exécute les instructions de calcul, d'affectation, de contrôle, et de communication avec l'environnement,
- une tâche de gestion de robots, qui commande les mouvements du ou des robots et les actions de leurs outils,
- une tâche de surveillance, qui évalue périodiquement les conditions d'arrêt formulées par les options JUSQUA dans les instructions DEPLACER et de commande d'actions d'outils.

La tâche de gestion des robots est en fait décomposée plus finement en sous-tâches (une sous-tâche par mouvement ou action d'outil et par robot).

Pour remplir ces tâches, l'exécutif est composé de quatre modules:

- un séquenceur ("scheduler") qui gère le séquençement et la priorité des tâches ci-dessus,
- un module d'exécution pour chacune de ces tâches.

Nous détaillons ci-dessous le fonctionnement du séquenceur, ce qui nous permet de montrer comment sont activées les différentes tâches de l'exécutif et leurs priorités respectives. Nous présentons ensuite le fonctionnement de ces tâches, sauf la tâche de gestion de robots à laquelle nous consacrons le chapitre 3.

EXECUTIF LM

3.2. Le séquenceur

Le fonctionnement de l'exécutif LM repose sur une architecture multi-tâches dont la gestion est faite par le séquenceur. Ce séquenceur détermine, chaque fois que c'est nécessaire, la prochaine tâche à activer.

Nous avons choisi de réaliser nous-mêmes le module de gestion de tâches, plutôt que d'utiliser un système d'exploitation multi-tâches existant. En effet, nous voulions disposer d'une structure rapide et portable. L'exécutif LM est défini avec cette architecture pour fonctionner sur tout ordinateur monoprocesseur disposant d'une horloge à fonctions programmables. De plus le séquenceur ainsi défini en fonction de nos besoins a des performances qu'il aurait été difficile d'obtenir avec un système préexistant plus général.

Le nombre de tâches activables par le séquenceur est au maximum proportionnel au nombre de robots et d'outils commandés par l'exécutif LM. Dans un système LM mettant en jeu N robots, munis chacun d'un outil, le nombre de tâches maximum sera de $4*N+1$, soit:

- N tâches de surveillance des conditions d'arrêt de mouvement,
- N tâches de surveillance des conditions d'arrêt d'actions d'outils,
- N tâches de gestion de mouvements,
- N tâches de gestion d'actions d'outils,
- 1 tâche de fond.

Dans les versions de LM actuellement opérationnelles, ces tâches ont des priorités relatives fixées a priori. La priorité maximum est donnée aux tâches de surveillance des conditions d'arrêt, puis aux tâches de gestion des mouvements et des actions d'outils: la tâche la moins prioritaire est la tâche de fond.

Chaque tâche suivante peut se trouver dans l'un des trois états ENDORMI, PRET ou ACTIF. Elle dispose de sa propre horloge qui détermine lorsqu'elle veut passer de l'état ENDORMI à l'état PRET. Le séquenceur donne le contrôle à la

CHAPITRE 2: ARCHITECTURE INTERNE DE L INTERPRETEUR LM

tâche de plus haute priorité parmi celles qui sont dans l'état PRET; la tâche concernée rentre alors dans l'état ACTIF jusqu'à ce qu'elle rende le contrôle au séquenceur, et retourne à l'état ENDORMI, ou directement à l'état PRET suivant l'évolution de son horloge.

Lors du lancement de l'exécution d'un programme, seule la tâche de fond est à l'état PRET. C'est donc elle qui devient active en premier. Par la suite, une seule tâche au plus peut être dans l'état ACTIF (toutes les implantations actuelles étant "mono-processeur"), les autres étant soit dans l'état PRET, soit dans l'état ENDORMI.

Le choix du mode de fonctionnement du séquenceur que nous venons de décrire a des conséquences importantes sur l'implantation et le déroulement de l'exécutif LM. Les plus importantes sont les suivantes:

- La durée d'activation de chaque tâche doit être la plus réduite possible; c'est la durée d'activation la plus longue qui détermine le temps de cycle maximum de l'exécutif.
- Les interruptions sont à traiter de façon transparente au séquenceur (comme par exemple l'horloge). Elles ne peuvent pas activer directement une tâche de l'exécutif, mais ont accès aux horloges internes des tâches pour les faire transiter de l'état ENDORMI à l'état PRET. Les interruptions externes ont une priorité supérieure à l'exécutif LM et peuvent accéder à tous les modules et toutes les variables de l'exécutif (cf. paragraphe 4).
- Aucune tâche n'a la possibilité de se suspendre lorsqu'elle est active, autrement qu'en rendant le contrôle au séquenceur.
- Les risques de dégradation de l'exécutif sont minimes dans la mesure où l'utilisateur n'a pas accès directement aux différentes tâches. Comme de plus le séquenceur n'est pas activable par interruption, il n'est pas nécessaire de gérer des files d'attentes et des sauvegardes de contexte systématiques.
- Un module spécialisé, le "chien de garde", surveille l'évolution des horloges et des états des différentes tâches. Il signale toute anomalie soit en interrompant l'exécution du programme LM, soit par un message de compte-rendu en fin d'exécution.

3.3 La tâche de fond

La tâche de fond a pour fonction d'exécuter les instructions dites élémentaires du langage, c'est-à-dire toutes les instructions hormis celles qui commandent les déplacements de robots, les actions d'outils et l'évaluation de conditions d'arrêt.

Sa structure est très simple: elle dispose de son registre d'instruction, de son compteur ordinal, de sa mémoire propre (la pile d'exécution), et d'un ensemble de modules réalisant les opérations correspondant au jeu d'instructions exécutables. L'analogie avec la structure de l'unité centrale de micros-ordinateurs est voulue puisque nous nous sommes réservés la possibilité d'implanter l'exécutif LM sur un matériel spécialisé.

L'algorithme général de traitement est donc le suivant:

- 1- Remplissage du registre instruction (la zone des instructions est indiquée par le compteur ordinal),
- 2- Décodage des 4 bits de poids fort pour identifier la famille de l'instruction à traiter,
- 3- Appel de la procédure de traitement de la famille correspondante,
- 4- Incrémentation du compteur ordinal.

Les différentes familles d'instructions, décrites dans le paragraphe 2.2., font appel à des algorithmes opérant sur les données du programme en cours d'exécution. Ces algorithmes constituent la bibliothèque du système LM, et sont donc à la disposition de toutes les tâches LM, comme c'est le cas pour la bibliothèque "système" de tout système d'exploitation classique.

Le fonctionnement général du système LM, et en particulier celui du séquenceur, implique que le temps de traitement d'une instruction soit minime. Nous avons en effet dit qu'une tâche n'était pas interruptible par le séquenceur. Il existe quelques instructions élémentaires dont le temps d'exécution peut être trop long (c'est en particulier le cas des instructions qui traitent les liaisons entre repères). C'est pourquoi ces instructions ont été découpées en

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

plusieurs blocs; entre chacun d'eux, le séquenceur reprend le contrôle.

Il faut également ajouter que la tâche de fond a pour charge de rendre vivantes et de déclencher le premier passage de l'état ENDORMI à l'état PRET des autres tâches: par exemple, une instruction DEPLACER (au niveau LM-E) est tout d'abord décodée par la tâche de fond qui rend alors la main au séquenceur en lui signalant qu'un mouvement du robot doit être exécuté.

3.4. Les tâches de surveillance de conditions d'arrêt

Toutes les tâches de surveillance de conditions d'arrêt sont réalisées par un même module ayant en paramètre le numéro du robot concerné et éventuellement celui de son outil.

Le fonctionnement de ce module est tout à fait identique à celui de la tâche de fond. En effet, la condition d'arrêt correspond à une séquence d'instructions élémentaires identiques à celles que peut avoir à exécuter la tâche de fond. La dernière instruction de cette séquence est un arrêt de l'action donnée en paramètre (mouvement de robot ou action d'outil) conditionné à l'évaluation à VRAI de la condition. Une tâche de surveillance reste active tant que la condition d'arrêt n'a pas été évaluée complètement.

Ainsi, l'algorithme du module exécutant une tâche de surveillance est le suivant:

- 1- Réinitialisation de l'horloge propre du module (pour réactivation périodique).
- 2- Mise en place des paramètres du module.
- 3- Interprétation de la séquence d'instructions correspondant à l'évaluation de la condition.
- 4- SI (condition-vraie) ou (action correspondante terminée) ALORS arrêter l'action considérée (si ce n'est pas déjà fait) retour au séquenceur avec passage à l'état ENDORMI et blocage de l'horloge (la tâche se "tue")
SINON retour au séquenceur sans blocage de l'horloge.

Cet algorithme prend en compte l'utilisation d'une horloge particulière pour chaque tâche de surveillance. Cette

EXECUTIF LM

horloge est un compteur qui, si la tâche est vivante, se "décrémte" de façon périodique. Le passage à 0 du compteur signifie que la tâche correspondante rentre dans l'état "PRET". Une tâche de surveillance est vivante tant que son compteur évolue, ce qui veut dire qu'elle doit être réactivée. Le blocage de ce compteur peut être effectué par la tâche elle-même, comme le montre l'algorithme précédent, et uniquement par elle.

La valeur à laquelle le compteur est réinitialisé est proportionnelle au paramètre définissant la fréquence de l'évaluation de la condition d'arrêt (rappelons que cette fréquence peut être spécifiée par le programmeur, cf paragraphe 6 du chapitre 1).

3.5. Le chien de garde

Le module "chien de garde", mentionné au paragraphe 3.2, a pour rôle de contrôler l'évolution des différents compteurs et, dans le cas d'un mauvais fonctionnement du système, d'en avertir l'utilisateur. En effet, les tâches de surveillance sont les plus prioritaires, alors que leur temps d'exécution et la fréquence de surveillance sont sous le seul contrôle du programmeur. Rien n'empêche le programmeur d'une part de mettre une condition dont l'évaluation demande un temps d'exécution long et d'autre part de demander une fréquence d'évaluation élevée. On peut alors arriver à la situation paradoxale où une condition d'arrêt est sans cesse testée sans que le mouvement de robot (ou l'action d'outil) ne soit exécuté.

4. PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

4.1. Définitions et présentation

Les termes de portabilité et d'extensibilité peuvent avoir des significations différentes suivant la façon dont on juge un système informatique. C'est pourquoi il est important de préciser ce que nous entendons par portabilité et flexibilité du système LM.

a) Portabilité

Le critère de portabilité d'un logiciel informatique se juge sur la facilité de transport de ce logiciel d'un ordinateur à un autre. Il existe des moyens simples d'assurer la portabilité d'un logiciel. Les plus classiques sont:

- choisir un langage d'implantation universellement répandu et ayant fait l'objet d'une normalisation internationale,
- utiliser des fonctions système classiques, que l'on est assuré de rencontrer sur tout système d'exploitation,
- regrouper les instructions et appels des fonctions spécifiques dans des modules séparés, bien commentés,
- utiliser un codage interne indépendant de la machine en taille et en format.

La portabilité d'un système de programmation de robots se juge en plus sur la facilité d'adaptation de ce logiciel à de nouveaux robots. Dans ce domaine, peu de travaux ont été réalisés jusqu'à maintenant, les systèmes de programmation de robots étant le plus souvent réalisés par des constructeurs de robot et/ou d'armoires de commande. Certains critères sont toutefois évidents, comme par exemple de définir un langage permettant une description des opérations indépendante de la structure physique du ou des robots commandés, et de traiter cette description indépendamment des robots jusqu'à un aussi "bon" niveau que possible.

PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

Remarque: Nous nous intéressons ici à la portabilité du système de programmation, et non à la portabilité des programmes d'application écrits dans le langage de programmation du système.

b) Extensibilité

L'extensibilité d'un système de programmation se juge sur les outils dont dispose potentiellement ce système pour d'une part profiter au mieux de la machine sur laquelle il est implanté et, d'autre part, intégrer des possibilités nouvelles telles que de nouveaux outils (périphériques d'ordinateur, etc...), ou des liaisons soit avec d'autres applications sur le même calculateur, soit vers l'extérieur.

Pour un système de programmation de robots, il est en particulier souvent nécessaire de connecter de nouveaux capteurs, ce qui peut mettre en jeu des entrées/sorties logiques, ou analogiques, classiques, mais aussi des systèmes de vision complexes. De même des applications spécifiques peuvent nécessiter l'intégration d'outils spécialisés (pince asservie à plusieurs mors, visseuse, étau, table x-y...). La flexibilité d'un système de programmation de robots se juge donc également sur la facilité avec laquelle ces adaptations sont possibles.

La Figure II.3 présente les niveaux d'interfaces que nous avons distinguées pour l'exécutif LM. Nous reviendrons en détails sur les modules d'interface spécifiques à LM dans les paragraphes suivants, mais la distinction de ces niveaux nécessite ici deux commentaires complémentaires:

- Nous pensons que la portabilité et l'extensibilité d'un système de programmation passe par la définition rigoureuse d'interfaces entre l'exécutif du langage et l'équipement commandé. Nous avons donc défini les interfaces RACORD, MODFON et ESVOIE, qui sont normalisées dans le sens où elles sont identiques dans toutes les nouvelles implantations de LM.

- Jusqu'à ce jour, les comités de normalisation dans le domaine de la programmation des robots ne semblent pas s'être véritablement intéressés à ces interfaces, mettant plus l'accent sur les interfaces entre l'armoire de commande du robot (contenant l'exécutif du système y compris l'asservissement), et des systèmes informatiques extérieurs. Ces interfaces se situent au même niveau conceptuel que

CHAPITRE 2: ARCHITECTURE INTERNE DE L INTERPRETEUR LM

l'interface niveau 1- niveau 2 de la figure II.2: le code LM-E.

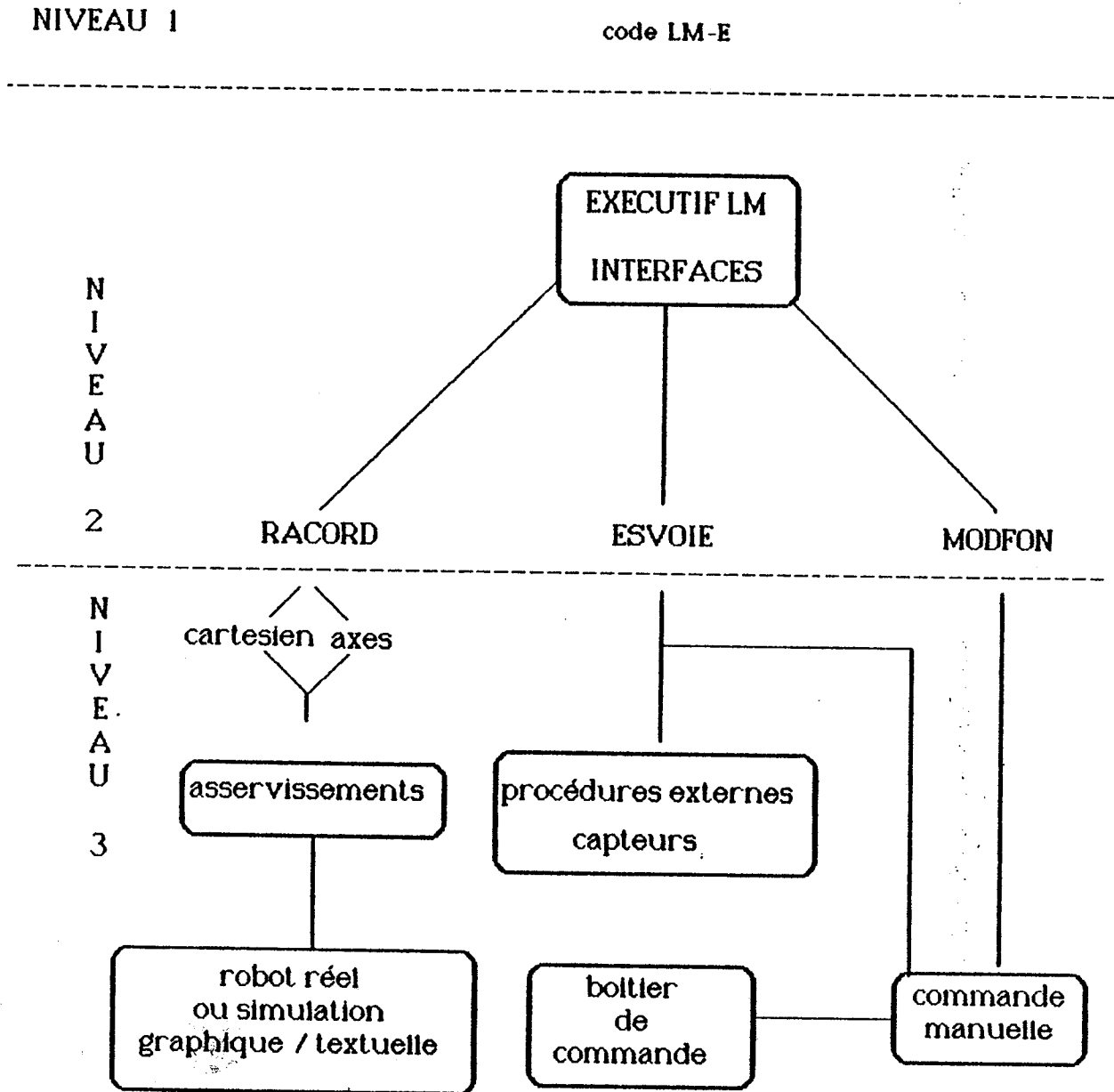


Figure II.3 : Interfaces de l'exécutif LM.

PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

4.2. Portabilité du système

La portabilité de LM s'applique à deux niveaux: au niveau du calculateur hôte, et au niveau du ou des manipulateurs commandés.

1- Au niveau du calculateur hôte:

A ce niveau, les moyens mis en oeuvre correspondent à ceux donnés au paragraphe précédent en ce qui concerne la portabilité d'un système informatique classique:

- Le langage d'implantation de LM est Fortran IV, conforme aux normes ANSI de 1974. Des compilateurs Fortran sont disponibles sur l'immense majorité des calculateurs actuels, et sont souvent très performants, puisque dédiés à l'informatique scientifique.

- Nous avons montré au paragraphe 3 la structure particulière de l'exécutif LM. Elle ne nécessite que peu d'appels à des fonctions système: il faut que le système d'exploitation permette l'utilisation de procédures d'interruption programmables avec horloge, et qu'il assure aussi l'exécution en multi-tâches, en particulier pour la gestion des mouvements (cf. chapitre III).

- Les instructions et appels de fonctions spécifiques ont été regroupés au sein de modules séparés. Ceci concerne en particulier toutes les procédures d'initialisation, et toutes les opérations de lecture et d'écriture sur terminal ou sur fichier.

- Le compilateur LM, comme nous l'avons dit au paragraphe 2, est un compilateur traducteur. Il génère un code intermédiaire de type postfixé (le code LM-E) qui est interprété par l'exécutif LM. Ce code intermédiaire est donc indépendant du code machine du calculateur hôte.

Le respect de ces règles de portabilité nous a permis de réaliser des implantations de LM sur plusieurs calculateurs. L'expérience montre que le compilateur et l'éditeur de liens peuvent être implantés sur un nouveau calculateur en très peu de temps. L'exécutif LM requiert, pour sa part, plus de travail en raison de sa structure. Quelques sous-programmes sont écrits en langage d'assemblage, mais ils sont aisément transposables sur de nouvelles machines. Aujourd'hui, LM

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

est opérationnel sur plus d'une demi-douzaine de calculateurs différents (cf. chapitre IV).

2. Au niveau du robot

Nous avons mis progressivement en place une interface normalisée permettant de raccorder un robot très rapidement au système. Cette interface, baptisée "RACORD", existe suivant deux versions; la première version se situe au niveau cartésien, et la seconde au niveau des coordonnées articulaires du robot.

a) RACORD niveau cartésien:

La charnière entre l'espace cartésien dans lequel sont décrits la plupart des mouvements d'un robot et l'espace des coordonnées propres (ou articulaires) de ce robot utilise les transformateurs de coordonnées dits inverse et direct. Ces algorithmes travaillent sur un repère particulier, dit repère extrémité du robot et noté CP ci-dessous. C'est pourquoi l'information transmise par l'interface RACORD niveau cartésien concerne toujours ce repère, indépendamment du repère ayant servi à spécifier le mouvement dans l'instruction DEPLACER.

Par exemple, un déplacement suivant un "segment libre" (cf. Chapitre I) correspond à l'envoi par LM de la position cartésienne finale du repère CP, assorti du coefficient de vitesse souhaité sur le mouvement. Les modules situés en aval de RACORD ont la charge de réaliser l'échantillonnage linéaire dans l'espace des coordonnées propres et la synchronisation entre les axes.

Un déplacement suivant un "segment cartésien" (ou "circulaire", ou "planifié") correspond quant à lui à l'envoi, à chaque point d'échantillonnage, de la position, l'orientation et la vitesse désirée du repère, assorti du délai d'exécution de la commande (période d'échantillonnage).

Les modules situés en aval de RACORD ont la charge de réaliser l'échantillonnage linéaire dans l'espace des axes, et la synchronisation entre les axes, pour chaque parcours situé entre deux points d'échantillonnage dans l'espace

PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

cartésien.

Les informations qu'il est nécessaire de fournir au système LM pour mettre en place cette interface avec un nouveau robot sont les limites de vitesses et accélérations cartésiennes du robot, ainsi que les transformations directe et inverse de passage du repère CP au repère ROBOT(i) de LM (CP et ROBOT(i) peuvent être le même repère, mais cela n'est pas toujours judicieux).

b) RACORD niveau axes:

Au contraire de l'interface "niveau cartésien", l'interface "niveau axes" est résolument dépendante du manipulateur à commander. Elle constitue la frontière entre l'asservissement "de base" et le reste du système. Par asservissement "de base", nous entendons un ou plusieurs modules capables de réaliser l'asservissement de chaque axe dans l'espace de ses coordonnées propres, en recevant des consignes à intervalles de temps constants (typiquement entre 20 et 200 millisecondes).

Ce découpage a plusieurs avantages. D'une part la tâche d'asservissement est bien cernée et gagne ainsi en robustesse; d'autre part les informations transmises sont de même nature quel que soit le mode de trajectoire du mouvement concerné; enfin, la taille du système total est plus réduite, certains modules des tâches d'échantillonnage pouvant s'appliquer au mode libre comme aux autres modes (cf. chapitre III).

Les valeurs transmises par LM au travers de l'interface RACORD "niveau axes" sont réduites à des doublets $(\theta_i, \dot{\theta}_i)$, pour $i=1, N$, où θ_i est la position codeur désirée de l'axe i , $\dot{\theta}_i$ sa vitesse (avec N = nombre d'axes du manipulateur).

Les informations qu'il est nécessaire de fournir au système LM pour mettre en place cette interface avec un nouveau robot sont les éléments nécessaires à l'interface "niveau cartésien", les limites de vitesse et d'accélération sur chaque axe, et les valeurs des butées logicielles (limites sur chaque axe en général incluses dans les butées matérielles pour des raisons de sécurité).

Ces deux niveaux d'interface "RACORD" ne sont pas les seuls possibles, mais correspondent à des découpages de fonctions

CHAPITRE 2: ARCHITECTURE INTERNE DE L'INTERPRETEUR LM

naturels. On peut toutefois se poser la question de l'intérêt de l'interface "niveau cartésien". Deux raisons nous y poussent:

- Nous avons souvent eu à implanter LM sur des systèmes déjà existants, comportant quelques possibilités d'apprentissage et de programmation, ou sur des systèmes dont l'asservissement incluait déjà les changements de coordonnées. L'interface "niveau cartésien" est dans ces cas là la plus adaptée.

- De même que nous souhaitons conserver à LM son caractère d'outil général, certains industriels et chercheurs, qui souhaitent le voir installé sur leurs robots, veulent pour des raisons qui peuvent être stratégiques, techniques, ou autres, conserver la maîtrise de la commande "de bas niveau" des robots.

4.3. Extensibilité

Nous avons développé deux interfaces particulières: l'interface MODFON et l'interface ESVOIE. La première a pour rôle de permettre d'agir sur les modes de fonctionnement de l'exécutif LM; la seconde joue un rôle plus général puisqu'elle permet de se connecter à l'environnement de l'exécutif LM, soit pour ajouter des facilités au langage (par exemple: communication avec l'environnement, contrôle d'outils terminaux), soit pour utiliser des modules externes indépendants écrits dans d'autres langages.

a) L'interface MODFON

L'exécutif LM offre plusieurs modes de fonctionnement qui permettent à un opérateur de choisir le contrôle qu'il désire avoir sur l'exécution de son programme. Ces modes sont les suivants:

* Le mode "pas à pas". Il correspond à une découpe de l'exécution du programme par actions élémentaires. Une action élémentaire correspond à une modification physique du matériel, par exemple un mouvement (ou une partie d'un mouvement dans le cas d'un mouvement avec plusieurs segments), ou une action d'outil (par exemple: ouverture ou

PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

fermeture de pince).

* Le mode "ligne à ligne". Il correspond à une découpe exacte de l'exécution par rapport aux lignes successives du programme source LM.

* Le mode "cycle par cycle". Il correspond à la découpe faite par le programmeur dans son application grâce à l'instruction LM "CYCLE - FINCYCLE" (cf. Manuel de Référence)

* Le mode "automatique". Le cycle est répété de façon automatique tant que le mode n'est pas modifié.

Le passage d'un mode à un autre peut se faire à tout moment, et la progression pour les modes autres que "automatique" se fait par validation. L'opérateur dispose pour cela d'un boîtier manuel comportant les touches de fonction appropriées.

b) L'interface ESVOIE

Nous avons présenté en début de ce paragraphe (paragraphe 4.1.b) comment se juge la flexibilité d'un système de programmation, et nous avons montré que ce critère de flexibilité est particulièrement important pour un système de programmation de robots. C'est pour donner cette flexibilité au système LM que nous avons mis en place l'interface "ESVOIE".

Cette interface est en fait la clé qui permet l'ouverture du système vers l'extérieur: elle regroupe toutes les entrées-sorties sur voies, et tous les appels à des fonctions particulières disponibles sur un système d'exploitation. Il est également possible de relier l'exécutif LM à tout dispositif interne ou externe (raccordement de nouveaux outils ou capteurs, liaisons avec d'autres calculateurs ou des automates programmables par des protocoles de réseaux, etc ...).

Le fonctionnement et les règles de transfert d'information et de contrôle sont simples: cette interface est appelée chaque fois que l'exécutif LM exécute une instruction de lecture ou écriture sur voie, ou un appel à une fonction capteur ou à une variable d'état banalisée. L'interface donne en paramètre le numéro de la primitive concernée

CHAPITRE 2: ARCHITECTURE INTERNE DE L INTERPRETEUR LM

(capteur, numéro de voie, etc...), le type de l'information transmise ou à retourner et des informations de contrôle (sens du transfert, retour d'erreur, ...).

Par exemple, l'instruction LM: ECRIRE 1 SUR VOIE VERROU_1; correspond à l'activation de ESVOIE avec les paramètres suivants:

- numéro de canal concerné: VERROU_1 (valeur entière)
- écriture
- valeur à transmettre: 1 (valeur entière)

L'instruction LM: LIRE ETAT_VERROU SUR VOIE VERROU_1; correspond à l'activation de ESVOIE avec les paramètres suivants:

- numéro de canal concerné: VERROU_1 (valeur entière)
- lecture
- valeur à retourner (du type de la variable ETAT_VERROU)

[Remarque: Cette seconde instruction est équivalente à
ETAT_VERROU:= SIGN-BOO(VERROU_1);
avec ETAT_VERROU variable de type booléen.]

Pour les appels des fonctions capteurs, les numéros de voie (FX, FY, FZ, ...) sont fixés a priori, et les numéros de voies des variables d'état banalisées sont fournies en paramètres de l'appel dans le programme LM.

Dans l'exemple précédent, nous pouvons par exemple remplacer l'instruction LIRE par:

```
ETAT_VERROU:= SIGN-BOO(VERROU_1);  
(avec ETAT_VERROU variable de type booléen).
```

L'appel de FX(1) dans une expression LM correspond à l'activation de ESVOIE avec les paramètres suivants:

- numéro de canal concerné: 2048 (prédéfini par l'interface)
- lecture
- valeur à retourner: Réel.

A FX(2) correspond le numéro de canal 2048+128, à FY(2) le numéro 2049+128, etc...

PORTABILITE ET EXTENSIBILITE DU SYSTEME LM

Cette interface ESVOIE s'est avérée essentielle dans la réalisation d'installations industrielles: il est possible ainsi d'engendrer des exécutifs LM faits "sur mesure" avec l'avantage d'avoir une frontière tangible entre le fonctionnement propre de l'exécutif LM et le fonctionnement des modules spécifiques d'une armoire de commande, ou même d'une application particulière.

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

1. INTRODUCTION

Dans ce chapitre nous définissons l'architecture informatique pour l'interprétation de l'instruction DEPLACER de LM. De même que cette instruction permet de combiner de multiples façons un nombre restreint de concepts, son interprétation est réalisée par un nombre réduit de modules différents pouvant avoir des interactions complexes. Nous pensons que l'architecture proposée présente une généralité dépassant la seule interprétation de LM et qu'elle peut servir de base à la construction d'autres systèmes de commande de mouvements de robots.

Jusqu'à présent la plupart des travaux sur la commande de mouvements ont été consacrés à des aspects ponctuels relatifs à cette commande. Par exemple, [Taylor 79] propose une méthode de suivi de trajectoire cartésienne garantissant un écart donné par rapport à une droite de référence, [Featherstone 83] un algorithme permettant d'améliorer la rapidité des calculs de transformation de coordonnées pour un manipulateur rotoïde, [Hollerbach 82] une méthode efficace de calcul des équations dynamiques, [Renaud 84] des algorithmes permettant d'engendrer automatiquement les équations nécessaires au changement de coordonnées. Les modes de commande des manipulateurs ont été étudiés en particulier par [Renaud 84], [Paul 81], et [Luh 83].

Peu d'auteurs consacrent une partie de leur étude à l'intégration des modules de traitement qu'ils ont élaborés dans un système complet de commande de robot. Cette lacune résulte en une réduction sensible de la portée pratique de certains algorithmes. C'est le cas notamment de plusieurs algorithmes de génération de trajectoire qui ne peuvent être utilisés que dans des calculs s'effectuant hors ligne ou presque hors ligne (90 pour cent du temps de calcul étant passé avant que le mouvement ne soit physiquement commencé).

La plupart des systèmes de commande de mouvements de robots utilise une architecture informatique linéaire. Cette architecture comporte généralement trois ou quatre modules (figure III.1):

INTRODUCTION

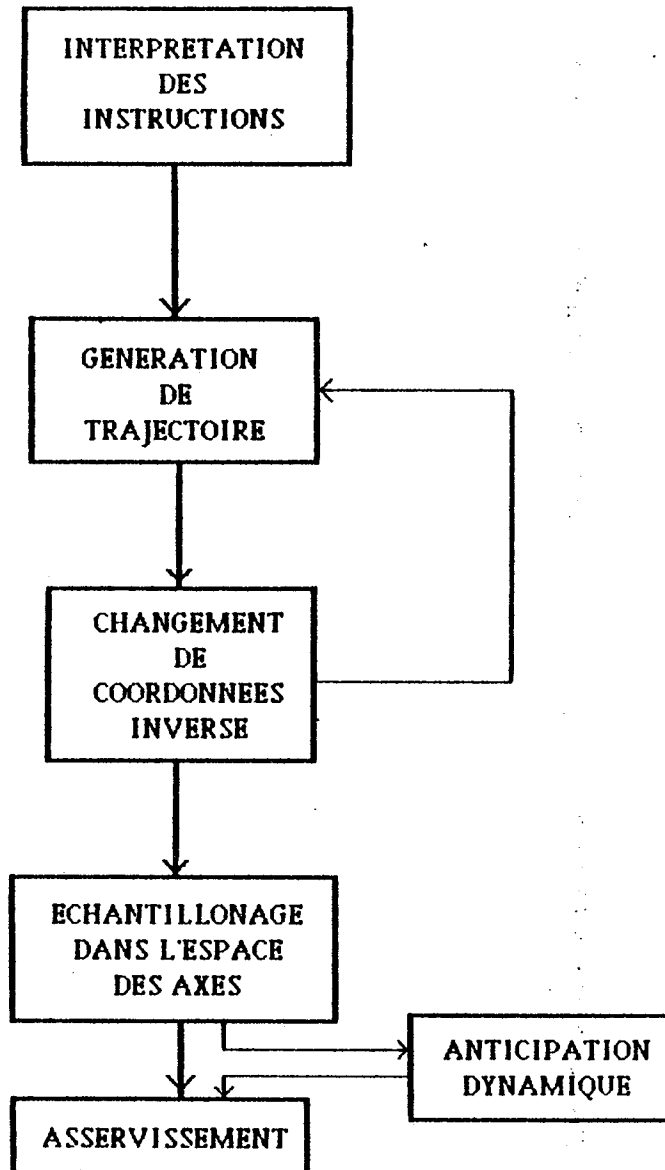


Figure III.1

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

- un module chargé de calculer le déplacement dans l'espace cartésien,
- un module chargé de la transformation de coordonnées,
- un module d'échantillonnage dans l'espace des coordonnées des axes du robot,
- un module de calcul des forces généralisées, si l'on réalise une commande dynamique.

Le système peut permettre au flot d'information de traverser ces différents modules de façon continue (mode pipe-line), pour réduire le temps global de calcul nécessaire à l'envoi d'une consigne aux asservissements des actionneurs. Ce type d'architecture très simple se heurte à de nombreuses difficultés dans un système réel tel que l'interpréteur LM; en effet, dans un tel système, il ne permet pas de faire face à la combinatoire engendrée par la diversité des options, des conditions initiales, et des événements asynchrones.

C'est pourquoi nous avons conçu pour LM une architecture plus élaborée. Cette architecture, dont les modules sont montrés à la figure III.2 est décrite dans ce chapitre. Le paragraphe 2 en présente les principaux concepts, les paragraphes suivants traitent de chaque module séparément.

2. PRINCIPE DE L'ARCHITECTURE

L'architecture développée s'inspire des concepts suivants:

- a) la programmation modulaire, la notion d'objet:

Le système proposé est construit à partir de modules d'interprétation pouvant s'échanger des messages. Ils ressemblent à des objets Smalltalk [Cointe 83], certains de ces modules étant des instanciations d'un même type d'unité de traitement (générateur de trajectoire par axe). Ils peuvent apparaître similaires de l'extérieur (partie visible), mais utiliser des méthodes d'interprétation différentes (partie cachée), comme c'est le cas pour les modules de changement de coordonnées. Des langages de type LMAC [Rousseau 83] ou Smalltalk [Cointe 83], fournissent les

PRINCIPE DE L ARCHITECTURE

technologies logicielles nécessaires pour implanter facilement ce type de structure; ils se heurtent cependant au problème de rapidité d'exécution. Il est aussi possible d'utiliser les ressources offertes par un système d'exploitation pour définir ces modules sous forme de tâches, et les faire communiquer par des messages transitant par le système comme le fait par exemple RCCL [Hayward et al 84].

b) La programmation concourante, le parallélisme:

Commander plusieurs degrés de liberté et assurer leur synchronisation dans le temps est typiquement un problème de parallélisme. De nombreux algorithmes concernant le contrôle des déplacements utilisent la notion de "boucle" pour calculer différents paramètres sur chaque degré de liberté; ainsi pour calculer le temps mis par chaque degré de liberté pour atteindre une position donnée, la méthode en général employée consiste à écrire:

```
pour i=1 jusqu'à 6
faire
temps(i)=calcul_temps(pc(i),pf(i),temps(i))
finfaire
```

La notion de processus parallèle décrite dans CSP [Hoare 78] permet de remplacer cette boucle par 6 processus parallèles calculant chacun le paramètre du degré de liberté correspondant. La forme syntaxique de ce type de commande peut être la suivante:

```
<calcul_temps(pc(1),pf(1),temps(1))>;
<calcul_temps(pc(2),pf(2),temps(2))>;
<----->
<calcul_temps(pc(6),pf(6),temps(6))>;
```

Ce type de description correspond mieux à la nature du problème: calculer 6 paramètres indépendants. L'interprétation de ces deux types de programmes sur un mono-processeur n'apporte rien d'un point de vue de l'efficacité, cependant l'introduction de machines multi-processeurs et de systèmes d'exploitation spécialisés tels que CÉSAR et CLEOPATRE (cf. chapitre I paragraphe 9) changent considérablement la manière dont les programmes doivent être pensés. L'utilisation de langages tels que CSP permet de structurer ces nouvelles méthodes de conception de programmes et conduit à l'écriture de programmes pouvant

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

s'exécuter facilement sur des machines multi-processeurs. Nous avons essayé de réaliser un découpage en modules favorisant l'exécution parallèle, certains de ces modules pouvant eux-mêmes être découpés en processus parallèles.

La figure III.2 représente l'ensemble des modules servant à la commande de mouvements dans l'interpréteur LM. Les arcs entre les différents modules représentent des canaux par lesquels transitent les messages.

Dans les paragraphes suivants, nous décrivons le système en terme des fonctions des modules représentés et de leurs interactions. La décomposition choisie repose en partie sur l'expérience accumulée lors des implantations successives de LM; elle se justifie aussi sur des critères de portabilité.

Pour chaque module nous préciserons:

- la fonction,
- le mode d'activation,
- les connexions avec l'environnement,
- les constantes particulières à une instanciation ou dépendant d'un robot donné,
- les principes des algorithmes utilisés.

Un module peut être activé de trois manières différentes:

- l'activation par requête

Une requête correspond à peu près à l'appel d'un sous-programme, à la différence qu'il n'y a pas forcément retour dans le module appelant.

- l'activation périodique

Le module est lié à une horloge qui l'active périodiquement. Plusieurs fréquences de réveil sont présentes dans le système: la période d'activation du module d'asservissement est de l'ordre de 1000hz, alors que celle des modules de suivi de trajectoire est de l'ordre de 50hz.

- l'activation périodique après requête

La première activation a lieu sur une requête d'un autre module, le module étant ensuite activé périodiquement. Il est aussi possible de supprimer une horloge associée à un module.

PRINCIPE DE L ARCHITECTURE

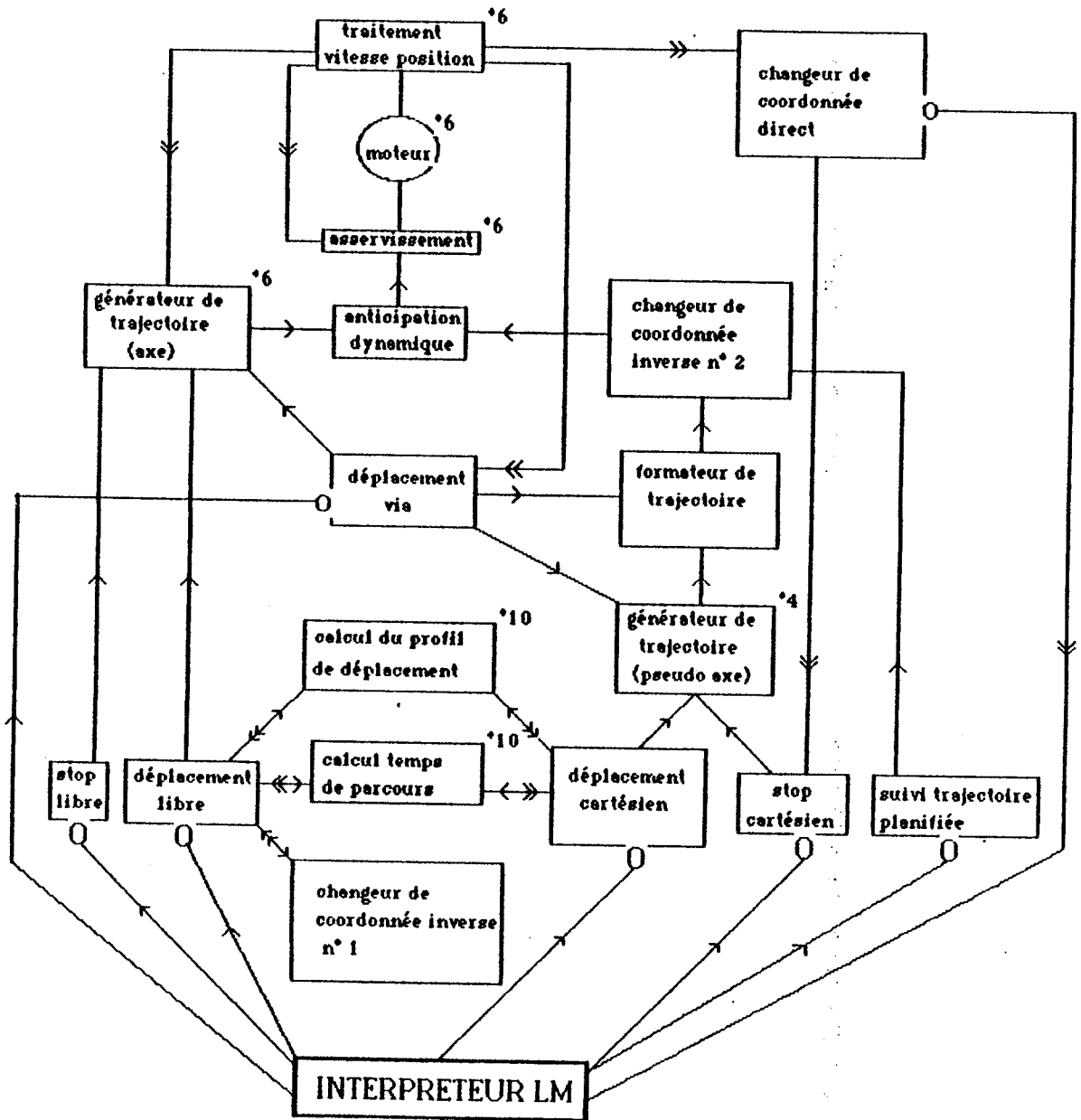


Figure III.2

Architecture du contrôle des déplacements

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

Les communications entre modules peuvent prendre trois formes:

- Envoi d'un message simple

Si le module 1 envoie un message vers le module 2 on représente ce type de communication par:

1 ----->----- 2

- Envoi d'un message avec attente d'un message de réponse

Si le module 1 envoie un message vers le module 2 en attendant une réponse (appel de sous programme) on note ce type de communication par:

1 ----<<---->----- 2

- Prise asynchrone d'une valeur

Le module 1 prend dans le module 2 une valeur sans se synchroniser sur la fin de traitement du module 2 (mémoire partagée). On note ce type de communication par:

1 ----->>----- 2

Nous avons choisi certaines conventions de notation que l'on retrouve dans tous les modules:

- le repère de référence utilisé par les changeurs de coordonnées sera noté CP (Centre du Poignet), car il correspond effectivement à la position communément calculée pour des structures mécaniques à 6 degrés de liberté.

- les positions et vitesses des axes sont notées θ et $\theta..$. Leurs valeurs s'expriment de façon cohérente avec celles utilisées par les changeurs de coordonnées.

Nous avons également sous entendu dans l'architecture globale l'utilisation d'une loi de vitesse trapézoïdale, en particulier dans les formats des messages échangés; la loi choisie peut être quelconque, mais doit rester homogène dans tout le système.

Comme [Paul 81], nous avons défini pour les déplacements cartésiens 2 axes virtuels, l'un en translation, l'autre en rotation; la combinaison de ces deux déplacements correspond à la sémantique des mouvements en mode cartésien tels qu'ils sont définis en LM.

PRINCIPE DE L ARCHITECTURE

Chaque module a été supposé ne retournant aucune erreur. Il faudrait ajouter à cette architecture un module spécialisé pour le recouvrement et la transmission des erreurs. Il manque également la partie concernant l'interprétation des mouvements en mode circulaire qui est directement calquée sur celle concernant les mouvements cartésiens. D'autre part, il faudrait également rajouter une partie pour contrôler des déplacements asservis à des données de capteurs (cf. chapitre I paragraphe 8) qui ne sont pas encore intégrés dans la syntaxe de LM.

3. MODULE DEPLACEMENT LIBRE

Fonction:

Ce module permet de commander tous les mouvements suivant des segments libres autorisés par la syntaxe du langage LM.

Il peut être activé alors que le manipulateur est en mouvement, permettant ainsi de changer de trajectoire ou de vitesse (option "IMMEDIAT"). Son mode d'activation (type 1) permet à l'interpréteur LM de reprendre le contrôle sans attendre la fin du mouvement (option "SANS ATTENTE").

Mode d'activation:

Par requête avec les paramètres d'appel suivants:

- la position finale PF du repère CP en terme des coordonnées cartésiennes,
- le torseur de vitesse TVCF à la position finale PF,
- le coefficient C de vitesse du robot.

Connexions avec les autres modules:

- Interpréteur des commandes LM, qui envoie une requête.
- modules "Calcul de Temps de Parcours", qui permettent de déterminer le temps de parcours sur chacun des axes à vitesse constante.
- modules "Calcul Profil de Déplacement", qui calculent les instants de fin d'accélération et de début de décélération pour chacun des axes.
- module "Changeur de Coordonnées Inverse n1", qui calcule

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

la position et la vitesse finale de chaque axe.
- modules "Génération de Trajectoire", qui sont chargés d'appliquer le profil de déplacement défini pour chaque axe.
- modules "Traitement Vitesse-Position", qui permettent de connaître les conditions initiales sur chacun des axes.

Paramètres du robot utilisés:

N: nombre d'axes du manipulateur.

Principe de l'algorithme:

- 1- Appel du module "changement de Coordonnées Inverse n1" avec comme paramètres:
la position finale PF
le torseur de vitesse final TVPF
le message de retour contient un N uplet:
(θ_i) $i=1,N$ qui donne la position finale de chaque axe.
- 2- Détermination des conditions initiales par lecture sur les modules "Traitement Vitesse-Position" des N doublets ($\theta_i, \dot{\theta}_i$) pour $i=1,N$.
- 3- Calcul du temps de parcours théorique par activation, pour chaque axe, du module "Calcul du Temps de Parcours" en donnant en paramètres deux doublets:
($\theta_i, \dot{\theta}_i$): la position et la vitesse courante
($\theta_{iF}, \dot{\theta}_{iF}$) la position et la vitesse finale.
En retour, on obtient les temps T_i mis pour atteindre chaque position finale θ_{iF} . On calcule le temps $T = \text{MAX}(T_i) * C$, où C est le coefficient de vitesse.
- 4- Appel du module "Calcul Profil de Déplacement" pour chaque degré de liberté, avec les paramètres d'appel suivants:
($\theta_i, \dot{\theta}_i$): la position et la vitesse courante
($\theta_{iF}, \dot{\theta}_{iF}$): la position et la vitesse finale
T: le temps de parcours
C: le coefficient de vitesse.
Les paramètres de retour permettent de définir complètement le profil du déplacement: durée d'accélération, de décélération, temps total de mouvement sur chacun des degrés de liberté.
- 5- Réinitialisation du système: Réinitialisation et désactivation de tous les processus des modules

MODULE DEPLACEMENT LIBRE

"Asservissement" et "Traitement Vitesse Position".

6- Activation du mouvement: les paramètres obtenus au pas 4 sont directement envoyés au module "Génération de Trajectoire" qui est activé.

4. MODULE CHANGEUR DE COORDONNEES INVERSE NUMERO 1

Fonction:

Ce module réalise le changement de coordonnées inverse en position et vitesse.

Mode d'activation:

Envoi requête avec attente réponse, avec comme paramètres d'appel:

- une position cartésienne PF du repère CP
- le torseur de vitesse désiré à cette position.

Le contenu de la réponse est:

($\theta_i, \dot{\theta}_i$): positions et vitesses finales de chaque axe.

Connexion avec les autres modules:

Module "Déplacement Libre": seul module pouvant l'utiliser (par une activation de type 2).

Paramètres du robot utilisés:

L'algorithme du changement de coordonnées fait lui-même partie de la base de données, ainsi que les valeurs caractéristiques propres au manipulateur qui lui sont nécessaires.

Principe de l'algorithme:

Les algorithmes employés sont ceux que l'on peut trouver pour les différents types de manipulateurs [Renaud, ...]

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

5. MODULE TRAITEMENT VITESSE POSITION

Fonction:

Renseigner sur l'état d'un axe (position et vitesse).

Mode d'activation:

Fonctionnement autonome périodique sans paramètres d'appel.

Information disponible:

($\theta, \dot{\theta}$): position et vitesse de l'axe considéré.

Connexion avec les autres modules:

Module "asservissement"

Module "déplacement libre"

Module "stop libre"

Module "changement coordonnées direct"

Paramètres du robot utilisés:

- facteurs de conversion unité codeur-unités système,
- facteur constant de recalage par rapport au zéro mathématique

Principe de l'Algorithme:

Ce module fait l'interface avec les systèmes électromécaniques. Il réalise la conversion et le recalage des données fournies par les capteurs de vitesse et position. La période d'activation est de l'ordre de 10000 hz.

MODULE CALCUL TEMPS DE PARCOURS

6. MODULE CALCUL TEMPS DE PARCOURS

Fonction:

Calcule le temps de parcours pour effectuer un déplacement correspondant aux paramètres d'appel: les conditions initiales et finales en position et vitesse et le coefficient de vitesse désiré pour l'axe concerné.

Mode d'activation:

envoi requête avec attente réponse (type 2).

Les paramètres d'appel de la requête sont:

- le doublet ($\theta_i, \dot{\theta}_i$) position et vitesse initiale
- le doublet ($\theta_f, \dot{\theta}_f$) position et vitesse finale
- C: le coefficient de vitesse.

contenu de la réponse:

T: temps de parcours estimé correspondant aux paramètres envoyés.

Connexion avec d'autres modules:

Ce module peut être appelé

- par le module "déplacement libre"
- par le module "déplacement cartésien".

Paramètres du robot utilisés:

Accélération et vitesse maximum de l'axe concerné.

Principe de l'algorithme:

L'algorithme dépend de la loi d'accélération choisie pour les déplacements (accélération sinusoidale, courbe du 3ème degré, ...). Dans le cas d'une loi de vitesse trapézoïdale (accélération constante, vitesse palier, décélération constante) les cas de figure génériques sont donnés (Figure III.3), la longueur de chacun des différents segments pouvant être nulle.

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

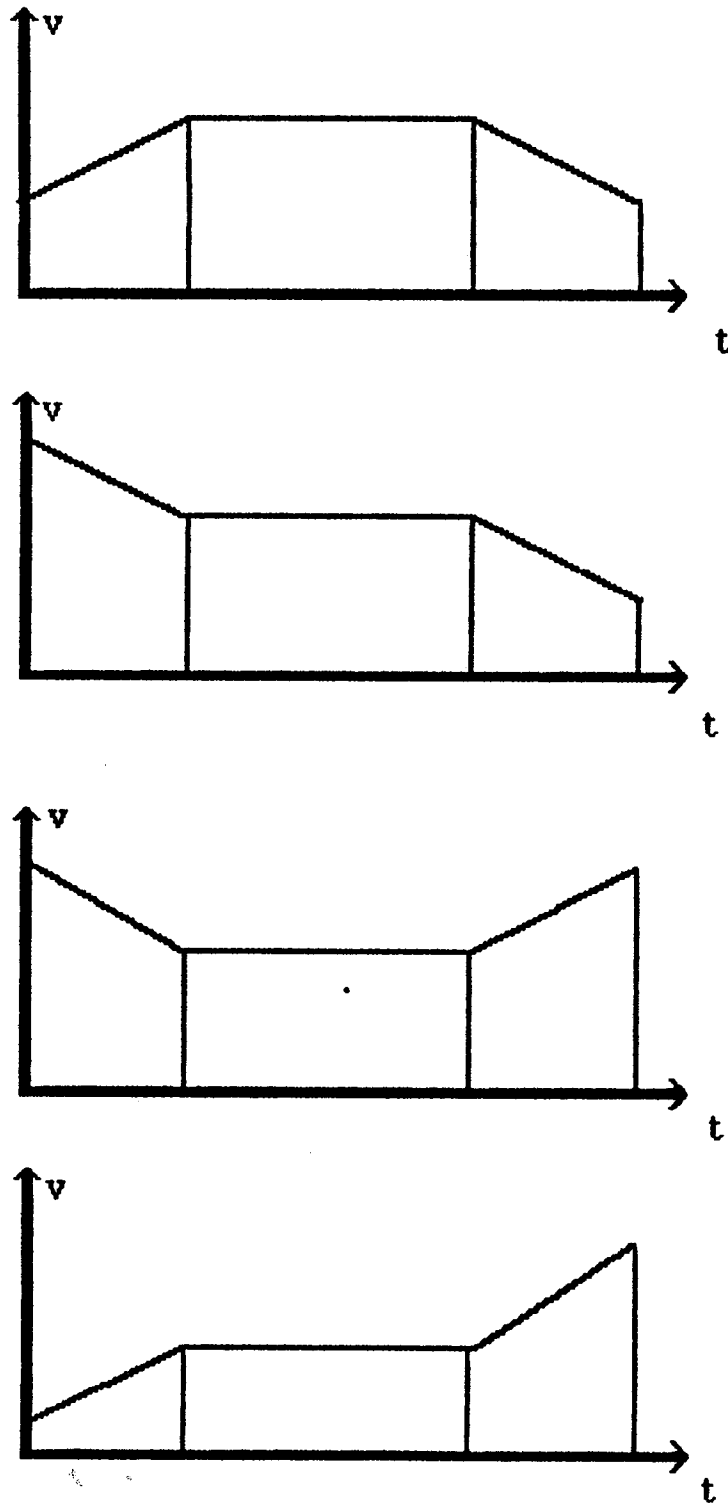


figure III.3

Profils typiques d'une loi de vitesse trapezoidale

MODULE CALCUL TEMPS DE PARCOURS

Le principe de l'algorithme consiste à trouver un profil compatible avec les paramètres d'entrée. La détermination de ce principe donne la vitesse palier, d'où l'on peut déduire le temps de parcours. Dans le cas d'une loi de vitesse trapézoïdale, on choisit d'abord à l'aide du coefficient de vitesse la valeur absolue de l'accélération. Dans chacun des profils, des équations du second degré permettent de calculer la vitesse palier, dont on déduit le temps de parcours.

7. MODULE CALCUL PROFIL DE DEPLACEMENT

Fonction:

Calculer la durée d'accélération, de vitesse palier et de décélération de l'axe concerné, connaissant le temps de parcours total, le coefficient de vitesse, et les positions et vitesses initiales et finales.

Mode d'activation:

envoi requête avec attente réponse (type 2).

Les paramètres d'appel de la requête sont:

- le doublet ($\theta_i, \theta_i.$): position et vitesse initiale
- le doublet ($\theta_{iF}, \theta_{iF}.$): position et vitesse finale
- T: temps de parcours
- C: coefficient de vitesse.

Contenu de la réponse:

- T_{Ai}: durée d'accélération
- A_i: valeur d'accélération
- T_{Di}: durée de décélération
- D_i: valeur de décélération.

Connexion avec d'autres modules:

Ce module peut être appelé

- par le module "déplacement libre"
- par le module "déplacement cartésien".

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

Paramètres du robot utilisés:

Accélération et vitesse maximum de l'axe concerné.

Principe de l'algorithme:

Cet algorithme est le dual de celui utilisé dans le module "calcul temps de parcours". Il dépend donc également de la loi d'accélération choisie. Dans le cas d'une loi de vitesse trapézoïdale, on détermine les profils permettant de réaliser les consignes dans le temps demandé. On en choisit une qui ne provoque pas d'oscillation (si c'est possible). On peut ensuite en déduire les durées et valeurs d'accélération et de décélération.

8. MODULE GENERATION DE TRAJECTOIRE

Fonction:

Ce module réalise l'échantillonnage de la trajectoire par interpolation linéaire dans l'espace des axes du robot (cas du déplacement libre), ou dans l'espace cartésien (cas du déplacement cartésien). Lors de son activation, il reçoit les paramètres de temps (accélération, décélération et vitesse palier) et les valeurs d'accélération, de vitesse et de position correspondantes qui sont les consignes qu'il devra respecter. Ce module se réveille ensuite périodiquement tous les Δt (noté Dt), Dt étant l'intervalle de temps d'échantillonnage entre deux points consécutifs (modifiable dynamiquement, mais uniquement entre deux activations du module de génération de trajectoire).

Mode d'activation:

Périodique après requête (période fonction du temps d'échantillonnage Dt)

Les paramètres d'appel de la requête sont:

- T: temps de parcours
- T_{Ai} : durée d'accélération
- A_i : valeur d'accélération

MODULE GENERATION DE TRAJECTOIRE

- TDi: durée de décélération
- Di: valeur de décélération.

Connexion avec les autres modules:

- l'activation de ce module peut se faire:
par le module "déplacement libre"
par le module "déplacement cartésien".
- la scrutation de l'état courant de ses paramètres de sortie peut se faire par le module "déplacement VIA".
- ce module active le module "anticipation dynamique" à chaque pas d'échantillonnage dans le cas d'un mode libre, ou le module de formattage de la trajectoire cartésienne, dans le cas d'un mode cartésien.

Paramètres de la base de données utilisés.

Dt: temps d'échantillonnage.

Principe de l'algorithme:

On échantillonne le profil défini par les paramètres d'entrée avec la période Dt, ce qui permet de calculer la position et la vitesse désirée de l'axe concerné.

9. MODULE ANTICIPATION DYNAMIQUE

Fonction:

Ce module calcule les forces généralisées devant être appliquées aux degrés de liberté pour réaliser la prochaine consigne en vitesse et position. Il doit permettre d'améliorer les performances du manipulateur à grande vitesse.

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

Mode d'activation:

par requête (type 1).

Les paramètres d'appel de la requête sont:

- $(\theta_i, \dot{\theta}_i)$ $i=1, N$: consignes en position et en vitesse de chacun des axes.

Connexion avec les autres modules:

- l'activation de ce module peut se faire par l'ensemble des générateurs de trajectoire par axe ou par le module de transformation de coordonnées inverse n2.

- ce module tient les consignes de position, de vitesse et de couple à la disposition des modules d'asservissement de chacun des axes.

Paramètres du robot utilisés:

L'algorithme d'anticipation dynamique fait lui même partie de la base de données car il dépend du manipulateur. Il utilise des paramètres comme la masse et l'inertie de chacun des degrés de liberté.

Principe de l'algorithme:

Il s'agit de résoudre en fonction des conditions initiales et finales l'équation: $\Gamma = f(\theta, \dot{\theta})$, où Γ est le vecteur représentant l'ensemble des forces généralisées.

Plusieurs formalismes peuvent être utilisés pour résoudre ces équations (formalisme Lagrangien, ...). Ces méthodes ne prennent en général pas en compte l'objet transporté par le robot; elles peuvent introduire certaines simplifications comme la suppression des efforts dûs aux forces de Coriolis.

Une fois l'équation résolue, on envoie les valeurs $(\theta_i, \dot{\theta}_i, I_i)$ pour $i=1, N$ aux modules d'asservissement.

Le calcul des valeurs de couple ajoutées aux consignes de vitesse et de position permet un meilleur asservissement des axes.

MODULE ASSERVISSEMENT

10. MODULE ASSERVISSEMENT

Fonction:

L'asservissement est actif dès que le système est en route et que les moteurs sont asservis. Il y a un module par axe et il a pour charge de réaliser les consignes position-vitesse et couple fournies par le module d'anticipation dynamique.

Mode d'activation:

Activé à chaque mise sous tension du moteur de l'axe correspondant.

Paramètres de la base de données du robot utilisés:

Dt: temps d'échantillonnage
paramètres propres au matériel.

Connexion avec les autres modules:

Seul le module "anticipation dynamique" peut lui fournir les paramètres à prendre en compte pour réaliser l'asservissement.

Seul l'interpréteur LM peut l'activer et le désactiver.

Principe de l'algorithme:

Nous nous plaçons dans le cas d'un asservissement programmé; il effectue la boucle d'asservissement avec une fréquence F de l'ordre de 1000hz ([Luh 83]), et il scrute tous les Dt les nouvelles consignes données par le module d'anticipation dynamique. En cas d'absence de nouvelle consigne, il tâche de réaliser un maintien en position à vitesse nulle à la dernière consigne de position. Lorsque ce module est activé, il réalise le maintien de l'axe sur la position courante.

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

11. MODULE STOP LIBRE

Fonction:

Ce module a pour charge d'arrêter tous les axes du manipulateur dans l'espace des axes. Il est activé lorsqu'il faut arrêter le manipulateur alors qu'il est en déplacement en mode libre.

Mode d'activation:

Par requête (type 1) sans paramètres.

Connexion avec les autres modules:

- Interpréteur des commandes LM, qui envoie une requête.
- module traitement vitesse position pour connaître l'état courant (vitesse) de chacun des axes.
- module de génération de trajectoire pour lui donner les consignes permettant l'arrêt.
- module de déplacement VIA, pour le désactiver.

Paramètres du robot utilisés:

Accélération et vitesse maximum de chacun des axes.

Principe de l'algorithme:

1. on demande la vitesse courante V_{Ci} de chacun des axes aux modules de traitement Vitesse Position.
2. on calcule le temps maximum de décélération T et on en déduit la valeur de décélération pour chaque axe D_i .
3. on envoie la requête suivante aux générateurs de trajectoire:
 - T: temps de parcours
 - T_{Ai} : durée d'accélération
 - A_i : valeur d'accélération
 - T_{Di} : durée de décélération
 - D_i : valeur de décélération.
4. on désactive le module de déplacement VIA.

MODULE DEPLACEMENT CARTESIEN

12. MODULE DEPLACEMENT CARTESIEN

Fonction:

Ce module permet de commander tous les mouvements correspondant aux instructions du langage LM (pour les mouvements en mode cartésien).

Il peut être activé alors que le manipulateur est en mouvement, autorisant ainsi un changement de trajectoire ou de vitesse (option "IMMEDIAT"). Son mode d'activation (type 1) permet à l'interpréteur LM de reprendre le contrôle sans attendre la fin du mouvement (option "SANS ATTENTE").

Mode d'activation:

par requête (type 1).

Les paramètres d'appel de la requête sont:

- la position finale RPF du repère à déplacer.
- la transformation PRP permettant de calculer la position relative du repère à déplacer par rapport au repère CP.
- le torseur de vitesse TVRPF associé à la vitesse finale en fin de déplacement du repère à déplacer (non nul dans le cas d'un déplacement VIA).
- Le coefficient C de vitesse du manipulateur.

Connexion avec les autres modules:

- modules "calcul temps de parcours"
- modules "profil de déplacement"
- modules "génération de trajectoire"
- module de formattage chargé d'exprimer la position du repère CP en fonction des résultats des modules de génération de trajectoire
- module de changement de coordonnées direct.

Principe de l'algorithme:

1. appel du changeur de coordonnées direct qui fournit en retour:

- CP: la position du repère CP
- TVCP: le torseur de vitesse associé au centre du repère CP.

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

2. calcul de la position et du torseur de vitesse initial du repère à déplacer. Ces résultats doivent être exprimés dans le référentiel du repère à déplacer.
3. calcul des paramètres géométriques définissant les deux axes virtuels VTD et VRD permettant de terminer le mouvement en cours; on calcule aussi le repère RI définissant la position théorique d'arrêt du mouvement en cours.
4. calcul des profils de freinage sur les deux axes virtuels VTD et VRD.
5. Calcul des paramètres géométriques définissant les deux axes virtuels VTA et VRA permettant de réaliser le mouvement demandé.
6. Calcul du temps de parcours maximum pour le mouvement sur chacun des deux axes virtuels.
7. Calcul des profils sur les deux axes virtuels pour réaliser le mouvement en fonction du temps de parcours de l'axe virtuel le plus long.
8. Appel du formateur de trajectoires: le message transmis contient les valeurs suivantes: PRP, RI, VTD, VRD, VTA et VRA.
9. Lancement du mouvement: les paramètres définissant les profils des mouvements des degrés de liberté virtuels sont envoyés aux quatre modules "génération de trajectoire" correspondants.

MODULE FORMATEUR DE TRAJECTOIRE

13. MODULE FORMATEUR DE TRAJECTOIRE

Fonction:

Ce module a pour fonction de fournir au transformateur de coordonnées inverse numéro 2 la position et le torseur de vitesse du repère CP de façon à réaliser le mouvement échantillonné selon chacun des quatre degrés de liberté virtuels.

Mode d'activation:

Première activation (initialisation): par requête

Activations suivantes: par requête.

Paramètres de la requête:

Lors de la première activation les paramètres d'appel sont les paramètres géométriques définissant les quatre degrés de liberté virtuels: PRP, RI, VTD, VRD, VTA et VRA.

Lors des activations suivantes, les paramètres d'appel sont les quatres doublets:

- $(\theta_i, \dot{\theta}_i)$ pour $i=1,4$

correspondant aux positions et vitesses de chacun des degrés de liberté virtuels.

Connexion avec les autres modules:

- Les modules "génération de trajectoire", qui activent périodiquement ce module de façon à calculer la prochaine position du repère CP.

- Le module "déplacement cartésien", qui initialise les paramètres géométriques nécessaires.

- Le module "changeur de coordonnées inverse n2" qui reçoit la prochaine consigne à exécuter dans l'espace cartésien.

Principe de l'algorithme:

- Requête d'initialisation:

on range les paramètres fournis en mémoire.

- Requêtes suivantes:

1. Les données scalaires fournies par les générateurs de

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

- trajectoire sont interprétées selon les quatre axes virtuels pour calculer le prochain point du mouvement.
2. La transformation PRP est appliquée pour calculer la position et la vitesse du repère CP.
 3. Ces informations de position et vitesse sont transmises au changeur de coordonnées inverse numéro 2.

14. MODULE CHANGEUR DE COORDONNEES INVERSE NUMERO 2

Fonction:

Ce module a la même fonction que le module "changeur de coordonnées inverse numéro 1" (cf. paragraphe 4), à ceci près que la requête ainsi que les connexions avec les autres modules sont différents:

- le mode d'activation est de type 1 (sans réponse)
- les connexions avec les autres modules sont:
 - le module "formateur de trajectoire" qui lui fournit les positions et vitesses cartésiennes qu'il doit transformer
 - le module "anticipation dynamique" auquel il envoie les informations de position et de vitesse sur chaque axe.

15. MODULE CHANGEUR DE COORDONNEES DIRECT

Fonction:

Ce module fournit, lorsqu'il est appelé, la position et la vitesse cartésiennes instantanées du repère CP.

Mode d'activation:

Envoi requête avec attente de réponse (type 2) sans paramètres d'appel.

Contenu de la réponse:

- CP: position du centre du poignet CP
- TVCP: valeur du torseur de vitesse du repère CP.

MODULE CHANGEUR DE COORDONNEES DIRECT

Connexion avec les autres modules:

- Ce module lit sur le module "traitement vitesse-position" les N doublets $(\theta_i, \dot{\theta}_i)$.
- Ce module peut être activé soit par le module "générateur de trajectoire", soit par le module "stop cartésien"; dans les deux cas, le mouvement en cours est en mode cartésien.

Ce module sert également à tout moment à retourner la position du manipulateur sur toute demande faite à l'intérieur d'un programme LM. Cette activation a alors lieu directement à partir de l'interpréteur LM.

Paramètres du robot utilisés:

Comme pour les changeurs de coordonnées inverses, on peut dire ici que l'algorithme, et toutes les valeurs nécessaires aux calculs, font partie de la base de données.

Principe de l'algorithme:

Les algorithmes employés sont ceux que l'on peut trouver dans la littérature suivant les types de robot [Renaud 84, ...].

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

16. MODULE STOP CARTESIEN

Fonction:

Ce module se charge de calculer un arrêt du robot quand cet arrêt doit être réalisé dans l'espace cartésien, autrement dit lorsque le manipulateur effectue le mouvement en cours en mode cartésien.

Mode d'activation:

Par requête (type 1) sans paramètre.

Connexion avec les autres modules:

- Les deux modules "génération de trajectoire" pour les axes virtuels correspondant au nouveau segment de trajectoire.
- Le changeur de coordonnées direct pour connaître la vitesse selon ces deux axes virtuels.
- Le module "déplacement VIA" pour le désactiver.

Paramètres du robot utilisés:

Décélération maximum cartésienne en translation et en rotation.

Principe de l'algorithme:

Comme dans le cas du "Stop libre", on change les paramètres des générateurs de trajectoire de façon à obtenir une décélération maximum sur chacun des axes virtuels du mouvement courant. Les paramètres géométriques des générateurs de trajectoire restant inchangés, la décélération a lieu le long du segment courant.

Les deux générateurs de trajectoire chargés de décélérer sur le mouvement précédent restent inchangés. Par contre, il est nécessaire de désactiver le module déplacement VIA.

MODULE DEPLACEMENT VIA

17. MODULE DEPLACEMENT VIA

Fonction:

Ce module permet de réaliser les mouvements correspondant aux instructions de type "VIA" du langage LM. Ces instructions décrivent un mouvement comme une suite de segments pouvant être soit cartésiens, soit libres. Son mode d'activation permet à l'interpréteur LM de reprendre le contrôle sans attendre la fin du mouvement.

Mode d'activation:

Périodique après requête.

Les paramètres d'appel de la requête sont une suite de descripteurs de segments de trajectoire; ces descripteurs sont différents suivant le mode de la trajectoire sur le segment, et correspondent aux paramètres d'appel des modules "déplacement libre" ou "déplacement cartésien".

Connexion avec les autres modules:

- Module "déplacement libre" dans le cas où il faut lancer un mouvement libre.
- Module "déplacement cartésien" dans le cas où il faut lancer un mouvement cartésien.
- Module "changeur de coordonnées direct" pour la surveillance de l'approche d'un point intermédiaire.

Paramètres du robot utilisés:

Les rayons des sphères définissant le voisinage d'un point intermédiaire (en translation et en rotation).

Principe de l'algorithme:

"les doigts dans le nez, à fond les gamelles"

CHAPITRE 3: ARCHITECTURE DE LA GENERATION DES MOUVEMENTS

Les descripteurs des segments sont rangés dans une pile FIFO (First In, First Out). Chaque élément de la pile est composé de deux parties comportant respectivement la position du point intermédiaire du repère à déplacer et le descripteur du segment correspondant.

A chaque activation, ce module compare la position du point intermédiaire en sommet de pile avec la position courante du repère à déplacer en utilisant d'une part le module "changeur de coordonnées direct" et d'autre part la transformation PRP (qui se ramène à la matrice Identité dans le cas d'un mode libre).

Si la distance est supérieure aux rayons des sphères définissant la proximité, aucune action n'est engagée. Dans le cas contraire, le descripteur du segment est envoyé au module du mode de trajectoire correspondant.

18. MODULE SUIVI TRAJECTOIRE PLANIFIEE

Fonction:

Ce module permet l'interprétation des instructions de déplacement le long d'une trajectoire préenregistrée.

Mode d'activation: Périodique après requête d'appel.

Les paramètres de la requête sont:

- PRP: transformation de passage du repère à déplacer au repère CP.
- ADRTRAS: adresse mémoire de la table des points échantillonnant la trajectoire.

Connexion avec les autres modules:

- Module "changeur de coordonnées inverse numéro 2".

Principe de l'algorithme:

MODULE SUIVI TRAJECTOIRE PLANIFIEE

La trajectoire planifiée contient des informations de position et vitesse cartésienne correspondant aux pas d'échantillonnage. Ces informations sont toujours relatives à la position précédente. Le premier point est relatif à la position initiale du repère à déplacer.

L'algorithme consiste à transformer cette spécification pour le repère CP et à envoyer le résultat au module de changement de coordonnées inverse n2.

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

1. INTRODUCTION

Le langage LM a donné lieu à plusieurs implantations sur différentes configurations matérielles. Ces implantations ont été réalisées depuis 4 ans soit par le Laboratoire LIFIA, soit par la Société ITMI, soit par d'autres laboratoires sous la supervision du LIFIA ou d'ITMI.

Toutes les implantations réalisées ne sont pas identiques. Certaines différences proviennent de limitations matérielles, soit au niveau de l'ordinateur de commande (par exemple: insuffisance de la place mémoire, processeur trop lent), soit au niveau de la commande asservie mise en place (par exemple: impossibilité de coordonner plusieurs axes le long d'une trajectoire échantillonnée), soit au niveau de la mécanique/motorisation (par exemple: insuffisance des degrés de liberté, axes "tout ou rien").

L'un des intérêts de la multiplicité des implantations de LM est la variété des utilisateurs actuels du langage; ainsi, les critiques de certains de ces utilisateurs sont à l'origine de nombreuses améliorations: par exemple, introduction des modes de fonctionnement (paragraphe 4.3. chapitre II), extensions en cours du langage (paragraphe 9 chapitre I), environnement de programmation (chapitre V), etc... Un autre intérêt a été de nous pousser à définir une structure d'interpréteur et des interfaces (chapters II et III) rendant les programmes d'interprétation de LM aussi portables que possible. En particulier, les limitations matérielles évoquées ci-dessus ne permettant pas de donner à toutes les implantations LM exactement les mêmes fonctionnalités, nous avons dû rendre modulaire l'interpréteur LM en fonction des limitations possibles; le découpage de la tâche de gestion de mouvement de robot décrit au chapitre III est un exemple typique de cette modularisation.

Il est aujourd'hui clair que sans l'expérience de plusieurs implantations, toute tentative a priori de modularisation aurait conduit à un découpage totalement inapproprié. Un troisième intérêt de la multiplicité des implantations a été de nous pousser à mettre en oeuvre des instructions (organisation de la mémoire, séquenceur de tâches,...) légères et performantes relativement aux besoins de

INTRODUCTION

l'interprétation du langage LM.

L'existence d'implantations opérationnelles, même imparfaites, dès la fin de 1980 a rendu possible la mise en oeuvre de manipulations expérimentales d'envergure. Au Laboratoire LIFIA, ces manipulations ont été l'oeuvre de chercheurs, d'étudiants en DEA, et d'élèves-ingénieurs de l'ENSIMAG. Elles ont permis à l'équipe Intelligence Artificielle et Robotique d'acquérir un savoir-faire peu commun dans le domaine de la programmation des robots. Des réflexions méthodologiques sur la façon de programmer un robot ont été à la base de la structuration de l'environnement de LM présenté au chapitre V.

Dans ce chapitre, nous présentons les implantations de LM réalisées (paragraphe 2) et quelques expérimentations effectuées (paragraphe 3). Le paragraphe 4 essaie de tirer de ces développements pratiques quelques leçons pour le futur.

2. IMPLANTATIONS DE LM

Nous donnons ci-dessous une présentation succincte des principales implantations de LM dans les Laboratoires et dans l'industrie. La difficulté d'une telle présentation est qu'une implantation, qu'elle soit de Laboratoire ou industrielle, évolue dans le temps; ainsi on n'a pas souvent une réalisation unique, mais plusieurs, qui se suivent. Nous avons donc organisé la présentation par "site", un site étant soit un laboratoire, soit un industriel disposant d'une implantation originale.

2.1. Laboratoires

a) LIFIA (1979...)

La première implantation de LM a été réalisée en 1980 sur un processeur LSI 11-02 sous le système d'exploitation RT11-FB. Le matériel de robotique était composé d'un manipulateur Robitron à 4 degrés de liberté, muni d'un capteur de force (poignet), d'un manipulateur Barras-Provence à 2 degrés de liberté, et d'un système de vision bidimensionnelle [Mazer 81]. Cette implantation consistait essentiellement en un exécutif de type machine LM, qui préfigurait l'exécutif LM

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

actuel. De plus un analyseur lexical de type assembleur permettait d'écrire les programmes sous forme symbolique en notation post-fixée. Cette implantation a été ensuite complétée et améliorée par nous-mêmes, notamment en introduisant le compilateur et l'éditeur de liens.

b) LAAS-ARA (1981...)

LM a été choisi comme langage de programmation de robots dans le cadre du projet national Automatique et Robotique Avancés (ARA). LM a ainsi été implanté sur le site expérimental d'ARA au laboratoire LAAS à Toulouse. L'implantation est opérationnelle sur un ordinateur GOULD SEL 32 bits et commande un robot à 6 degrés de liberté ACMA TH8 et un robot à 6 degrés de liberté SCEMI. Les besoins de la recherche et la coopération entre plusieurs équipes au sein d'ARA ont fait de cette implantation une réalisation assez particulière. Ainsi l'ensemble de l'interpréteur LM pour piloter le robot ACMA TH8 se trouve sur le calculateur SEL, alors que l'interpréteur LM pour piloter le robot SCEMI se trouve réparti entre le calculateur SEL et le calculateur LSI 11-23 de l'armoire de commande du robot SCEMI; par ailleurs cette armoire de commande étant elle-même munie d'un interpréteur LM (cf. paragraphe 2.2.a), le robot SCEMI et son armoire peuvent être utilisés de façon séparée.

c) CERT-DERA (1982...)

LM a été implanté sur ordinateur Perkin-Elmer 32 bits pour piloter un robot AKR à 6 degrés de liberté (robot de peinture). L'exécutif a ensuite été réalisé sur un calculateur 68000 chargé par le Perkin-Elmer. Cette implantation a été réalisée par l'équipe du CERT-DERA.

L'implantation de la nouvelle version de LM incluant commandes gardées et blocs parallèles (cf. chapitre I, paragraphe 9) est réalisée au LIFIA sur le système CESAR/CLEOPATRE développé par le CERT/DERA, en relation avec ce laboratoire.

IMPLANTATIONS DE LM

d) INRIA (1982-1983)

L'équipe de Robotique de l'INRIA a également réalisé une implantation de LM sur un ordinateur Perkin-Elmer 32 bits pour piloter un robot à 6 degrés de liberté ACMA V80 muni d'une commande à base de processeurs 68000 développée par l'INRIA.

e) ECAM (1983-1984)

Des étudiants de l'ECAM (Ecole Catholique des Arts et Métiers de Lyon) ont implanté une version LM opérationnelle sous RT11 (calculateur PDP 11-23) sur un robot 5 axes entièrement conçu et réalisé à l'ECAM. Cette implantation, effectuée dans le cadre des projets de fin d'études, a consisté à connecter la partie "asservissements" réalisée par ailleurs par l'ECAM au système LM et à programmer en langage LM un boîtier de programmation et d'apprentissage.

2.2. Industrie

a) SCEMI (1981...)

La première implantation industrielle du langage LM a été réalisée par nous mêmes dans le cadre d'un contrat entre la société SCEMI et le Laboratoire LIFIA. Le robot concerné étant le robot à 6 degrés de liberté de la SCEMI, alors en cours de développement. Le calculateur était un DEC LSI 11-23 (avec processeur flottant) sous le système RT11-FB. Le calculateur était connecté à une commande d'axes réalisée par l'équipe de la SCEMI ("Q bus"). Cette implantation a été présentée à la Biennale de la Machine-Outil à Paris en Juin 1982 avec comme application la palétisation-dépalétisation de robots avec montage de roulements à billes présentée en exemple au chapitre I.

Cette implantation de LM a ensuite été adaptée par le département "Programmation des robots" d'ITMI au système d'exploitation RT11-XM qui permet de disposer d'un espace de mémoire adressable plus large. La SCEMI a également transporté cette implantation sur l'armoire de commande de son nouveau robot à 4 degrés de liberté.

Plusieurs dizaines de robots SCEMI munis de langage LM sont aujourd'hui opérationnels dans des Laboratoires et sur des

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

chaines de montage.

b) MATRA-ROBOTRONICS (1982...)

Cette implantation, réalisée par l'équipe "Programmation des Robots" de la Société ITMI, fait partie du produit SYSCOMAT (armoire de commande de robots) développé par la Société MATRA-Robotronics. L'architecture du SYSCOMAT est structurée en trois niveaux: au niveau 1, l'asservissement des axes; au niveau 2, la gestion et la synchronisation des axes; au niveau 3, l'interpréteur LM et les entrées-sorties de haut niveau (connexion avec la vision, base de données CAO, réseaux, ...). L'architecture matérielle est à base de microprocesseurs Intel 8086/8087. Le système d'exploitation était au départ iRMX 86 (un système d'exploitation temps réel multi-tâches), puis a évolué vers un système "maison" plus adapté.

L'armoire SYSCOMAT n'est pas spécifique d'un robot. Elle a été présentée à la Mondiale de la Machine-Outil à Paris en juin 1983, avec une connexion à deux robots SORMEL Cadratic. L'un de ces robots réalisait la découpe laser de tableaux de bord; l'autre réalisait la mise en place des divers boîtiers d'un tableau à partir d'informations fournies par le système VISIOMAT de MATRA-Robotronics. Le VISIOMAT, système de vision, a une architecture et un système d'exploitation similaire au SYSCOMAT, et les applications en vision sont maintenant programmables en langage LV (cf. Chapitre I paragraphe 5, et exemple en annexe 1).

c) ITMI (1983...)

Parallèlement à ses travaux d'implantation de LM pour divers constructeurs et utilisateurs, ITMI a développé sa propre armoire de commande de robots munie de LM (armoire ACOR). Cette armoire utilise un ordinateur Hewlett-Packard série 1000, sur lequel est implanté l'interpréteur LM, et un microprocesseur 68000 pour la commande d'axes. Le ordinateur HP est exploité à l'aide du système temps réel multi-tâches RTE-A. L'interface entre ce ordinateur et le microprocesseur est l'interface RACORD "niveau axes" (cf. chapitre II, paragraphe 4.2.); physiquement il est réalisé par un bus HP-IB aux normes IEEE 488.

L'armoire ACOR n'est pas spécifique d'un robot. Elle a été connectée à un robot SCEMI pour la Société Hewlett-Packard,

IMPLANTATIONS DE LM

l'application visée étants le montage de composants électroniques sur une carte. Elle a également servi à piloter le robot RM de CITROEN-industries; un programme de démonstration en LM a permis le montage d'éléments de moteurs électriques, avec approvisionnement aléatoire sur chaque poste, etc... Bien entendu, cette armoire dispose également de tout le système de vision de la société ITMI.

d) AUTOMATA (1983...)

Le robot AUTOMATA est un robot cartésien à 4 degrés de liberté muni d'un capteur de forces. Il a été conçu et réalisé aux Etats Unis; le système de commande est à base du calculateur PDP 11/23 avec le système d'exploitation RT11-XM de Digital Equipment Corporation. La particularité de ce robot est son rapport coût/performances particulièrement intéressant.

Cette implantation de LM a été effectuée aux Etats Unis et le robot AUTOMATA a été présenté lors de l'exposition mondiale "Robots 8" à Détroit. Des programmes de démonstration réalisaient le montage de circuits sur des cartes électroniques en utilisant le capteur d'efforts, et le dépôt de colle sur un flasque d'unité de disquette avec des trajectoires complexes générées par le module d'échantillonnage de trajectoires hors ligne (cf. chapitre V).

Plusieurs autres implantations industrielles de LM sont en développement, et partiellement opérationnelles (GDA en RFA, DISTRIBEL en Belgique, etc...), sur différents systèmes (Motorola 68000, IBM PC,...).

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

3. EXPERIMENTATIONS

Au sein du Laboratoire LIFIA, le langage LM a été utilisé pour mettre en oeuvre de nombreuses manipulations, notamment dans le cadre de l'assemblage mécanique. Nous en présentons trois ci-après, choisies parmi les travaux expérimentaux les plus récents de l'équipe Intelligence Artificielle et Robotique.

a) Montage d'un amortisseur

Cette application consiste à assembler un amortisseur de Peugeot 505 [Crebassa, Salmon et Schmitt 84]. Beaucoup d'opérations d'assemblage sont des insertions verticales, mais les éléments sont constitués de multiples faces, tailles et matériaux. De plus, les tolérances entre plusieurs pièces sont très fines.

L'assemblage a été réalisé à l'aide du robot SCEMI 6 axes équipé d'un poignet avec capteur d'efforts, d'un système de vision bidimensionnelle noir et blanc, et d'un système de vision tridimensionnelle. Ce dernier utilise un plan laser [Borianne 84]. La figure IV.1 montre quelques étapes de l'assemblage.

Le système de vision 2D sert à déterminer les positions initiales de certaines pièces. Le capteur de forces est utilisé pour des mouvements asservis pour remédier à l'incertitude de positionnement des pièces lors d'insertions délicates. Le capteur 3D rend possible le bon alignement de l'axe du piston avec l'axe du cylindre pour commencer dans de bonnes conditions l'insertion la plus délicate de l'application.

L'utilisation de capteurs a permis une installation de l'environnement du robot assez peu spécifique à l'application. En particulier, les posages sont réalisés à partir de composants modulaires standard. Cependant, le temps passé à définir cet environnement a été largement plus long que prévu, surtout en regard du temps mis à écrire les programmes.

EXPERIMENTATIONS

b) Insertion de composants électroniques

Cette application consiste à insérer des composants non standard comme des résistances et des capacités sur une carte [Caloud et Durand 84]. Le robot SCEMI est équipé du même système de vision 3D que ci-dessus. Des expérimentations ont été effectuées avec des composants ayant des pattes de longueurs différentes. La carte est dans une position connue par rapport au capteur 3D. La figure IV.2 montre plusieurs étapes de l'insertion d'une capacité.

La vision est utilisée pour repérer les pattes dans l'espace. Un premier mouvement asservi aux données du capteur amène l'extrémité de la plus longue tige sur le plan de lumière. Ensuite un mouvement libre donne à LM un deuxième point sur la patte qui est approximée par le programme LM comme ligne droite.

D'autres mouvements libres tournent la patte de façon à la placer verticalement, la translatent au dessus du trou, et réalisent l'insertion. L'insertion de la deuxième patte est effectuée de la même façon. Ceci demande de manoeuvrer le moins possible la première patte.

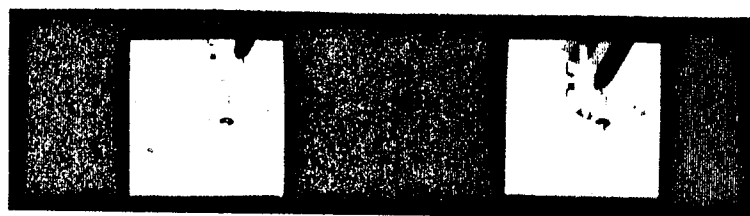
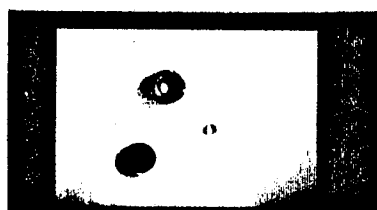
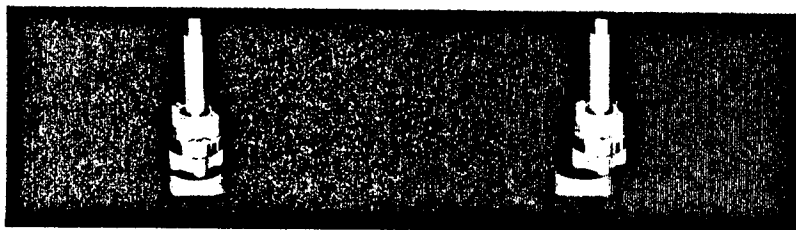
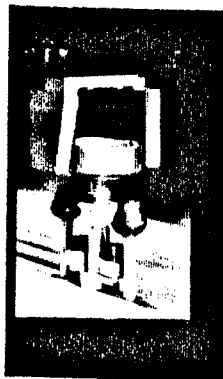
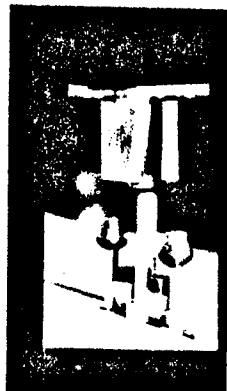
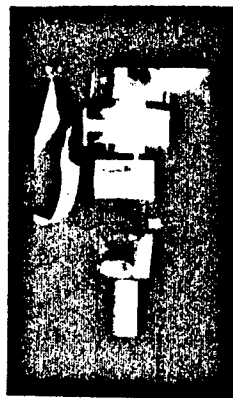
Cette méthode a fourni de bons résultats, lorsque le calibrage avait été effectué avec précautions, en acceptant une grande variation des positions et orientations relatives des deux pattes. Elle est potentiellement plus performante que des méthodes basées sur un retour d'efforts.

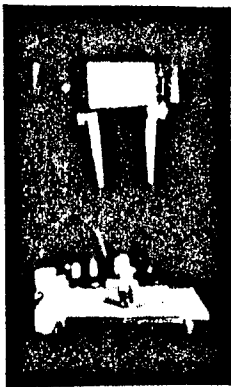
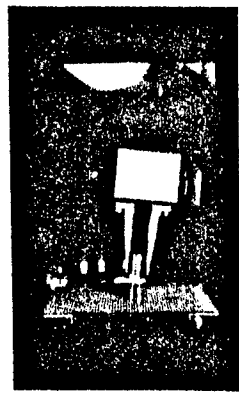
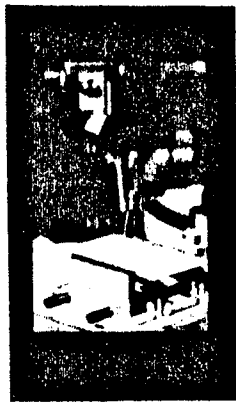
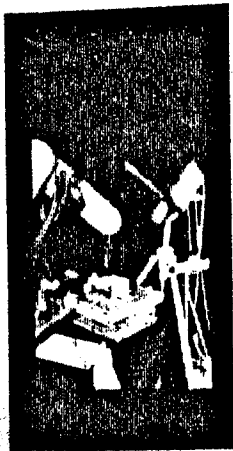
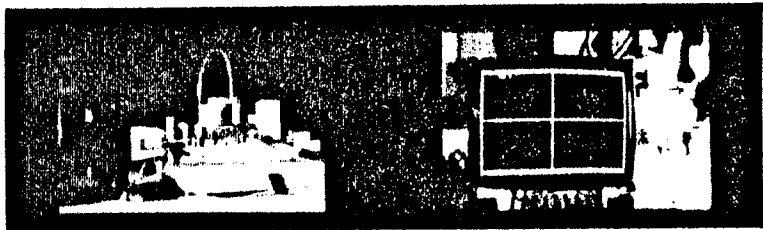
c) Réglage d'un moniteur de terminal alphanumérique

Cette application consiste à régler l'écran vidéo d'un terminal alphanumérique CII.HB [Mondot 84]. Elle nécessite d'analyser des images de référence (mires) et d'exécuter des actions correctrices telles que tourner des potentiomètres, et positionner des aimants. Le terminal est équipé d'un processeur qui génère les images de référence.

Une grande partie de l'application a été automatisée en utilisant un robot SCEMI et un système de vision avec le logiciel CAIMAN. La figure IV.3 illustre les opérations exécutées par l'ensemble du système.

CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS





CHAPITRE 4: IMPLANTATIONS ET EXPERIMENTATIONS

Le robot est utilisé d'une part pour actionner les touches de fonction commandant la visualisation des images de test et d'autre part pour réaliser les actions correctrices. Le programme LM ne fait pas que contrôler ces opérations; il déduit les actions correctrices à partir des défauts repérés par le système de vision (par exemple: l'image est aplatie par rapport à l'axe de l'écran).

Cette application met en évidence la nécessité d'algorithmes et de possibilités de calcul minimum dans un langage de programmation de robots. Bien que le système complet soit relativement lent (ceci est dû en particulier aux échanges inter-calculateurs), il montre la faisabilité de l'automatisation de tâches avec un robot.

4. LECONS

Pendant ces trois dernières années, le système LM (ou une partie de ce système) a été expérimenté de façon intensive:

- comme système autonome, pour développer des programmes de manipulation tels que ceux présentés au paragraphe précédent, soit par le LIFIA, soit par des ingénieurs de plusieurs sociétés.

- comme langage cible des recherches de l'équipe sur la programmation automatique: saisie automatique, mouvements fins, et génération de trajectoires sûres.

Plusieurs conclusions d'ordre général sont ressorties de ces expérimentations:

- 1) La programmation de robots est un véritable besoin.

En fait, la difficulté pour programmer des robots accroit bien plus vite que la complexité apparente des applications. Alors que l'assemblage de jouets simples ne demande que quelques heures ou moins pour être programmé, plusieurs semaines peuvent être nécessaires pour un assemblage comme celui d'un amortisseur.

Bien que la programmation automatique ait fait de gros progrès récemment [Latombe 83], un système industriel complet est encore loin. Cependant des outils interactifs basés sur ces techniques doivent contribuer grandement à simplifier la programmation des robots.

LECONS

2) La description est un élément important de la programmation des robots.

Programmer des robots ne nécessite pas uniquement de planifier des actions et des tests de capteurs: cela demande également de décrire l'environnement des robots (par exemple les liaisons). De même que les capteurs réduisent le nombre d'ingénieurs nécessaires pour faire fonctionner un robot, la description de l'environnement doit être considérée comme une partie importante du problème complet. Dans cette perspective, l'équipe du LIFIA a initié un travail de recherche sur la description des relations d'assemblage [Ingrand 84].

3) L'automatisation de la programmation doit être réalisée "en ligne".

Une des plus grandes difficultés de la programmation de robots vient de la nécessité de prendre en compte toutes les classes de situations demandant des traitements particuliers. Le nombre de ces classes croît avec la complexité de l'application.

Faire opérer les techniques de programmation automatique "en ligne" donne la possibilité d'identifier les situations dans lesquelles on se trouve, et permet ainsi une simplification de ces techniques. De plus, cela peut réduire également de façon significative le nombre d'ingénieurs nécessaire pour faire fonctionner des robots, car le nombre de situations différentes pourrait croître p arbitrairement. Les techniques d'apprentissage inductif [Dufay et Latombe 83c] seraient alors très efficaces pour augmenter l'efficacité de tout le système en limitant les appels à des procédures de programmation automatique.

Ces leçons tirées des expérimentations concernent en fait la programmation automatique, et pas le système LM en lui-même. Cela tient au fait qu'il nous semble qu'il y a plus d'enseignements à tirer, à court et moyen terme, des recherches sur la programmation automatique plutôt qu'à redéfinir un système de niveau manipulation.

CHAPITRE 5: ENVIRONNEMENT DE LM

CHAPITRE 5: ENVIRONNEMENT DE LM

1. INTRODUCTION

Les chapitres précédents concernaient le langage LM et son interpréteur. Autour de ce langage, il a été construit un environnement interactif constitué d'outils d'aide au développement et à la mise au point de programmes en LM. C'est cet environnement que nous présentons dans ce chapitre.

Les principales fonctions de l'environnement de LM sont: l'interprétation en mode interactif de programmes LM, la programmation par l'exemple, la simulation graphique, la programmation géométrique et la planification hors-ligne de trajectoires. Certaines de ces fonctions, la programmation par l'exemple et la simulation graphique, correspondent à des approches de la programmation des robots différentes de l'approche textuelle (cf. Introduction). L'objectif est de combiner les avantages de ces approches pour faciliter le travail du programmeur.

Il existe d'autres systèmes, certains à l'état opérationnel, d'autres en cours de développement, qui visent aussi la combinaison de plusieurs approches de la programmation des robots. Par exemple, les sociétés ASEA [ASEA] et Cincinatti Milacron [Holt 77] proposent des systèmes de programmation par l'exemple assorties de quelques facilités typiques de la programmation textuelle ("apprentissage" de trajectoires en relatif, utilisation de capteurs simples, ...); la société Dassault Systèmes développe en coopération avec le laboratoire LAM (Montpellier) des outils de programmation graphique étendus de façon analogue pour permettre un certain usage de capteurs.

Nous pensons que la programmation textuelle étant le mode de programmation de robots le plus expressif (au niveau manipulation), il est naturel de l'utiliser comme noyau d'un système complet, et non pas l'inverse comme le proposent les exemples ci-dessus. En effet, tous les modes de programmation peuvent alors fournir de façon interne un code commun, celui du langage de programmation textuelle, même si ce code n'est pas visible à tous les utilisateurs. Il est aussi possible d'écrire certaines parties du système dans

INTRODUCTION

ce langage.

Nous n'avons participé à la construction de l'environnement de LM qu'au niveau de sa conception. Aussi notre présentation n'entre-t-elle pas dans les détails des modules qui le compose. Nous reportons le lecteur aux publications citées pour plus de précision.

2. INTERPRETEUR INTERACTIF

L'interpréteur du langage LM se compose de trois éléments de base (cf. chapitre II): le compilateur, l'éditeur de liens et l'exécutif. Il offre deux structures de contrôle possibles. L'une correspond au mode compilation: les unités de programme sont compilées et liées avant d'être exécutées (c'est celle que nous avons présentée au chapitre II). L'autre structure de contrôle correspond au mode interactif: en alternant les appels au compilateur et à l'exécutif, elle permet d'exécuter des séquences d'instructions LM à la demande.

Dans le mode interactif, le programmeur peut donner une ou plusieurs instructions soit en les écrivant au terminal, soit en les sélectionnant à partir de fichiers, et demander leur exécution. Il peut aussi sélectionner les séquences qu'il désire pour composer le programme final qui sera sauvegardé dans un fichier.

Il est également possible d'interpréter et d'exécuter un programme existant instruction par instruction, et de visualiser les valeurs des différentes variables du programme à tout moment.

Des développements en cours permettront prochainement de réaliser sous certaines conditions l'exécution inverse de séquences d'instructions afin de revenir à une situation antérieure à la situation courante, non seulement en termes des variables du programme, mais aussi en ce qui concerne l'univers physique (les robots et leur environnement).

Le choix d'utiliser les mêmes modules d'interprétation en mode compilation et en interactif n'a pas eu comme seul objectif de réduire le travail de mise en oeuvre du mode interactif. Il correspond aussi au souci de donner au mode

CHAPITRE 5: ENVIRONNEMENT DE LM

interactif les mêmes performances, pendant l'exécution, que celles du mode compilation. C'est pourquoi, en mode interactif, les instructions à exécuter sont toutes compilées avant que la première ne soit exécutée. Ainsi pendant l'exécution, seul l'exécutif est en fonctionnement avec le même code interne, quel que soit le mode, compilation ou interactif [ITMI 84a].

3. PROGRAMMATION PAR L'EXEMPLE

La programmation par l'exemple est souvent opposée à la programmation textuelle, la première étant considérée comme plus simple, et la seconde comme plus expressive. En fait, ce n'est qu'une demi-vérité; considérons par exemple le programme LM suivant:

```
PROGRAMME UTILITAIRE;
BOOLEEN ENCORE; REPERE POS; FICHER PROG;
DEBUT
ECRIRE "PROGRAMME RESTIT;" DANS PROG;
ECRIRE "DEBUT" DANS PROG;
ENCORE:=VRAI;
CO boucle d'apprentissage;
TANTQUE ENCORE FAIRE
    POS:=ROBOT;
    MANUEL;
    SI DISTANCE(ROBOT,POS)/=0
        OU ANGLE(ROBOT,POS)/=0
    ALORS
        ECRIRE "DEPLACER ROBOT A (",ROBOT,");" DANS PROG;
    SINON
        ENCORE:=FAUX;
    FINSI;
    ECRIRE "FIN;" DANS PROG;
FINFAIRE;
FIN;
```

Ce programme peut être exécuté par un utilisateur ignorant tout de LM, en tant qu'utilitaire d'apprentissage par l'exemple. Il donne de façon interactive le contrôle au palonnier de commande (instruction MANUEL). Quand le contrôle est rendu au programme, la nouvelle position et orientation de l'outil terminal du robot (repère ROBOT) est comparée à la valeur précédente (mémorisée dans le

PROGRAMMATION PAR L EXEMPLE

repère POS). Si l'une des valeurs a changé, la position courante de ROBOT permet d'engendrer l'instruction DEPLACER correspondante dans le fichier prog.

Une simplification de ce programme consisterait à mémoriser uniquement les positions et orientations successives de ROBOT dans un fichier et à exécuter un programme de restitution reprenant ce fichier. Cependant, il est intéressant d'engendrer explicitement un programme textuel parce que ce programme est facilement éditable: il peut par la suite être complété par des instructions de contrôle et des instructions mettant en jeu des capteurs. De plus, une fois compilé, ce programme peut s'avérer plus performant que le programme de restitution.

Cet exemple très simple montre qu'un langage textuel comme LM est un outil puissant pour développer des utilitaires de programmation par l'exemple. Cette propriété peut être exploitée de deux façons:

- pour réaliser des interfaces mises à la disposition d'utilisateurs n'ayant aucune connaissance en informatique: il est relativement facile de créer des programmes d'apprentissage spécialisés et adaptés à des tâches variées telles que le chargement-déchargement de machine, la palétisation et la soudure en point à point.
- pour implémenter des outils de développement de programmes destinés à des programmeurs expérimentés: même pour de tels utilisateurs, des utilitaires d'apprentissage sont d'une grande aide pour décrire l'environnement du robot en utilisant le robot comme instrument de mesure, pour créer des parties simples de programmes, ou pour construire des squelettes de programmes à compléter par la suite.

Cette complémentarité de la programmation textuelle et de la programmation par l'exemple a été également reconnue et exploitée dans quelques autres systèmes, par exemple: POINTY dans le système AL [Grossman and Taylor 78], et XPROBE dans le système AML [Summers and Grossman 84].

Plusieurs utilitaires de programmation par l'exemple ont été implantés en utilisant le langage LM. Beaucoup sont très simples (en fait, beaucoup ont été créés par des étudiants comme exercices de programmation en robotique). L'un d'eux, LM-EX [Bansard 83], qui a été développé pour assister le programmeur pour des tâches d'assemblage

CHAPITRE 5: ENVIRONNEMENT DE LM

complexes, est particulièrement sophistiqué. Il constitue à la fois un environnement de programmation et un environnement d'exécution:

- En tant qu'environnement de programmation, il combine des commandes de programmation et d'édition. Les commandes de programmation comprennent les mouvements du robot par un palonnier et par des fonctions spécialisées données au terminal. Par exemple, les grands mouvements sont facilement commandés à partir d'un palonnier, et les fonctions spécialisées sont utilisées pour les petits mouvements de positionnement précis (déplacements incrémentaux relatifs). Les commandes de programmation incluent aussi un sous-ensemble des facilités offertes par LM, construit autour d'un petit nombre de repères cartésiens prédéfinis, et de registres et accumulateurs pour chaque type de donnée de LM. Ce sous-ensemble donne accès à toutes les facilités de calcul de LM, ainsi qu'à des procédures et fonctions prédéclarées.

Chaque commande de programmation peut être stockée sur fichiers en utilisant les commandes d'édition. Le programmeur obtient sur demande le programme textuel en LM correspondant à ce qu'il a mémorisé dans ses fichiers.

- En tant qu'environnement d'exécution, le système autorise l'exécution d'un programme engendré par LM-EX soit en mode pas à pas, soit en mode automatique. Le programme en cours d'exécution peut être stoppé à tout moment, et son contexte est automatiquement sauvegardé pour pouvoir reprendre l'exécution au même point plus tard. On dispose également de possibilités d'interruption pour modifier le programme en passant directement sous l'environnement de programmation LM-EX.

LM-EX a été utilisé en particulier pour engendrer le programme complet de l'assemblage d'un essuie-glaces. L'expérience acquise avec LM-EX a été très utile pour développer une nouvelle génération de boîtiers de programmation et d'apprentissage à usage général avec touches de fonctions spécialisées intégrées et dynamiquement modifiables. Le logiciel de contrôle de ces boîtiers est entièrement écrit en LM.

SIMULATION GRAPHIQUE

4. SIMULATION GRAPHIQUE

4.1. Caractéristiques principales

Depuis cinq ans, la simulation graphique s'est révélée un outil pour la programmation de robots lorsque l'on dispose de facilités telles que celles offertes par les systèmes de CAO [Meyer 81] [Dooner, Taylor et Bonney 82] [Sata, Kimura et Amano 81], [Arai 83], [Liégois, Borrel et Dombre 84].

Les caractéristiques principales du simulateur LM, appelé ISR [Laugier 84a], sont les suivantes:

- 1- Il permet soit d'exécuter des programmes d'application existants afin de les mettre au point et de juger leurs performances, soit d'utiliser des utilitaires de programmation par l'exemple combinés avec des possibilités de description graphique pour réaliser de nouveaux programmes d'application.
- 2- Il inclut des fonctions pour simuler les incertitudes liées au monde réel, et les entrées d'information venues de capteurs d'environnement.
- 3- Pour donner à l'utilisateur une sensation réaliste de l'univers tridimensionnel, l'affichage graphique peut se faire sur un terminal de synthèse d'images réalistes avec visualisation de faces pleines [Martinez 82].

Le simulateur interagit avec l'utilisateur et avec l'interpréteur LM. Il peut recevoir trois types de commandes:

- Les commandes du programme: Elles sont envoyées par l'exécutif LM en fonction des instructions du programme LM en cours de traitement, par exemple: mouvements du robot, appels de fonctions capteur.
- Les commandes de simulation: Elles sont envoyées soit par l'exécutif, soit par l'opérateur. Elles concernent les paramètres de visualisation, la mise à jour des liaisons entre les objets, le temps de déroulement de l'exécution, l'arrêt de la simulation, etc...

CHAPITRE 5: ENVIRONNEMENT DE LM

- Les commandes de l'opérateur: Elles sont envoyées par l'opérateur. Il s'agit des requêtes graphiques interactives transmises pendant l'exécution d'utilitaires de programmation par apprentissage.

L'interpréteur du langage LM est connecté au simulateur sans modifications, en mode interactif comme en mode compilé, pour contrôler des robots réels et interagir avec le simulateur ISR. Le même programme LM peut être exécuté dans les deux cas. Les commandes du programme sont émises par l'interface RACORD de l'exécutif (cf. chapitre II) sous la même forme que pour commander un robot réel. Les commandes de simulation sont envoyées par l'interface ESVOIE de l'exécutif lors de l'exécution d'instructions LM du type ECRIRE <commande de simulateur> DANS SIMUL; où SIMUL est un canal réservé à la communication avec ISR.

L'opérateur interagit avec le simulateur par l'intermédiaire d'un terminal graphique, de menus, curseurs et touches de fonctions.

4.2. Composants du simulateur ISR

Le simulateur ISR se compose de sept modules:

a- Simulation d'un manipulateur:

Ce module simule l'exécution des mouvements du manipulateur et des actions de son ou ses outils décrits en LM, et demande les mises à jour correspondantes du modèle au module de modélisation (paragraphe b).

b- Modélisation de l'univers:

Ce module est chargé de tenir à jour un modèle géométrique de l'univers. Ce modèle est représenté en interne sous la forme d'un graphe de relations entre objets. Les relations peuvent être soit articulées, soit rigides, ces dernières étant soit permanentes, soit temporaires [Laugier et Pertin Troccaz 84].

c- Interface graphique:

Ce module a pour rôle d'une part de visualiser la scène et d'autre part d'assurer l'interface avec l'opérateur. Les scènes peuvent être représentées en dessin au trait ou en faces pleines. La position d'observation -le point de vue- peut être fixe ou attachée à un objet mobile, le centre de la pince du robot par exemple.

Les commandes graphiques offrent de nombreuses facilités:

SIMULATION GRAPHIQUE

rendre des objets invisibles, superposer des images, déplacer la source lumineuse, réaliser des "zooms" et des "travellings" sur une scène, etc [Laugier, Evieux et Pertin Troccaz 84].

d- Déroulement temporel:

Ce module a pour charge d'orchestrer dans le temps les opérations du simulateur en fonction des coefficients de "dilatation" du temps d'exécution fixés par l'opérateur. La simulation peut ainsi être interrompue (le temps est suspendu), ralentie ou accélérée, la vitesse maximum étant la vitesse de déroulement du programme dans des conditions réelles, régie par l'exécutif LM.

e- Simulation du monde réel:

Ce module simule deux aspects du monde réel: les incertitudes et les collisions. Une fonction apporte des modifications aléatoires sur des positions d'objets dans le modèle, en réponse à des commandes de simulation spécifiant les variations maximales. Une autre fonction sert à détecter les collisions. Ceci est réalisé en activant sur demande des capteurs de toucher virtuels attachés à des objets ou à des parties d'objets ou du manipulateur. Ces capteurs virtuels signalent toute collision sur l'un ou l'autre des objets qu'ils sont en charge de surveiller. Ils peuvent être activés et désactivés à tout moment.

f- Simulation de capteurs:

Ce module a pour fonction de simuler les capteurs d'environnement qui sont ou pourraient être utilisés avec le robot réel. Certaines fonctions de simulation de capteurs sont des fonctions autonomes ne demandant pas d'interaction avec l'opérateur; par exemple un capteur de vision 2D simulé fournit la position de l'objet à reconnaître en fonction du contenu de la base de donnée du modèle de l'univers. D'autres fonctions requièrent une intervention limitée de l'opérateur, comme par exemple une estimation des valeurs d'un torseur d'efforts après détection d'une collision. La simulation de capteurs est simple, mais elle ne traduit pas les temps de réponse de l'interrogation de capteurs réels.

g- Simulation du palonnier de commande manuelle:

Le but de ce module est de permettre à l'opérateur de manoeuvrer le robot simulé de la même façon que s'il utilisait le palonnier de commande du robot (instruction

CHAPITRE 5: ENVIRONNEMENT DE LM

LM "MANUEL"). Il est possible d'utiliser le palonnier réel pour cette commande mais la visualisation bidimensionnelle de la scène sur l'écran rend cette utilisation difficile. C'est pourquoi ISR comprend un module interactif basé sur les concepts développés au Laboratoire LAM [Liegeois, Borrel et Dombre 84]: l'opérateur désigne des points ou des lignes sur l'écran, et demande leur superposition ou leur alignement. Les mêmes utilitaires de programmation par l'exemple sont disponibles que ce soit sur le robot réel ou le robot simulé par ISR.

4.3. Implantation

Le simulateur a été implanté en LISP sur calculateur CII-HB 68, il est également opérationnel sur VAX 750. Il a été connecté au terminal HELIOS-GETRIS de synthèse d'images, développé au Laboratoire ARTEMIS à l'IMAG [Martinez 82].

La figure V.1. montre une séquence d'images extraites d'un film simulant l'exécution d'un programme LM et engendrées à l'aide de ISR.

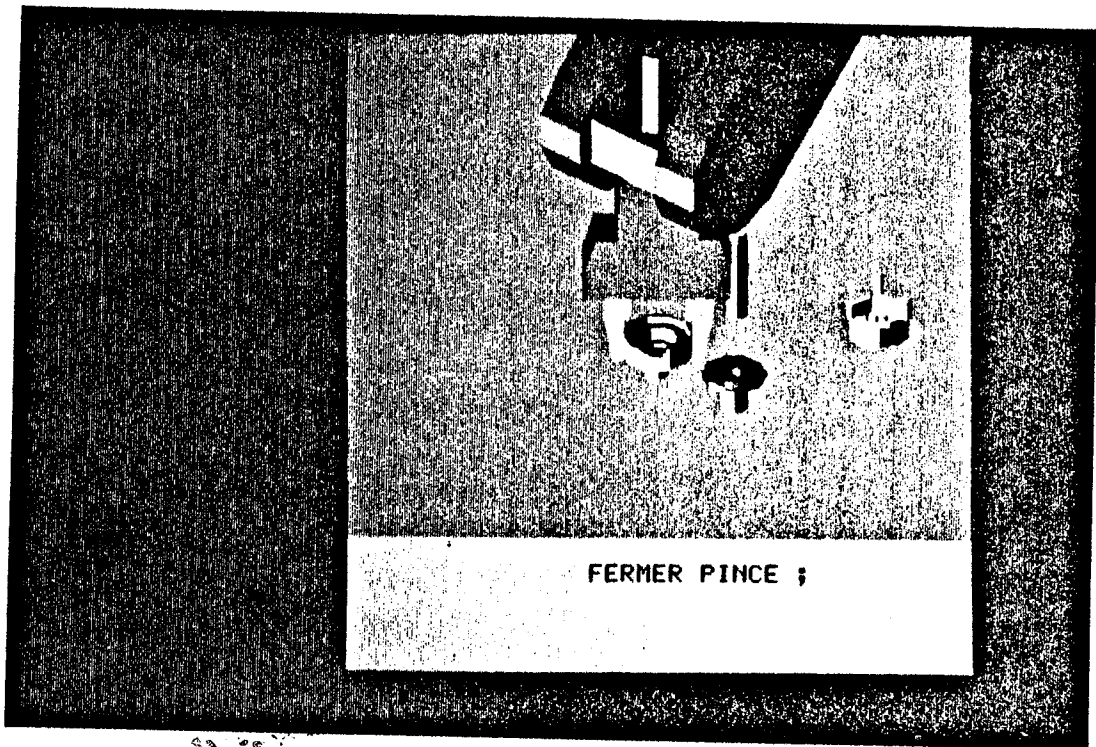
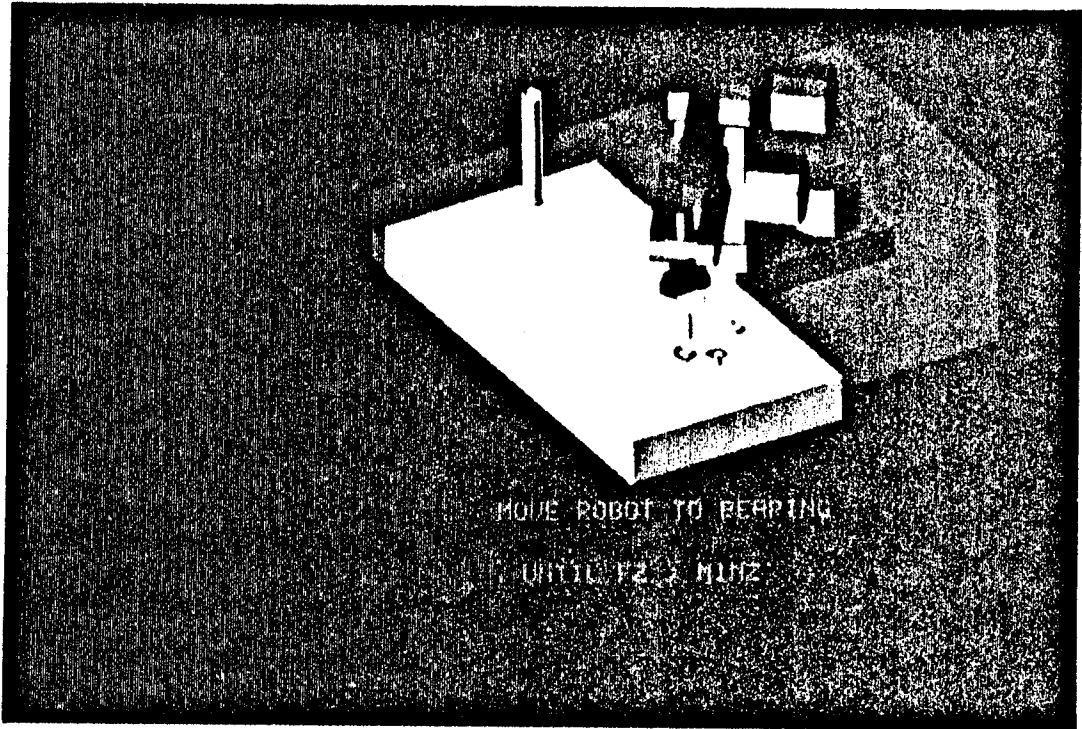
5. PROGRAMMATION GEOMETRIQUE

5.1. Objectif

Définir les positions de repères cartésiens par des transformations combinant translations et rotations est parfois délicat et/ou fastidieux. Cela peut également résulter en des programmes difficiles à comprendre.

La plupart du temps, le programmeur se sert d'utilitaires de programmation par l'exemple, qui facilitent la description des positions, mais n'améliorent pas la lisibilité des programmes.

PROGRAMMATION GEOMETRIQUE



CHAPITRE 5: ENVIRONNEMENT DE LM

Une extension du langage LM, appelée LM-GEO, a été réalisée pour rendre possible la définition de positions d'objets par des relations géométriques explicites sur ces objets [Mazer 83]. LM-GEO vise à la fois à simplifier l'écriture de programmes et à accroître leur lisibilité.

Les concepts de base de LM-GEO sont inspirés des travaux sur le langage RAPT [Poplestone, Ambler et Bellos 78], et du langage LRG [Miribel 81].

5.2. Le langage LM-GEO

LM-GEO est une extension de LM: toutes les instructions de LM sont donc utilisables en LM-GEO. Toutefois le principal intérêt de LM-GEO est d'éviter l'utilisation explicite des repères cartésiens et des transformations, grâce à une nouvelle construction appelée "situation géométrique".

Par exemple, une séquence d'instructions LM pour placer un robot en position de prise d'un objet peut être:

```
t:=TRANSLAT(VECT(-1,0,-1),12.)*ROT(VZ,PI);
```

```
DEPLACER ROBOT A boite *t;
```

En LM-GEO le même objectif peut être atteint par une seule instruction, par exemple:

```
DEPLACER ROBOT POUR REALISER prise-boite;
```

où "prise-boite" est une situation géométrique.

Une situation géométrique est définie par des relations symboliques entre des entités géométriques des objets. Par exemple, "prise-boite" pourrait être définie par:

```
haut DE pince EST-CONTRE(50) r1 DE boite;  
mors_1 DE pince EST-CONTRE(0) r2 DE boite;  
mors_2 DE pince EST-CONTRE(0) r3 DE boite;  
face_1 DE pince EST-PARALLELE-A(25) r4 DE boite;
```

Dans cette définition, pince et boite sont des noms donnés à deux objets; haut, mors_1, mors_2, et face_1 désignent des faces de l'objet pince.

PROGRAMMATION GEOMETRIQUE

Les relations pour décrire des situations géométriques sont actuellement les suivantes:

<plan> EST-CONTRE (d)<plan>, où d est une distance entre deux plans
<plan> EST-PARALLELE-A (d)<plan>,
<ligne> EST-ALIGNE-AVEC <ligne>,
<ligne> EST-PARALLELE-A <ligne>,
<point> EST-SUR <point>

Chaque situation géométrique définit la position d'un objet par rapport à un autre objet dont la position sera connue au moment de l'utilisation de la situation par le programme. Chaque objet d'une situation est soit un objet élémentaire, soit un sous-assemblage défini par une situation.

Chaque objet élémentaire doit être décrit dans la base des modèles. Il faut lui attacher un repère cartésien (qui sera utilisé par l'interpréteur de LM-GEO), et donner un nom à toutes les entités géométriques (faces, arêtes, sommets) qui sont utilisées dans les situations. Les équations ou coordonnées de ces entités sont définies relativement au repère cartésien attaché à l'objet.

La position initiale d'un objet peut être définie par une situation géométrique, ou être obtenue en cours d'exécution (par exemple par un système de vision).

CHAPITRE 5: ENVIRONNEMENT DE LM

5.3. L'interpréteur LM-GEO

L'interpréteur LM-GEO est principalement un programme de réécriture générant du code LM. Il opère en trois étapes:

La première étape consiste à calculer les transformations entre les deux repères attachés aux objets figurant dans chaque situation géométrique. Les relations d'une situation sont converties en équations vectorielles, elles mêmes transformées en un ensemble d'équations linéaires et une équation non linéaire, qui est alors résolue par des méthodes conventionnelles. A l'inverse de la méthode basée sur le calcul symbolique utilisée en RAPT, cette méthode analytique trouve toujours un résultat, s'il existe et s'il est unique; dans le cas contraire, LM-GEO retourne un message: la situation n'est pas suffisamment explicite, ou des relations sont incompatibles [Mazer 82b].

La deuxième étape consiste à engendrer du code LM pour déclarer un tableau de repères, OBJET(i), pour tous les objets référencés dans le programme LM-GEO, et un tableau de situations, SITUATION(j), pour toutes les situations géométriques. Les valeurs des transformations calculées lors de la première étape sont affectées aux éléments correspondants du tableau SITUATION.

La troisième étape consiste à traduire les constructions de LM-GEO en constructions LM. Par exemple:

DEPLACER prisme-1 POUR REALISER approche-prisme-2;
pourrait être traduit en:
DEPLACER objet (5) A objet (6) * Situation (4);

L'interpréteur LM-GEO est écrit en langage Lisp, et a été expérimenté sur des tâches d'assemblage simples [Mazer 83].

6. PLANIFICATION DE MOUVEMENTS

La tâche de génération de mouvements de l'exécutif LM (cf. chapitre III) planifie et exécute en temps réel des déplacements suivant des séquences en mode libre, en ligne droite ou en mode circulaire. Dans les deuxième et troisième modes, la vitesse sur chaque segment est

PLANIFICATION DE MOUVEMENT

constante, sauf durant les transitions entre segments et durant les phases de démarrage et d'arrêt du mouvement.

Il existe plusieurs types d'applications qui requièrent un contrôle précis de position et vitesse suivant des trajectoires complexes, par exemple le suivi de joint de soudure et le dépôt de colle. Le planificateur de mouvements du système LM est un outil développé pour répondre à ce genre d'applications. Il opère hors-ligne.

Un mouvement est décrit au planificateur sous la forme d'un ensemble d'équations symboliques spécifiant la géométrie de la trajectoire et la vitesse le long de cette trajectoire. La sortie est une suite de positions, affectées chacune d'une vitesse, distribuée de façon uniforme sur l'axe des temps suivant une période d'échantillonnage donnée. Cette table est mémorisée sur fichier, et peut être utilisée par des instructions LM de la forme:

DEPLACER <rep> SUIVANT <fichier-trajectoire>;

Durant l'exécution du mouvement, l'exécutif LM transforme les positions et vitesses de la table en appliquant la transformation qui amène la première position dans la table sur la position initiale du repère à déplacer. Cette transformation permet de définir des trajectoires relatives. Le planificateur ne contrôle donc que les limites d'accélération et de vitesse imposées dans l'espace cartésien et non dans l'espace des axes du manipulateur. Le contrôle des limites dans ce dernier espace est à la charge de l'exécutif LM lors de l'exécution.

Le planificateur de mouvements a été d'abord développé pour des trajectoires à orientation fixe. Un mouvement est alors spécifié par quatre équations symboliques définissant la trajectoire du centre du repère à déplacer et la vitesse le long de ce segment:

$$X=X(c) \quad Y=Y(c) \quad Z=Z(c) \quad V=V(c)$$

où c est un paramètre permettant de relier les quatre équations. Une telle description permet de spécifier séparément la géométrie et la vitesse du mouvement. Le planificateur calcule automatiquement la relation entre C et le temps.

CHAPITRE 5: ENVIRONNEMENT DE LM

Plus récemment, le planificateur a été complété pour permettre également la description de mouvements mettant en jeu des orientations du repère déplacé. La rotation est spécifiée par trois équations définissant les angles d'Euler en fonction de C. La vitesse de rotation est imposée par la vitesse linéaire.

Le planificateur de mouvements permet à l'utilisateur de décrire un mouvement comme une suite de segments, où chaque segment est défini comme décrit ci-dessus. Le planificateur calcule les transitions nécessaires pour passer d'un segment au segment suivant.

Ce module est implémenté en Fortran. Il a été expérimenté sur plusieurs applications telles que dépôt de colle sur le contour d'un fer à repasser électrique, et dépôt de pâte sur des carters moteur [ITMI 84b].

CONCLUSION

CONCLUSION

La présentation de notre contribution aux développements sur le système LM s'est tout d'abord appuyée sur la mise en évidence des fonctions nécessaires à un langage de programmation de robots (chapitre I). Nous avons ensuite montré la structure générale du système, en insistant tout particulièrement sur les objectifs de portabilité et d'extensibilité que nous nous étions fixés (chapitre II). Le chapitre suivant (chapitre III) présentait l'architecture quasi idéale que nous préconisons pour réaliser les calculs et contrôles des mouvements décrits dans la grammaire du langage. Nous donnions ensuite (chapitre V), la liste et les caractéristiques particulières des diverses implantations de LM réalisées au niveau universitaire comme au niveau industriel. Enfin, le dernier chapitre (chapitre V), a été l'occasion de présenter l'environnement du langage LM, et en particulier les divers outils d'aide à la programmation qui ont été développés autour de LM.

Il est difficile de tirer une conclusion générale portant sur l'avenir de LM. Cependant il nous paraît important de tirer les enseignements de notre travail et des développements nombreux effectués sur le système LM en termes d'objectifs pour la réalisation d'un système de programmation de robots. Nous pouvons en citer trois qui nous paraissent essentiels:

- Il faut un système de programmation puissant pour profiter au mieux du potentiel des robots; la programmation textuelle paraît la meilleure voie car elle combine un pouvoir d'expression élevé et des possibilités englobant le mode "par apprentissage" (apprentissage graphique, boîte à boutons, ...).

- Il est important que le système de programmation tienne compte de l'évolution prochaine vers une intégration de plus en plus poussée de l'environnement: capteurs variés, réseaux de communication, etc...

CONCLUSION

- Il faut également tenir compte du fait que la programmation de robots n'est pas une chose simple et qu'elle comporte des risques matériels. Des outils d'aide au développement et à la mise au point sont essentiels pour aider le programmeur dans sa tâche.

Ce sont ces objectifs qui ont présidé à la réalisation du système LM, et nous pensons qu'ils sont aujourd'hui atteints ou en passe de l'être.

Un dernier objectif constitue le pari qui n'est pas encore gagné (ni perdu!): faire de LM un "standard" universellement répandu. Les conventions d'interface présentées au chapitre II sont un élément important pour la réussite de cet objectif: grâce à elles, LM est certainement le langage de programmation de robots disponible sur le plus grand nombre de matériels, tant manipulateurs que calculateurs (cf.chapitre IV).

Rendez-vous dans trois ans ...

BIBLIOGRAPHIE

Références:

IJCAI = International Joint Conference on Artificial Intelligence
AAAI = American Association for Artificial Intelligence
ISIR = International Symposium on Industrial Robots

[Arai 83]

T.ARAI: "A robot language system with a colour graphic simulator"
Proc. of Advanced software in Robotics, Liège, Mai 1983
(p.215-226)

[Bansard 83]

"Le système LM_EX"
Rapport interne, Laboratoire LIFIA, 1983

[Blanchard 82]

M.BLANCHARD : "Comprendre, maîtriser et appliquer le GRAFCET"
CEPADUES Editions, Toulouse 1982

[Borianne 84]

"Contributions à la vision par ordinateur tridimensionnelle", Thèse de 3ème cycle, INPG, Avril 1984.

[Caloud P. et Durand P. 1984]

"Programmation automatique de robots: application à l'insertion de composants électroniques non standards sur une carte"

[Coïnte 83]

COÏNTE P.: "Comprendre Smalltalk, Penser Smalltalk"
mes Journées BIGRE Octobre 1983

[Crebassa M, Salmon F. et Schmitt P. 1984]

"Assemblage d'un amortisseur par un robot"
Rapport interne, Laboratoire LIFIA, 1984

[Delaunay 80]

M.DELAUNAY, J.F. GRABOWIECKI: "Note technique d'utilisation du Transformateur de Grammaire L1(1)"
Rapport de recherche IMAG 234, Septembre 1980

[Dooner 82]

DOONER M., TAYLOR M.K. and BONNEY M.C.: "Planning robot installations by CAD"
CAD Conference, Brighton 1982

[Dufay, 82b]

B.DUFAY, C.GANDON, J.C.LATOMBE, C.LAUGIER, E.MAZER, J.F.MIRIBEL, J.PERTIN: "Vers la non programmation des robots"
lères Journées ARA, Poitiers, Septembre 1982.

[Dufay, 83a]

BIBLIOGRAPHIE

- B.DUFAY: "Apprentissage par induction en Robotique. Application à la synthèse de programmes de montage"
Thèse de 3ème Cycle, Institut National Polytechnique de Grenoble, Juin 1983.
- [Dufay, 83b]
B.DUFAY, J.C.LATOMBE: "Robot programming by inductive learning"
8th International Joint Conference on Artificial Intelligence, Karlsruhe, Août 1983.
- [Dufay, 83c]
B.DUFAY, J.C.LATOMBE: "An approach to automatic robot programming based on inductive learning"
1st International Symposium of Robotics Research, New-Hampshire, Août-Septembre 1983.
- [Falek 80]
D.FALEK, M.PARENT: "An evolutive language for an intelligent robot"
The Industrial Robot, September 1980.
- [Featherstone 83]
FEATHERSTONE R. The international Journal of Robotics Research vol 2, Number 2 été 1983
- [Finkel 74]
R.A.FINKEL et al.: "AL, a programming system for automation"
Artificial Intelligence Lab., Stanford, Memo AIM-243, November 1974.
- [Finkel 76]
R.A.FINKEL: "Constructing and debugging manipulator programs"
Artificial Intelligence Lab., Stanford, Memo AIM-284, August 1976.
- [Germain 84]
F.GERMAIN: "Génération automatique de trajectoires sans collision"
Rapport de DEA, Laboratoire LIFIA, 1984
- [Gini 79]
G.GINI, M.GINI, R.GINI, D.GIUSE: "MAL: A multi-task system for mechanical assembly"
International Seminar on Programming Methods and Languages for Industrial Robots, INRIA, Juin 1979.
- [Griffiths 69]
M.GRIFFITHS: "Analyse Déterministe et Compilateurs"
These de Doctorat d'Etat, Université de Grenoble, Octobre 1969
- [Grossman 78]
GROSSMAN D.D. and TAYLOR R.H.: "interactive generation of object models with a manipulator"
IEEE Trans. Systems, Man, Cybernetics SMC-8,9,p667,679

BIBLIOGRAPHIE

1982

[Hayward 84]

HAYWARD V. and PALU R.P.: "Introduction to RCCL: a robot control 'C' library"
Proc. IEEE International Conference on Robotics, Atlanta, p293,297, 1984

[Hoare 78]

HOARE C.A.R.: "Communicating Sequential Processes"
Comm. of the ACM Vol 21, No 8, Aout 1978

[Hollerbach 82]

HOLLERBACH j.m.: "A recursive Lagrangian formulation of manipulators dynamics and a comparative study of dynamics formulation complexity Robot Motion"
MIT Memo 1982

[Ingrand 84]

INGRAND F. and LATOMBE J.C.: " Functional reasoning for automatic fixtures design"
CAM-I's 13th Annual Meeting and Technical Conference, Clearwater Beach, Florida 1984

[ITMI 84a]

M.BLIN: "Manuel d'utilisation de l'interpreteur LM interactif"
ITMI S.A., 1984

[ITMI 84b]

J.M.LEFEBVRE, M.PELTIER: "Manuel d'utilisation du générateur de trajectoire"
ITMI S.A., 1984

[Latombe, 79]

J.C.LATOMBE: "Une analyse structurée d'outils de programmation pour la robotique industrielle"
Séminaire International sur les Méthodes et Langages de Programmation des Robots Industriels, IRIA, Juin 1979.

[Latombe, 80a]

J.C.LATOMBE, E.MAZER: "Définition d'un langage de programmation pour la robotique (LM)- Analyse fonctionnelle de l'interpreteur"
Rapport de Recherche IMAG, no. 197, Mars 1980.

[Latombe, 81a]

J.C.LATOMBE: "Langages de programmation pour la commande de robots de manipulation"
3ème Journées ADEPA Scientifiques et Techniques de la Production Automatisée, Toulouse, Juin 1981.

[Latombe, 81b]

J.C.LATOMBE, E.MAZER, J.F.MIRIBEL: "LM: un langage de programmation de haut niveau pour la robotique"
Congrès AFCET Automatique, Nantes, octobre 1981.

[Latombe, 81c]

J.C.LATOMBE, E.MAZER: "LM: a high-level programming

BIBLIOGRAPHIE

- language for controlling manipulators"
11th International Symposium on Industrial Robots,
Tokyo, Octobre 1981.
- [Latombe, 82a]
J.C.LATOMBE: "Equipe 'Intelligence Artificielle et
Robotique' -- Etat d'avancement des recherches au 1er
Octobre 1981"
Rapport de Recherche IMAG, no. 291, Février 1982.
- [Latombe, 82d]
J.C.LATOMBE: "Recherches sur les langages de haut
niveau en robotique d'assemblage"
Journées ARA/NSF, Paris, Juin 1982.
- [Latombe, 82f]
J.C.LATOMBE: "A survey of advanced software for robot
manipulators"
Congrès AICA, Padoue, Octobre 1982.
- [Latombe 84]
J.C.LATOMBE, C.LAUGIER, J.M.LEFEBVRE, E.MAZER,
J.F.MIRIBEL: "Design and Implementation of the LM
Robot Programming System", 2nd ISRR Conférence, Kyoto,
Aout 1984.
- [Laugier, 81a]
C.LAUGIER: "Détermination automatique de prises
d'objets à partir d'indices morphologiques locaux"
Congrès AFCET "Reconnaissance de Formes et Intelligence
Artificielle", Nancy, Septembre 1981.
- [Laugier, 81b]
C.LAUGIER: "A program for automatic grasping of objects
with a robot arm"
11th International Symposium on Industrial Robots,
Tokyo, Octobre 1981.
- [Laugier, 82]
C.LAUGIER: "LISP-3D: Logiciel graphique pour la
manipulation et la visualisation de scènes
tridimensionnelles"
Rapport de Recherche IMAG, no. 328, Septembre 1982.
- [Laugier, 83]
C.LAUGIER, J.PERTIN: "Automatic grasping: a case study
in accessibility analysis"
International Meeting on Advanced Software in Robotics,
Liège, Mai 1983, également Rapport de Recherche IMAG,
Décembre 1982.
- [Laugier 84a]
C.LAUGIER, C.EVIEUX, J.PERTIN-TROCCAZ: "Un système de
simulation graphique de robots incluant une gestion
élémentaire des incidents et des capteurs", Rapport de
recherche IMAG no 421, Mars 1984.
- [Laugier 84b]

BIBLIOGRAPHIE

- C.LAUGIER: "Robot programming using a high-level language and CAD facilities", 1st Robotics Europe Conference on "Industrial Robot Activities and Perspectives", Brussels, juin 1984.
- [Laugier 84c]
C.LAUGIER, J.PERTIN-TROCCAZ: "Graphic simulation as a tool for debugging robot control programs", 1st International Symposium on Design and Synthesis, Tokyo, juillet 1984.
- [Lefebvre 82]
LEFEBVRE J.M: "Nouvelle définition de l'instruction DEPLACER dans le langage LM"
Rapport de DEA, Laboratoire LIFIA, 1982
- [Lieberman 77]
L.I.LIEBERMAN, M.A.WESLEY: "AUTOPASS: An automatic programming system for computer controlled mechanical assembly"
IBM Journal of Research and Development, July 1977.
- [liégeois 84]
LIEGEOIS A., BORREL P. et DOMBRE E.: "Programming, simulating and evaluating robot actions"
Proc. of 2nd International Symposium on Robotics Research, Kyoto, p309-316, 1984
- [Lozano-Perez 76]
T.LOZANO-PEREZ: "The design of a mechanical assembly system"
Artificial Intelligence Lab., MIT, AI-TR-397, December 1976.
- [Lozano-Perez 79]
T.LOZANO-PEREZ, M.A.WESLEY: "An algorithm for planning collision-free paths among polyhedral obstacles"
Communications of the ACM, October 1979.
- [Lozano-Perez 80a]
T.LOZANO-PEREZ: "Spatial planning: a configuration space approach"
Artificial Intelligence Lab., MIT, Memo 605, December 1980.
- [Lozano-Perez 80b]
T.LOZANO-PEREZ: "Automatic planning of manipulator transfer movements"
Artificial Intelligence Lab., MIT, Memo 606, December 1980.
- [Lozano-Perez 82]
LOZANO-PEREZ T.: "Robot programming",
A.I. Memo No 698, MIT, Dec. 1982.
- [Lozano-Perez 84]
LOZANO-PEREZ and BROOCKS R.A. : "An approach to automatic robot programming"

BIBLIOGRAPHIE

in "Solid modelling by computers: from theory to applications", edited by Boyse J.W. and Pickett M.M., Plenum Press, New-York 1984

[LPR 81]

F.SKODA: "Commande de robot-version 5, L.P.R. langage de programmation pour robots"

Régie RENAULT, D.T.A.A., décembre 1981

[Luh 83]

LUH S.Y.S.: "Conventional Controlle Design for Industrial Robots"

IEEE Transaction on System, Man and Cybernetics, Vol SMC-13 No3, Mai-Juin 1983

[Mazer, 81]

E.MAZER: "Réalisation d'un support expérimental de recherche pour le projet robotique PANDORE. Définition et implantation du langage LM"

Thèse de 3ème Cycle, INPG, Grenoble, Janvier 1981.

[Mazer, 82a]

E.MAZER: "LM-Geo: geometric programming of assembly robots"

Rapport de Recherche IMAG, no. 296, Mars 1982.

[Mazer, 82b]

E.MAZER: "An algorithm for computing the relative position between two objects from symbolical specifications"

Rapport de Recherche IMAG, no. 297, Mars 1982.

[Mazer, 83]

E.MAZER: "Geometric programming of assembly robots (LM-GEO)"

International Meeting on Advanced Software in Robotics, Liège, Mai 1983.

[Mazer 84a]

MAZER E. et MIRIBEL J.F. "Le langage LM: Manuel de Référence"

édité par les Editions CEPADUES, Toulouse (118p), 1984

[Mazer 84b]

MAZER E. et MIRIBEL J.F: "LM: le langage de la raison" Paru dans la revue "Micros et Robots", Numero 8, Juin 1984

[Meyer 81]

MEYER J.: "An emulation system for programmable sensory robots"

IBM Journal of Research and Development, Vol.26, No6, 1981

[Mezin, 82]

G.MEZIN: "Réalisation d'un système d'acquisition d'images tridimensionnelles pour la robotique"

Mémoire CNAM, IMAG, Novembre 1982.

BIBLIOGRAPHIE

- [Miribel, 81]
J.F.MIRIBEL: "Un langage pour la représentation de relations géométriques entre objets"
Rapport de Recherche IMAG, no. 251, Mai 1981.
- [Miribel, 82]
J.F.MIRIBEL, E.MAZER: "Manuel de référence LM -- Edition Octobre 1982"
Rapport IMAG (Equipe IA & Robotique), Octobre 1982.
- [Mondod 84]
MONDOT p.: "Réglage automatique de moniteurs télévision"
Rapport de DEA, Laboratoire LIFIA, 1984
- [Mujtaba 79]
S.MUJTABA, R.GOLDMAN: "AL users' manual"
Stanford AI Lab Memo AIM 323, Janvier 79.
- [Paul 77]
R.P.PAUL: "WAVE: A model-based language for manipulator control"
The Industrial Robot, March 1977.
- [Paul 80]
R.P.PAUL et al.: "Advanced industrial robot control systems"
School of Electrical Engineering, Purdue University, TR-EE 80-29, July 1980.
- [Paul 81]
R.P.PAUL: "Robot manipulators: mathematics, programming and control"
The MIT Press, 1981.
- [Pertin-Troccaz 84]
J.PERTIN-TROCCAZ: "SMGR: un système de modélisation géométrique et relationnelle pour la Robotique",
Rapport de Recherche IMAG no 422, juin 1984.
- [Popplestone 78]
R.J.POPPLESTONE. A.P.AMBLER, I.BELLOS: "RAPT: a language for describing assemblies"
The Industrial Robot, September 1978.
- [Renaud 82]
M.RENAUD: "Calcul de la matrice jacobienne nécessaire à la commande coordonnée d'un manipulateur"
Mechanism and Machine Theory
- [Renaud 84]
Modèles des robots manipulateurs, application à leur commande"
CEPADUES Editions, Toulouse Mai 1984
- [Rousseau 83]
R.ROUSSEAU: "Définition et implémentation de système modulaire de programmation pour la productique: LMAC"
Thèse de 3ème Cycle, Université de Franche-Comté, avril

BIBLIOGRAPHIE

- 1983.
- [Salisbury 83]
SALISBURY J.K. Jr: "Interpretation of contact Geometries from Force Measurements"
1st Int. Sym. on Robotics Research, Bretton Woods(USA)
28 Aout, 2 sept 1983
- [SATA 81]
T.SATA, F.KIMURA, A.AMANO: "Robot simulation system as a task programming tool"
11th ISIR, Tokyo, October 1981.
- [Shimano 84]
SHIMANO B.B., GESCHKE C.C and SPALDING C.H.: "VAL-II: a new robot control system for automatic manufacturing"
Proc. IEEE International Conference on Robotics, Atlanta 1984.
- [Summers 84]
SUMMERS P.D. and GROSSMAN D.D: "XPROBE: an experimental system for programming robots by examples"
The International Journal of Robotics Research, Vol.3, No.1, p25-39
- [Taylor 79]
TAYLOR R.H. IBM Journal of Research and Development Vol 23, No4, Juillet 1979
- [Taylor 82]
R.H.TAYLOR, P.D.SUMMERS, J.M.MEYER: "AML: A manufacturing language"
The International Journal of Robotics Research, Vol.1, 1982.
- [Unimation 79]
UNIMATION, Inc.: "User's guide to VAL, a robot programming and control system"
Version 11, Unimation Inc., Février 1979.
- [Wesley 80]
M.A.WESLEY et al.: "A geometric modeling system for automated mechanical assembly"
IBM Journal of Research and Development, Janvier 1980.

```

Ligne      Texte du programme
 1      PROGRAMME TESTLV;
 2
 3      CO DECLARATION DE LA COLLECTION D'OBJETS AVEC LAQUELLE ON VEUT TRAVAILLER;
 4      %COLL COL1
 5
 6      ENTIER I;
 7      OBJET O1;
 8
 9      DEBUT
10      CO INITIALISATION;
11      CALIBRER CAMERA 1;
12
13      DEB;
14      ECRIRE "DONNER LE SEUIL DESIRE : ";
15      LIRE I;
16      FIXER SEUIL CAMERA 1 A I;
17      CO ACQUISITION D'UNE IMAGE SUR LA CAMERA 1 ;
18      ACQUERIR IMAGE 1 SUR CAMERA 1;
19
20      CO ON AFFICHE L'IMAGE ;
21      AFFICHER IMAGE 1;
22
23      CO EXTRACTION DES CONTOURS DE CETTE IMAGE ;
24      EXTRAIRE CONTOURS IMAGE 1;
25
26      CO ON AFFICHE LES CONTOURS;
27      AFFICHER CONTOURS IMAGE 1;
28
29      CO Y A-T-IL UN OBJET QUELCONQUE DE LA COLLECTION COL1 DANS CETTE IMAGE ?
30      RECONNAITRE O1 DANS IMAGE 1;
31
32      CO O1, VARIABLE OBJET : O1.INFO INDIQUE SI ON A RECONNU UN OBJET .
33      CO SI OUI ALORS O1.REF DONNE LA REFERENCE DE L'OBJET RECONNU
34      CO ET O1.X,O1.Y,O1.TETA SON ORIENTATION ;
35
36      CO SI UN OBJET A ETE RECONNU, ON ECRIT SON NOM ;
37      SI O1.INFO=RECONNU ALORS ECRIRE O1;
38      SINON ECRIRE "PAS D'OBJET RECONNU DANS CETTE IMAGE";
39      FINI;
40
41      ECRIRE " FIN ? (1=OUI)";
42      LIRE I;
43      SI I=1 ALORS ALLERA FINI;FINI;
44
45      CO AVANT DE RECOMMENCER ON EFFACE CE QUI EST AFFICHE ET ON LIBERE
46      CO L'IMAGE UTILISEE ;
47      EFFACER IMAGE 1;
48      EFFACER CONTOURS IMAGE 1;
49      LIBERER IMAGE 1;
50      ALLERA DEB;
51
52      FINI;
53      ECRIRE "FIN DU PROGRAMME ";
54      FIN;
Programme correct

```


AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU le rapport de présentation de Monsieur le Professeur J.C LATOMBE,

Monsieur Jean-François MIRIBEL

est autorisé à présenter une thèse en soutenance en vue de l'obtention du titre de

DOCTEUR de TROISIEME CYCLE, spécialité "Informatique".

Fait à Grenoble, le 17 octobre 1984

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

