Réalisation d'un support expérimental de recherche pour le projet robotique PANDORE : définition et implantation du langage LM

Emmanuel Mazer

▶ **To cite this version:**

Emmanuel Mazer. Réalisation d'un support expérimental de recherche pour le projet robotique PANDORE : définition et implantation du langage LM. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1981. Français. NNT : . tel-00294243

HAL Id: tel-00294243

https://theses.hal.science/tel-00294243

Submitted on 9 Jul 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
« Génie Informatique»

par

Emmanuel MAZER

REALISATION D'UN SUPPORT EXPERIMENTAL DE RECHERCHE POUR LE PROJET ROBOTIQUE PANDORE. DEFINITION ET IMPLANTATION DU LANGAGE LM.

Thèse soutenue le 12 janvier 1981 devant la commission d'examen

L. BOLLIET Président

P. COIFFET
O. FLACH
J.C. LATOMBE Examinateurs
R. POPPLESTONE
G. VEILLON

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioélectricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahïm	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

Je tiens à remercier

Monsieur Louis BOLLIET, Professeur à l'IUT d'Informatique de Grenoble, qui m'a fait l'honneur de présider le jury de cette thèse,

Monsieur Gérard VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui a encouragé ces travaux et favorisé le développement de la robotique au sein de l'ENSIMAG,

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

Monsieur Philippe COIFFET, Maître de Recherche au CNRS, et Monsieur Olivier FLACH, Chef de projet à la SCEMI, pour l'intérêt qu'ils portent à nos travaux,

Monsieur Robin POPPLESTONE qui me fait l'honneur de participer au jury de cette thèse.

Je tiens à exprimer ma gratitude envers les personnes qui ont directement participé à la réalisation de ce projet

Monsieur Pascal DI GIACOMO, qui a réalisé l'ensemble des montages électriques du matériel mis en oeuvre et qui a su, par sa compétence, aplanir bien des difficultés,

Monsieur Augustin LUX, qui a réalisé le logiciel de traitement d'images et qui a participé à la définition de l'interpréteur du langage LM.A,

Monsieur Jean-François MIRIBEL, qui a implanté de nombreuses primitives du langage LM.A et pour les nuits blanches que lui a valu cette thèse,

Monsieur Jean-Paul SERPAGGI pour le soin qu'il a apporté au câblage des cartes microprocesseurs.

Je remercie les personnes qui ont aidé à la réalisation de ce projet

Monsieur Michel SAKAROVITCH, Professeur à l'Université Scientifique et Médicale de Grenoble, qui a encouragé ces travaux au sein du Laboratoire,

Monsieur Raymond BOUTTAZ, Responsable de l'atelier MICRO-INFORMATIQUE pour ses conseils et l'ensemble des moyens qu'il a mis à notre disposition,

Messieurs Jean-Pierre SCHOELLKOPF, Jacques LAURENT, Gérard BAILLE, pour les conseils et l'aide qu'ils ont apportés dans la définition et la mise au point des cartes micro-processeurs,

Monsieur Jean-Paul EYNARD pour ses conseils sur l'utilisation de l'émulateur TECTRONIX 8002,

Messieurs André EBERHARD et Bernard CASSAGNE pour le temps qu'ils m'ont accordé lors de mon apprentissage sur le SOLAR 16-65,

Monsieur LEDEP de ILL qui a mis à notre disposition un important logiciel pour le MICRO 1-03,

Messieurs CIVIDINO, CHEVAL, STOECKEL du Laboratoire de Spectroscopie Physique de l'Université Scientifique et Médicale de Grenoble pour les conseils en optique, mécanique et pour la réalisation du boîtier de la caméra CCD.

Je voudrais aussi remercier Marie-José DOREL qui a assuré la dactylographie de cette thèse, Mademoiselle Brigitte HUMBERT pour les dessins des figures, Mademoiselle Annie CULET, ainsi que le Service de Reproduction qui a réalisé le tirage.

Le travail décrit dans ce rapport a bénéficié des financements suivants :

- *Crédit d'équipement spécial de l'IMAG,*
- *Crédit d'équipement du GIS Mini & Micro Informatique de l'INPG,*
- *ATP CNRS "Internationale" n° 4288,*
- *ATP CNRS "Intelligence Artificielle" n° 4272,*
- *Contrat DGRST "Robotique" n° AUT/P74 bis,*
- *Contrat ADEPA,*
- *Contrat CERCI.*

INTRODUCTION

La Robotique a toujours constitué un thème majeur d'inspiration et d'application pour l'Intelligence Artificielle [19][15]. Elle a motivé de nombreux travaux, notamment sur la génération de plans d'actions (par exemple : [3]) et l'analyse de scènes visuelles (par exemple : [28]).

Pendant longtemps ces travaux ont eu un caractère principalement académique. Toutefois, l'évolution actuelle de la Robotique Industrielle vers une plus grande sophistication informatique, due d'une part au besoin d'automatiser des tâches de plus en plus variées et complexes (montage d'assemblages notamment), et d'autre part à la diminution du coût du matériel électronique (calculateur, mémoires, capteurs), donne une importance nouvelle à ces travaux [8].

Aussi, l'équipe d'Intelligence Artificielle de l'IMAG a-t-elle choisi en 1978 de s'engager dans un ensemble de recherches situées à l'intersection de la Robotique et de l'Intelligence Artificielle. Ces recherches sont regroupées en un projet appelé projet PANDORE.

Il est clair qu'une recherche en Robotique, même si elle a une orientation théorique importante, ne peut longtemps se passer d'une expérimentation poussée des modèles et méthodes élaborées. C'est pourquoi il a été décidé de développer un support expérimental au sein du projet.

Notre travail se place dans le cadre de la réalisation d'un tel support. Il comporte deux parties distinctes :

(1) la définition et la mise en oeuvre d'un matériel de Robotique composé de manipulateurs munis de capteurs (force, vision),

(2) la conception et l'implantation d'un langage de programmation, appelé LM, pour la description de tâches de manipulation et d'assemblage.

Dans la première partie, notre souci principal a été de mettre en oeuvre un ensemble matériel qui soit extensible pour s'adapter aux évolutions des recherches effectuées dans le cadre du projet PANDORE. Nous avons inclus dans cet ensemble les composantes fonctionnelles (manipulation, perception des forces et vision) dont la nécessité a semblé la plus immédiate. Par ailleurs, nous avons essayé de répartir les unités de

traitement pour permettre l'utilisation simultanée et indépendante de plusieurs sous-ensembles de ce matériel.

Dans la deuxième partie, nous avons cherché à développer un formalisme cohérent pour exprimer des tâches de manipulation complexe en termes des déplacements de manipulateurs, d'actions de leurs outils terminaux et des informations transmises par les capteurs. Notre but a été d'inclure dans ce formalisme des facilités de programmation très évoluées et bien adaptées à la robotique.

Bien qu'élaboré dans le cadre du projet PANDORE, le langage LM présente un intérêt industriel certain. En effet, jusqu'à présent, le seul mode de programmation réellement utilisé dans l'industrie est la programmation dite "par l'exemple". Il présente de nombreuses limitations [24], qui restreignent considérablement son application à certains types de tâches (manipulation complexe notamment). LM évite ces limitations tout en incluant des possibilités de programmation par l'exemple.

L'intérêt industriel pour le langage LM s'est traduit par deux contrats. L'un, avec la société CERCI, a porté sur la définition du langage. L'autre, avec la société SCEMI, concerne l'implantation du langage sur un matériel (manipulateur, commande de puissance, calculateur) développé par cette société.

Ce rapport présente en détail notre travail. Il comporte 6 chapitres :

- le 1er chapitre replace notre travail dans le cadre du projet PANDORE,
- le 2ème chapitre définit le langage LM,
- le 3ème chapitre présente une analyse fonctionnelle de ce langage,
- le 4ème chapitre décrit le matériel mis en oeuvre,
- le 5ème chapitre décrit l'implantation actuelle de LM sur ce matériel,
- le 6ème chapitre présente l'expérimentation que nous avons faite du matériel à travers le logiciel implanté.

CHAPITRE 1

PRESENTATION DE NOTRE TRAVAIL

Dans ce chapitre, nous replaçons notre travail dans le cadre du projet PANDORE.

1. LE PROJET PANDORE [11]

Le projet PANDORE se situe à l'intersection de l'Intelligence Artificielle et de la Robotique. Il a trois objectifs principaux :

a) Le *premier objectif* est de nature fondamentale. Il s'agit d'élaborer des modèles et des méthodes d'Intelligence Artificielle,d'une part pour augmenter la compétence fonctionnelle des robots, et d'autre part pour faciliter leur mise en oeuvre.

Les thèmes de recherche envisagés incluent la prise de décisions automatique (choix des actions en fonction d'objectifs à atteindre), la modélisation de raisonnements, la perception à l'aide de capteurs évolués (vision, force, toucher), la modélisation géométrique de l'univers des robots, la création et l'exploitation de bases de connaissances et l'apprentissage (induction de méthodes et/ou de concepts à partir de cas particuliers).

Pour donner une illustration plus concrète à ces thèmes, il a été choisi de travailler, au moins pendant une première phase, sur des problèmes liés à la manipulation complexe d'objets rigides, comme le montage d'assemblages, à l'aide de robots du type bras articulés (manipulateurs).

Parmi les problèmes abordés figurent :

- La saisie automatique d'un objet par un manipulateur muni d'une pince à deux mors, étant donné a priori un modèle géométrique de l'objet et le contexte de la tâche (notamment la destination de l'objet).
- Le choix des déplacements "fins" qu'un manipulateur doit effectuer pour assembler deux objets de positions imparfaitement connues, en utilisant des données sur la géométrie des objets, la relation d'assemblage à réaliser et les informations transmises à chaque instant par des capteurs de force.

- La coopération de plusieurs robots autonomes en fonction des capacités (et des incapacités) de chacun d'eux et d'objectifs à atteindre.
- L'identification et la localisation d'objets tridimensionnels de modèles connus dans une scène vue sous un angle non connu a priori, les objets pouvant se cacher partiellement.

b) Le *deuxième objectif* est de nature expérimentale. Il s'agit d'implanter une technologie informatique reposant sur les modèles et les méthodes élaborés dans le cadre du premier objectif, et de réaliser une expérimentation aussi complète que possible de cette technologie. Cette expérimentation est essentielle pour valider les résultats théoriques obtenus et aider à mieux formuler les problèmes à résoudre.

Cet objectif implique non seulement l'existence d'un matériel expérimental approprié, mais aussi celle de logiciels de base servant de supports aux programmes d'Intelligence Artificielle ou d'outils à leur implantation. Il a ainsi conduit à entreprendre les trois travaux suivants :

- Définition et mise en oeuvre d'un matériel composé de systèmes mécaniques articulés, de capteurs divers et d'unités de calcul.
- Conception et implantation d'un langage de programmation pour commander les opérations des systèmes mécaniques et pour communiquer avec les capteurs (langage LM).
- Conception et implantation d'un système d'analyse d'images, incluant diverses fonctions : extraction d'éléments de contraste dans une image digitalisée sur plusieurs niveaux de gris, formation de lignes de contraste (squelétisation), comparaison de lignes, analyse de connexité dans une image binaire, ... (système CAIMAN).

Il est également envisagé dans le cadre de cet objectif de réaliser un système de description de modèles géométriques adaptés aux besoins des problèmes à résoudre.

c) Le *troisième objectif* est de nature appliquée. Il s'agit de contribuer à développer certains résultats (notamment leurs performances) jusqu'à un niveau d'exploitation industrielle.

En effet, les résultats obtenus dans le cadre des deux objectifs précédents devraient permettre d'étendre le domaine d'application des robots vers une plus grande complexité de tâches et réduire le coût de leur mise en oeuvre sur de nouvelles tâches. Compte tenu de l'actuelle évolution de la robotique industrielle vers une plus grande sophistication informatique [8], ces résultats devraient rapidement trouver des applications pratiques.

Ce troisième objectif se traduit par des contacts suivis avec plusieurs industriels susceptibles de soumettre des problèmes pratiques et/ou d'utiliser certains résultats. Ces contacts ont une influence importante sur le choix des problèmes abordés. Ainsi, plusieurs des travaux mentionnés ci-dessus font l'objet de contrats en collaboration avec des industriels.

De plus, une nouvelle implantation du langage LM est en préparation sous contrat avec la Société SCEMI depuis Octobre 1980. Elle sera opérationnelle sur le système de commande du manipulateur construit par cette société à des fins de montage d'assemblages (contacteurs, petits moteurs), et devrait être commercialisée avec ce matériel.

2. NOTRE TRAVAIL

Notre travail se situe essentiellement dans le cadre de l'objectif expérimental du projet PANDORE (§ 1.b). Il a consisté :

- à définir et à mettre en oeuvre le matériel expérimental,
- à concevoir et à implanter le langage LM.

2.1. Le matériel

Le matériel doit inclure les composantes fonctionnelles indispensables à l'expérimentation des programmes d'Intelligence Artificielle envisagée dans le § 1.

Parmi les éléments essentiels d'un tel ensemble figurent notamment :

- un système mécanique programmable pouvant déplacer et orienter les objets,
- des capteurs,
- des unités de traitement assurant le contrôle des éléments précédents.

a) Le système mécanique

Nous avons mis en oeuvre deux robots manipulateurs, équipés chacun d'une pince de préhension, possédant 4 et 2 degrés de liberté. En faisant collaborer ces deux systèmes, on dispose des 6 degrés de liberté indépendants nécessaires pour amener un objet dans une position et une orientation quelconque (dans un certain volume accessible).

Les actionneurs de ces deux systèmes sont des moteurs pas à pas. On contrôle la position des deux manipulateurs en envoyant un nombre donné de pas à chaque actionneur. Cette commande, en boucle ouverte, rend les deux systèmes programmables.

b) Les capteurs

Nous avons choisi de placer un capteur de force dans le poignet du manipulateur à 4 ddl. Le capteur mis en oeuvre est un capteur piézo-électrique ; il permet de mesurer la composante de la force (FZ) suivant l'axe de la pince du manipulateur à 4 ddl et le moment autour de cet axe (MZ). Il sera prochainement complété pour mesurer les composantes latérales de force (FX et FY).

Ce capteur de force sera complété plus tard par un socle fixe de mesure d'efforts indépendant du manipulateur.

Le système de vision est constitué par une caméra CCD (100 × 100) connectée au SOLAR 16-65 de l'IMAG. Toutefois, en raison de sa faible résolution, il a été décidé par la suite de compléter ce système par une deuxième caméra à tube vidéo dont la digitalisation fournit une image de 512 par 512 points.

c) *Unités de traitements*

Plusieurs calculateurs rendent le système programmable ; il s'agit de :

- deux systèmes microprocesseurs pour la commande de bas niveau des systèmes mécaniques,
- un calculateur à base d'un DEC LSI 11/2 pour l'interprétation du langage LM,
- un calculateur SOLAR 16/65 pour le traitement des images fourni par la caméra CCD,
- un calculateur à base d'un DEC LSI 11/23 pour le traitement des images fourni par la caméra vidéo.

2.2. Le langage LM

Le langage LM est destiné à donner au matériel mis en oeuvre les qualités de programmabilité facilitant l'implantation de programmes d'Intelligence Artificielle. Toutefois, ses concepts ne sont pas spécifiques d'une classe de manipulateurs. Ainsi l'implantation actuelle de LM, qui sert à commander deux manipulateurs à 4 et 2 degrés de liberté actionnés par des moteurs pas à pas, est en cours d'adaptation pour commander un manipulateur à 6 degrés de liberté actionné par des moteurs à courant continu (manipulateur SCEMI).

Nous avons conçu LM pour qu'il constitue un formalisme cohérent et commode permettant de décrire des tâches de manipulation en termes des déplacements d'un ou de plusieurs manipulateurs, indépendamment de leurs structures mécaniques, et des opérations de leurs outils terminaux (pinces de préhension, pinces à souder, tournevis, ...). LM se situe donc au même niveau conceptuel, dit niveau "effecteur" [9], que des langages tels que WAVE [16], AL [5], VAL [26], MAL [7] et LAMA-S [4]. Toutefois, il présente une combinaison originale de caractéristiques relativement à ces langages. Nous en présentons quelques-unes ci-dessous :

- LM permet de modéliser la géométrie de l'univers des manipulateurs (positions des objets, des outils terminaux, ...) à l'aide de repères cartésiens.

- Il permet de spécifier les déplacements d'un manipulateur en termes des déplacements des repères attachés aux objets à déplacer, et relativement à des repères quelconques.
- Il inclut plusieurs formes de déplacements, adaptés notamment aux déplacements longs (déplacements passant par des positions intermédiaires) et courts (déplacements suivant des transformations).
- Il permet d'exécuter des déplacements préenregistrés, rendant ainsi possible une certaine forme de programmation par l'exemple.
- Il inclut des facilités pour décrire l'exécution simultanée et coordonnée de déplacements des manipulateurs et d'actions de leurs outils, en utilisant les données transmises par les capteurs.
- Grâce à la notion de "variables d'état", il permet d'utiliser simplement une large gamme de capteurs.
- Il offre la possibilité de surveiller des conditions pendant les déplacements et les actions des outils (dépassement d'un seuil par une force par exemple).
- Il permet de coordonner l'évolution des manipulateurs avec des processus externes (par exemple : autres manipulateurs, robots mobiles, tapis roulants).
- Il inclut la plupart des facilités habituellement offertes par les langages de programmation classiques : tableaux, branchement, instructions SI et TANTQUE, appel de procédures, entrées-sorties, ...

De plus, l'interpréteur de LM a été implanté sur un calculateur de taille réduite (Micro 1/03 de PLESSEY). Cette implantation n'est toutefois pas tout à fait complète. Elle ne permet d'interpréter qu'un langage syntaxiquement élémentaire, mais permettant d'exprimer simplement la sémantique de LM. Ce langage est celui d'une machine appelée "machine LM".

La machine LM est implantée entièrement en Fortran et elle communique avec les actionneurs des systèmes mécaniques et les capteurs par des "primitives actionneurs-capteurs". Elle ne fait appel qu'à un nombre très restreint de fonctions du système d'exploitation du Micro 1/03 (RT-11) de telle sorte qu'elle est presque entièrement portable.

L'intérêt du langage LM ne se limite pas au projet PANDORE. Comme nous l'avons vu au § 1.c, un développement industriel du langage est en cours avec la Société SCEMI.

CHAPITRE 2

DEFINITION DU LANGAGE LM

La description du langage LM donnée dans ce chapitre est illustrée par des exemples dans l'Annexe A.

1. COMPOSITION D'UN PROGRAMME

Un programme en LM se compose d'un *programme principal* et de *procédures* :

<programme> ::= <programme principal> {<procédures>}*

Le programme principal et les procédures sont appelés *unités de programme.*

Chaque unité de programme est composée de déclarations et d'instructions séparées par le symbole ";". Chaque instruction peut être identifiée par une étiquette. Une procédure commence par une en-tête.

On a :

<programme principal> ::= <texte de programme>

<procédure> ::= <en-tête de procédure><texte de programme>

<texte de programme> ::= {<déclaration>;}*<instruction étiquetable>

<instruction étiquetable> ::= [<étiquette>:]<instruction>

<étiquette> ::= identificateur

<instruction> ::= <affectation>|<tantque>|<si>|...|<bloc d'instructions>

<bloc d'instructions> ::= DEBUT <liste d'instructions> FIN

<liste d'instructions> ::= <instruction étiquetable>{;<instruction étiquetable>}*

De plus, on peut toujours insérer entre deux entités lexicographiques un commentaire de la forme C <commentaire>.

L'exécution d'un programme commence par l'exécution de la première instruction du programme principal et se termine avec celle du programme principal.

Note :

Nous utilisons les conventions suivantes :

- [*entité*] *signifie que l'entité est facultative,*
- {*entité*}* *signifie qu'elle apparaît 0 ou 1 ou 2 ou ... fois.*

2. TYPE DE DONNEES ET OPERATIONS ELEMENTAIRES

2.1. Les types de données

On distingue dans le langage LM six types de données : entier, réel, booléen, vecteur, transformation et repère. Nous les définissons ci-dessous aux limitations imposées par une implantation près :

- un *entier* est un nombre entier relatif ;
- un *réel* est un nombre réel ; on peut toujours utiliser un entier à la place d'un réel ;
- un *booléen* est une condition binaire ;
- un *vecteur* est un vecteur libre de $\mathbb{R}^3$;
- une *transformation* est une application de $\mathbb{R}^3$ dans $\mathbb{R}^3$ composée d'une rotation et d'une translation ;
- un *repère* est un repère cartésien OXYZ dans $\mathbb{R}^3$.

Les symboles de constantes du langage sont :
- les nombres entiers relatifs, de type entier,
- les nombres réels, de type réel,
- VRAI et FAUX, de type booléen,
- VX, VY, VZ, de type vecteur, qui représentent trois vecteurs libres de $\mathbb{R}^3$ de composantes (1,0,0), (0,1,0) et (0,0,1),
- STATION, de type repère.

Le repère STATION peut être considéré comme le repère de référence de l'utilisateur.

Dans le cas où ces données représentent des grandeurs physiques, leur interprétation dépend des unités de mesure. Le choix de ces unités peut varier d'une implantation à une autre. On peut prendre par exemple le millimètre comme unité de longueur, le newton comme unité de force...

2.2. Opérations élémentaires sur les données

Elles sont exprimées à l'aide d'opérateurs et de fonctions.

a) Opérations sur les entiers

- Addition de deux entiers e1 et e2 :

  e1 + e2

  Le résultat est un entier.

- Soustraction de deux entiers e1 et e2 :

  e1 - e2

  Le résultat est un entier.

- Changement de signe d'un entier e :

  -e

  Le résultat est un entier.

- Multiplication de deux entiers e1 et e2 :

  e1 * e2

  Le résultat est un entier.

- Division d'un entier e1 par un entier e2 :

  e1 / e2

  Le résultat est un entier (partie entière).

- Elévation d'un entier e1 à la puissance entière e2 :

  e1**e2

  Le résultat est un entier.

- Comparaison de deux entiers e1 et e2 :

  e1 < e2 e1 <= e2 e1 = e2 e1 >= e2 e1 > e2

  Les résultats sont des booléens.

- Valeur absolue d'un entier e :

  ABS(e)

  Le résultat est un entier.

b) Opérations sur les réels

- Addition de deux réels r1 et r2 :

  r1+r2

  Le résultat est un réel.

- Soustraction de deux réels r1 et r2 :

  r1-r2

  Le résultat est un réel.

- Changement de signe d'un réel r :

  -r

  Le résultat est un réel.

- Multiplication de deux réels r1 et r2 :

  r1 * r2

  Le résultat est un réel.

- Division d'un réel r1 par un réel r2 :

  r1 / r2

  Le résultat est un réel.

- Elévation d'un réel r1 à une puissance réelle r2 :

  r1**r2

  Le résultat est un réel.

- Comparaisons de deux réels r1 et r2 :

  r1 < r2 r1 <= r2 r1 = r2 r1 >= r2 r1 > r2

  Les résultats sont des booléens.

- Partie entière d'un réel r :

  ENTIER(r)

  Le résultat est un entier.

- Valeur absolue d'un réel r :

  ABS(r)

  Le résultat est un réel.

- Sinus, cosinus, tangente, arcsinus, arccosinus, arctangente d'un réel r :

  SIN(r) COS(r) TAN(r) ARSIN(r) ARCOS(r) ARTAN(r)

  Les résultats sont des réels.

- Exponentielle et logarithme népérien d'un réel r :

  EXP(r) LOG(r)

  Les résultats sont des réels.

c) *Opérations sur les booléens*

- Union de deux booléens b1 et b2 :

  b1 OU b2

  Le résultat est un booléen.

- Conjonction de deux booléens b1 et b2 :

  b1 ET b2

  Le résultat est un booléen.

- Négation d'un booléen b :

  NON b

  Le résultat est un booléen.

d) *Opérations sur les vecteurs*

- Génération du vecteur ayant les réels x, y et z pour composantes :

  VECTEUR(x,y,z)

  Le résultat est un vecteur.

[*Remarque : Les composantes du vecteur ainsi engendré ne s'interprètent dans aucun repère fixé a priori*].

- Extraction des composantes d'un vecteur v :

  X(v) Y(v) Z(v)

  Les résultats sont des réels.

- Addition de deux vecteurs v1 et v2 :

  v1 + v2

  Le résultat est un vecteur.

- Soustraction de deux vecteurs v1 et v2 :

  v1 - v2

  Le résultat est un vecteur.

- Changement de signe d'un vecteur v :

  -v

  Le résultat est un vecteur.

- Multiplication d'un vecteur v par un réel r :

  r * v

  Le résultat est un vecteur.

- Produit scalaire de deux vecteurs v1 et v2 :

  PSCAL(v1,v2)

  Le résultat est un réel.

- Produit vectoriel de deux vecteurs v1 et v2 :

  PVECTO(v1,v2)

  Le résultat est un vecteur.

- Longueur d'un vecteur v :

  LONG(v)

  Le résultat est un réel.

- Application d'une transformation t à un vecteur v :

  t * v

  Le résultat est un vecteur.

[*Remarque : La translation composant t n'a aucune influence sur le résultat, car v représente un vecteur libre de* $\mathbb{R}^3$].

e) Opérations sur les transformations

- Génération de la rotation d'angle x (x est un réel) autour d'un axe parallèle au vecteur v :

  ROTATION(v,x)

  Le résultat est une transformation.

  Le signe de x doit être établi suivant la règle de la "main droite". La longueur de v n'a pas d'influence sur la rotation engendrée.

  [*Remarque : La transformation ainsi engendrée n'est interprétée dans aucun repère fixé a priori. Lorsqu'elle est appliquée dans un repère particulier, les composantes du vecteur v sont interprétées dans ce repère et l'axe de rotation passe par l'origine de ce repère*].

- Génération de la translation de longueur x (x est un réel) suivant le vecteur v :

  TRANSLATION(v,x)

  Le résultat est une transformation.

  La longueur de v n'a pas d'influence sur la translation engendrée.

[*Remarque : La transformation ainsi engendrée n'est interprétée dans aucun repère fixé a priori. Lorsqu'elle est appliquée dans un repère particulier, les composantes du vecteur v sont interprétées dans ce repère*].

- Composition de deux transformations t1 et t2, t1 étant appliquée avant t2 :

  t2 * t1

  Le résultat est une transformation.

  [*Remarque : Les deux transformations sont appliquées dans le même repère*].

- Inversion d'une transformation :

  INV(t)

  Le résultat est la transformation telle que INV(t) * t soit la transformation identité.

- Extraction de la rotation et de la translation composant la transformation t :

  ROT(t) TRSL(t)

  Les résultats sont des transformations telles que t = TRSL(t) * ROT(t).

  [*Remarque : Si t = ROTATION (v1,x) * TRANSLATION(v2,y), TRSL(t) n'est en général pas égal à TRANSLATION(v2,y)*].

- Extraction du vecteur de la rotation ROT(t) qui compose la transformation t :

  VECT.ROT(t)

  Le résultat est un vecteur.

  La longueur du vecteur extrait est l'angle de la rotation.

- Extraction du vecteur de la translation TRSL(t) qui compose la transformation t :

  VECT.ROT(t)

  Le résultat est un vecteur.

  La longueur du vecteur extrait est la longueur de la translation.

6) *Opérations sur les repères*

- Application de la transformation t au repère r :

  t * r ou r * t

  Le résultat est un repère r.

  Soit t_r la transformation qui fait passer de STATION (cf. § 2.1) à r.

  Par définition :

  . le repère t * r est obtenu en appliquant la transformation t * t_r dans STATION,

  . le repère r * t est obtenu en appliquant la transformation t_r * t dans STATION.

  [*Remarque importante* :
  Il est souvent utile d'interpréter la composition t_r * t *de gauche à droite. Cela revient à considérer r * t comme étant le repère qui se déduit de r en appliquant t dans r. La définition de t doit alors être interprétée dans r. Par exemple, si ROT(t) = ROTATION(VX,x) et TRSL(t) = TRANSLATION(VY,y), VY et VY doivent être considérés comme les vecteurs dont les composantes suivant les axes de r sont 1,0,0 et 0,1,0 respectivement*].

- Extraction de la transformation entre le repère r1 et le repère r2 :

  TRANSF(r1,r2)

  Le résultat est la transformation t telle que r2 = r1 * t.

- Calcul de la distance euclidienne entre les origines de deux repères r1 et r2 :

  DISTANCE(r1,r2)

  Le résultat est un réel.

- Calcul de la distance angulaire entre deux repères r1 et r2 :

  ANGLE(r1,r2)

  Le résultat est le réel x tel que x = LONG(VECT.ROT(TRANSF(r1,r2))).

Dans la suite, nous appellerons *position absolue*, ou plus simplement *position* d'un repère r la transformation TRANSF(STATION,r) ; nous appellerons position du repère r2 relative au repère r1 la transformation TRANSF(r1,r2).

3. VARIABLES ET EXPRESSIONS

3.1. Les variables

a) Les variables déclarées

A l'exception des variables d'état (cf. § b), toutes les variables qui apparaissent dans une unité de programme doivent être déclarées au début de l'unité de programme.

Les variables déclarées sont des variables simples ou des variables indicées (tableaux).

Les déclarations des variables sont de la forme :

<type> [GLOBAL] <nom de variable>{,<nom de variable>}*

ou <type> TABLEAU [GLOBAL] <tableau>{,<tableau>}*

avec :

<tableau> ::= <nom de variable> (<dimension>)

<dimension> ::= <nombre> {,<nombre>}*

<nombre> ::= constante entière positive

<type> ::= ENTIER|REEL|BOOLEEN|VECTEUR|TRANSFORMATION|REPERE

<nom de variable> ::= identificateur.

Une variable dont la déclaration ne comporte pas la clause GLOBAL est locale à l'unité de programme où figure la déclaration. Une variable dont la déclaration comporte la clause GLOBAL est globale à toutes les unités de programme dans lesquelles la variable est déclarée ainsi.

b) Les variables d'état

Les variables d'état représentent des grandeurs qui ne sont pas sous le contrôle direct du programmeur (celui-ci ne peut pas leur affecter une valeur par une instruction d'affectation). Leurs valeurs sont automatiquement tenues à jour et leurs noms sont connus a priori de l'interpréteur. Elles sont globales à toutes les unités de programme.

Certaines variables d'état existent dans toutes les implantations du langage. Ce sont :

- ROBTi, i = (1,2,...) : variable de type repère qui désigne un repère attaché à l'extrémité utile du manipulateur i (sa valeur dépend à chaque instant des données transmises par les capteurs de position associés aux degrés de liberté du manipulateur).

- TEMPS : variable de type réel qui mesure le temps écoulé depuis le début de l'exécution du programme.
- TEMPS.DEPLi : variable de type réel qui mesure le temps écoulé depuis le début du dernier déplacement du manipulateur i.
- TEMPS.ACTIONi : variable de type réel qui mesure le temps écoulé depuis le début de la dernière action de l'outil du manipulateur i.
- EN.DEPLi : variable de type booléen qui est égale à VRAI lorsque le manipulateur i effectue un déplacement et à FAUX dans le cas contraire.
- EN.ACTIONi : variable de type booléen qui est égale à VRAI lorsque le manipulateur i effectue une action et à FAUX dans le cas contraire.

Les autres variables d'état dépendent du matériel sur lequel est implanté le langage. Ce sont par exemple :

- FXi, FYi, FZi : variables de type réel qui mesurent les composantes dans le repère ROBOTi de la force exercée à l'origine de ce repère par le manipulateur.
- MXi, MYi, MZi : variables de type réel qui mesurent les moments exercés autour des axes du repère ROBOTi.
- ECARTi : variable de type réel qui mesure la distance entre les deux mors parallèles d'une pince servant d'outil au manipulateur i.
- SIGNAL1,SIGNAL2,... : variables de type réel sans signification imposée a priori ; de telles variables d'état peuvent servir, par exemple, à ajouter des capteurs provisoires, ou à se synchroniser avec d'autres processus.

Dans la suite, nous noterons <var.entier>, <var.réel>, <var.booléen>, <var.vecteur>, <var.transformation> et <var.repère> les variables de types entier, réel, booléen, vecteur, transformation et repère respectivement ; nous noterons <variable> une variable de type quelconque.

3.2. Les expressions

Les expressions sont des constructions formées à l'aide des opérateurs +, -, *, **, /, <, <=, =, >=, >, ET, OU et NON, et d'opérandes (constantes, variables, fonctions, appels de procédures, expressions entre parenthèses), par exemple :

```
t1*(v1+t2*v2)
TRSL(TRANSF(rep1,rep2))*ROTATION(VECTEUR(x1,x2,x3),ARCOS(x)).
```

L'évaluation d'une expression est faite en évaluant tous les opérandes qui ne sont pas des constantes, puis en appliquant les opérateurs par ordre de priorité décroissante (d'abord **, puis * et /, puis + et -, puis <, <=, =, >=, >, puis NON, et enfin OU et ET). Les opérateurs de mêmes priorités sont appliqués de gauche à droite.

Dans la suite, nous noterons <entier>, <réel>, <booléen>, <vecteur>, <transformation> et <repère> les expressions qui s'évaluent à des données de types entier, réel, booléen, vecteur, transformation et repère respectivement ; nous noterons <expression> une expression qui s'évalue à une donnée de type quelconque.

4. INSTRUCTIONS

4.1. Instruction d'affectation

Une instruction d'affectation donne une valeur à une variable. Elle est de la forme :

<variable>:= <expression>

où <variable> et <expression> sont du même type.

La variable <variable> ne doit pas être une variable d'état.

4.2. Instructions de contrôle

a) Instruction ALLERA

L'instruction

ALLERA <étiquette>

déroute l'exécution d'une unité de programme à l'instruction identifiée par <étiquette> dans cette unité. L'étiquette doit figurer dans le bloc d'instructions courant ou dans un bloc englobant.

b) *Instruction TANTQUE*

L'instruction

TANTQUE <booléen> FAIRE <instruction>

exécute itérativement <instruction> tant que l'expression <booléen> s'évalue à VRAI.

c) *Instruction SI*

Elle est de la forme :

SI <booléen> ALORS <instruction>[SINON <instruction>]

Si l'expression <booléen> s'évalue à VRAI, l'instruction suivant ALORS est exécutée. Dans le cas contraire, l'instruction suivant SINON est exécutée. La partie SINON est facultative.

d) *Instruction PAUSE*

Elle est de la forme :

PAUSE <instruction conditionnelle> {;<instruction conditionnelle>}* FINPAUSE

où :

<instruction conditionnelle> ::= <booléen>:<instruction>.

Elle met l'exécution du programme en attente jusqu'à ce que l'une des expressions <booléen> s'évalue à VRAI. L'instruction située derrière cette instruction est alors exécutée.

Exemple :

```
PAUSE
   MZ > 20.0 : DEBUT
                  ARRETER ACTION ROBOT1 ;
                  RETOUR
               FIN ;
   FZ < 10.0 : DEPLACER ROBOT2 DE T
FINPAUSE
```

Si plusieurs expressions <booléen> s'évaluent à VRAI "simultanément", l'instruction qui suit l'une quelconque de ces expressions est exécutée.

L'instruction PAUSE est utile pour coordonner des déplacements de manipulateurs et des actions de leurs outils (cf. Annexe A).

e) Fin de l'exécution du programme principal

L'instruction

RETOUR

termine l'exécution du programme principal, lorsqu'elle figure dans celui-ci.

La fin de l'exécution du programme principal n'est effective que lorsque les manipulateurs et leurs outils sont immobiles.

4.3. Création de liaisons entre repères

a) Instruction LIER

L'instruction

LIER <var.repère><var.repère>

rend solidaires les deux repères désignés par les variables figurant dans l'instruction. Cela signifie que si par la suite, la position absolue de l'un d'eux est modifiée, par une instruction d'affectation ou une instruction DEPLACER (§ 4.4), la position absolue de l'autre repère sera automatiquement mise à jour de telle sorte que la position relative des deux repères reste inchangée.

Par plusieurs instructions LIER, on peut créer un graphe quelconque de liaisons entre repères ; par exemple, la séquence d'instructions :

LIER R1 R2 ; LIER R2 R3 ; LIER R4 R3 ;

lie indirectement R1 à R4. Dans la suite, on dira que deux repères sont *liés* s'ils sont liés directement ou indirectement.

On ne peut pas affecter une valeur (position) à une variable de type repère liée à un repère d'état. De plus, il est interdit de lier deux repères d'état.

Dans la suite, nous dirons qu'un repère est *déplaçable* s'il est lié à un repère ROBOT*i* ou s'il est un repère ROBOT*i*.

Remarque : Un repère est donc un objet symbolique ayant une existence propre indépendamment de sa valeur (transformation qui définit sa position).

b) Instruction DELIER

L'instruction

DELIER <var.repère> [<var.repère>]

détruit la liaison directe éventuelle entre deux repères, si deux repères sont spécifiés dans l'instruction, ou toutes les liaisons directes qu'a un repère avec d'autres repères, si un seul repère est spécifié dans l'instruction.

4.4. Contrôle des déplacements du manipulateur

a) Instruction DEPLACER

L'instruction

DEPLACER [IMMEDIAT] <déplacement> [JUSQUA <booléen>][/]

commande un déplacement du manipulateur décrit par <déplacement> (cf. § b). La clause éventuelle "JUSQUA <condition>" commande d'arrêter le déplacement dès que l'expression <booléen> s'évalue à VRAI, même si le manipulateur n'a pas atteint la destination spécifiée par <déplacement>. L'expression <booléen> est appelée *condition d'arrêt* du déplacement.

Exemple : DEPLACER PINCE DE TRANSF JUSQUA FZ > 10.0

Sauf si l'instruction se termine par le symbole "/", l'interpréteur n'attend pas la fin du déplacement pour exécuter les instructions qui suivent une instruction DEPLACER. S'il rencontre une nouvelle instruction DEPLACER, alors qu'un déplacement du même manipulateur est en cours, deux cas sont possibles :

- si l'instruction ne comporte pas la clause IMMEDIAT, l'interpréteur se met en attente jusqu'à la fin de ce déplacement,
- si l'instruction comporte la clause IMMEDIAT, l'interpréteur exécute "immédiatement" le déplacement spécifié par cette instruction, qui se substitue donc au déplacement en cours (celui-ci est définitivement abandonné).

Lorsqu'aucun déplacement n'est en cours, la clause IMMEDIAT est sans effet.

Lors de l'interprétation d'une instruction DEPLACER, les variables déclarées contenues dans <booléen> sont remplacées par leurs valeurs courantes. Ainsi, dans la séquence d'instructions :

```
S := 10.0 ;
DEPLACER PINCE DE TRANSF JUSQUA FZ > S ;
S := 20.0 ;
```

la seconde affectation ne modifie pas la condition d'arrêt du déplacement, qui reste FZ > 10.0.

La commande d'un déplacement impossible provoque une erreur qui s'accompagne de l'immobilisation des manipulateurs et de leurs outils, sauf si l'on a décidé de "récupérer" l'erreur dans une variable booléenne (cf. § 5.4).

Remarque :

- *La variable d'état TEMPS.DEPLi est remise à 0 au début de chaque déplacement du manipulateur i.*

b) Description des déplacements

Un déplacement est défini par :

<déplacement> ::= <var.repère> A <repère>|
<var.repère> A <repère> VIA <repère> {,<repère>}*|
<var.repère> DE <transformation>|
<déplacement enregistré>.

Nous commentons ces cinq constructions ci-dessous :

- *<var.repère>* A *<repère>* définit un déplacement à la fin duquel la position du repère <var.repère> est celle de <repère>. La trajectoire suivie est quelconque : elle est obtenue en lançant les mouvements des degrés liberté concernés simultanément et en les arrêtant simultanément. Le repère <var.repère> doit être déplaçable. L'expression <repère> est évaluée avant que le déplacement ne commence. Cette définition de déplacement est adaptée aux déplacements rapides qui ne présentent pas de risque de collision.
- *<var.repère>* A *<repère>* VIA *<repère>*{,*<repère>*}* définit un déplacement à la fin duquel la position de <var.repère> est celle de l'expression <repère> qui figure immédiatement derrière A. La trajectoire suivie par <var.repère> passe à proximité des positions données par les expressions <repère> qui figurent derrière VIA ; entre deux positions, le déplacement de l'origine de <var.repère> est rectiligne et la rotation de <var.repère> est effectuée à chaque instant autour d'un axe passant par l'origine de ce repère et parallèle à une direction unique (aux transitions aux

voisinages des positions intermédiaires près). Le repère <var.repère> doit être déplaçable. Les expressions <repère> sont toutes évaluées avant que le déplacement ne commence. Cette définition de déplacement est adaptée aux déplacements longs présentant des risques de collisions.

- *<var.repère> DE <transformation>* définit un déplacement tel que la position finale de <var.repère> soit celle de r*t, où r est la position initiale de <var.repère> et t la valeur de <transformation> déterminée avant que le déplacement ne commence. La trajectoire suivie par l'origine de <var.repère> est rectiligne et la rotation de <var.repère> est effectuée à chaque instant autour d'un axe passant par l'origine de ce repère et parallèle à une direction unique. Le repère <var.repère> doit être déplaçable. Cette définition de déplacement est adaptée aux déplacements courts et précis.
- *<déplacement enregistré>* définit un déplacement échantillonné en termes de coordonnées cartésiennes relatives mémorisées dans un fichier sur disque ou disquette identifié par <déplacement enregistré>. Ce déplacement est exécuté relativement à la position courante de ROBOT*i*. Le fichier <déplacement enregistré> peut avoir été construit par un programme distinct de l'interpréteur de LM. Cette définition de déplacement rend également possible une certaine forme de "programmation par l'exemple".

c) *Contrôle de la vitesse du déplacement*

L'instruction

VITESSE [IMMEDIATE] ROBOT*i* <réel>

permet de contrôler *approximativement* la vitesse de déplacement de l'extrémité utile du manipulateur *i*. L'expression <réel> doit s'évaluer à un nombre compris entre 0 (vitesse minimale) et 1 (vitesse maximale).

Une instruction VITESSE ne contenant pas la clause IMMEDIATE modifie un coefficient de vitesse qui est automatiquement initialisé à 1 au début de l'exécution du programme.

Une instruction VITESSE contenant la clause IMMEDIATE réduit la vitesse d'un déplacement en cours. Elle laisse intacte la valeur de ce coefficient pour les déplacements suivants. Si aucun déplacement n'est en cours, elle n'a aucun effet.

Une instruction VITESSE ne contenant pas la clause IMMEDIATE ne modifie pas la vitesse d'un déplacement en cours.

d) Arrêt d'un déplacement

On peut commander explicitement l'arrêt d'un déplacement en cours par l'instruction :

ARRETER DEPLACEMENT ROBOT*i*

Si aucun déplacement n'est en cours, l'instruction est sans effet.

4.5. Contrôle des actions de l'outil

a) Instruction ACTIONNER

L'instruction
ACTIONNER <outil> ROBOT*i* [<expression>{,<expression>}*][JUSQUA <booléen>][/]
commande une action de l'outil du manipulateur *i*. L'entité <outil> est un identificateur désignant cet outil, par exemple : PINCE, TOURNEVIS. Les identificateurs possibles dépendent du matériel sur lequel est implanté le langage. Les paramètres éventuels de l'action sont définis par les expressions <expressions> dont les types dépendent de l'outil ; par exemple, dans le cas d'une pince, il peut y avoir un seul paramètre réel spécifiant l'ouverture de la pince. La clause éventuelle "JUSQUA <booléen>" commande d'arrêter l'action de l'outil dès que l'expression <booléen> s'évalue à VRAI. Cette expression est appelée *condition d'arrêt* de l'action.

Exemples : ACTIONNER PINCE ROBOT1 0.0 JUSQUA PRESSION1+PRESSION2 > 5.0
ACTIONNER TOURNEVIS ROBOT0 JUSQUA MZ > 15.0.

Sauf si l'instruction se termine par le symbole "/", l'interpréteur n'attend pas la fin de l'action de l'outil pour exécuter les instructions qui suivent une instruction ACTIONNER. S'il rencontre ACTIONNER alors qu'une action de l'outil du même robot est en cours, il se met en attente jusqu'à la fin de cette action.

Lors de l'interprétation d'une instruction ACTIONNER, les variables déclarées contenues dans <booléen> sont remplacées par leurs valeurs courantes.

Remarque :

La variable d'état TEMPS.ACTIONi est remise à 0 au début de chaque action du manipulateur i.

b) *Arrêt d'une action*

On peut commander explicitement l'arrêt d'une action en cours par l'instruction :

ARRETER ACTION ROBOTi

Si aucune action n'est en cours, l'instruction est sans effet.

4.6. Procédures

a) *En-tête de procédure*

L'en-tête d'une procédure est de la forme :

PROCEDURE <nom de procédure>[(<nom de paramètre><type de paramètre>
{,<nom de paramètre><type de paramètre>}*)]
[<type du résultat>]

où :

- <nom de procédure> et <nom de paramètre> sont des identificateurs désignant la procédure et ses paramètres,
- <type de paramètre> est le type du paramètre correspondant,
- <type du résultat> est le type du résultat éventuellement retourné par la procédure.

Les paramètres d'une procédure doivent être considérés, dans la procédure, comme des variables déclarées par l'en-tête.

b) *Retour d'une procédure*

L'instruction

RETOUR [<expression>]

termine l'exécution d'une procédure. L'expression <expression> n'apparaît que dans le cas où la procédure retourne un résultat et sa valeur est ce résultat. Elle doit être du type annoncé dans l'en-tête de la procédure.

c) *Appel d'une procédure*

L'appel d'une procédure se fait par

<nom de procédure> [(<expression>{,<expression>}*)]

où <nom de procédure> identifie la procédure appelée. Le nombre des expressions <expression> et leurs types respectifs doivent être ceux des paramètres de la procédure.

L'appel d'une procédure cause l'exécution des instructions de cette procédure. Les arguments sont transmis par adresses. Si l'un d'eux est une variable d'état, on ne doit pas affecter une valeur au paramètre correspondant dans la procédure.

Un appel de procédure peut toujours constituer une instruction. Si la procédure retourne un résultat, celui-ci est perdu.

L'appel d'une procédure qui retourne un résultat peut aussi apparaître dans une expression.

d) *Déclaration de procédure*

L'appel d'une procédure dans une expression requiert que l'on ait déclaré la procédure dans l'unité de programme où a lieu cet appel. La déclaration est de la forme :

<type> PROCEDURE <nom de procédure>{,<nom de procédure>}*

où <type> est le type du résultat retourné par la procédure.

La déclaration d'une procédure doit figurer avec les autres déclarations au début de l'unité de programme concernée.

4.7. Instructions d'entrées-sorties

L'instruction

ECRIRE <sortie>{,<sortie>}* [DANS <fichier>]

où :

<sortie> ::= 'chaîne de caractères'|<expression>

permet

- d'enregistrer des chaînes de caractères et les valeurs d'expressions de types quelconques dans un fichier sur disque ou disquette identifié par <fichier>, si la clause "DANS <fichier>" figure dans l'instruction ;

- d'imprimer des chaînes de caractères et les valeurs d'expressions de types quelconques sur le terminal, si la clause "DANS <fichier>" ne figure pas dans l'instruction.

Un vecteur est exprimé par trois réels (ses composantes) ; une transformation est exprimée par les vecteurs de la rotation et de la translation qui composent la transformation ; un repère r est exprimé comme la transformation définie par TRANSF(STATION,r).

L'instruction

LIRE <variable>{,<variable>}* [DANS <fichier>]

permet de lire les valeurs de variables de types quelconques, soit dans un fichier sur disque ou disquette identifié par <fichier>, soit sur le terminal. Les conventions pour les vecteurs, les repères et les transformations sont les mêmes que ci-dessus.

Dans tous les cas, l'accès des fichiers <fichier> est séquentiel.

5. TRAITEMENTS DIVERS

5.1. Instruction MANUEL

L'instruction

MANUEL

donne à l'utilisateur le contrôle manuel des manipulateurs ROBOTi sur un boîtier ou pupitre de commande. Le transfert de contrôle n'est effectif que lorsque tous les manipulateurs (degrés de liberté et outils) sont immobiles.

L'exécution se poursuit en séquence lorsque l'utilisateur rend le contrôle au calculateur.

5.2. Fonctions capteurs

L'utilisation de capteurs non paramétrés est possible grâce aux variables d'état. Celle de capteurs paramétrés nécessite de faire appel à des fonctions, appelées *fonctions capteurs*. Ces fonctions dépendent de l'implantation

du langage. Par exemple, on peut envisager de disposer d'un système de vision à base de caméra qui localise un objet quelconque posé sur un plan horizontal ; la fonction capteur correspondante peut alors être VISION(nobj), où nobj est un entier qui identifie l'objet à localiser, et s'évaluer à la transformation qui définit la position d'un repère attaché à cet objet.

5.3. Synchronisation avec des processus externes

On peut synchroniser l'exécution d'un programme avec des processus externes, grâce aux variables d'état et à l'instruction :

ENVOYER <réel> SUR VOIE <entier>.

Cette instruction a pour effet d'afficher un signal d'amplitude égale à <réel> sur la voie désignée par <entier> prévue à cet effet. Ce signal peut être récupéré par un autre processus.

5.4. Récupération d'erreurs

Plusieurs erreurs sémantiques peuvent se produire au cours de l'exécution d'un programme (commande d'un déplacement impossible notamment). Elles entraînent normalement l'arrêt immédiat de l'exécution du programme, avec immobilisation des manipulateurs et de leurs outils.

On peut cependant récupérer les erreurs d'un type donné grâce à l'instruction :

RECUPERER ERREUR <entier> DANS <var.booléen>

où <entier> identifie le type d'erreur à récupérer. Si par la suite une erreur de ce type se produit, la valeur VRAI est donnée à la variable <var.booléen> et l'exécution du programme se poursuit.

L'instruction

FIN RECUPERER ERREUR <entier>

termine la récupération des erreurs du type identifié par <entier>.

5.5. Instruction ABORT

L'instruction

ABORT [DUMP]

arrête tout déplacement et toute action en cours et termine l'exécution du programme. Elle peut figurer dans n'importe quelle unité de programme.

Si la clause DUMP figure dans l'instruction, les valeurs courantes de toutes les variables sont imprimées sur le terminal.

6. PERSPECTIVE D'EVOLUTION

Il est envisagé d'ajouter de nouvelles instructions au langage LM dans les mois à venir. Ces compléments porteront principalement sur la description de trajectoires.

Parmi les facilités ajoutées figurera notamment la possibilité de décrire directement des déplacements dits à "compliance active" [13], par exemple :

DEPLACER ROBOT1 DE TRANSF AVEC FZ1 = 5.0

La programmation de tels déplacements avec les instructions présentées dans ce chapitre est possible, mais fastidieuse et peu efficace. Les instructions prévues permettront de définir un déplacement partiellement en termes géométriques et partiellement en termes de forces ou de moments. Elles s'appuieront sur les travaux décrits dans [13].

Une autre facilité sera de permettre la description incomplète de trajectoire. Par exemple, la saisie d'un objet de révolution par une pince peut, pour certaines prises, être réalisée sans qu'il soit nécessaire d'imposer l'orientation de la pince autour de son axe. Pour cela, il est prévu d'utiliser certains des résultats présentés dans [6].

CHAPITRE 3

ANALYSE FONCTIONNELLE DE L'INTERPRETEUR DU LANGAGE LM

Dans ce chapitre, nous présentons une analyse fonctionnelle de l'interpréteur du langage LM. Nous nous intéressons particulièrement à quatre points :

- la représentation des données et les opérations sur ces données,
- la gestion des liaisons entre repères,
- les calculs des déplacements,
- la structure de l'interpréteur.

Nous ne présentons pas de méthode pour traiter les données transmises par les capteurs. Nous supposons que chaque capteur possède son propre processeur de mise en forme, de telle sorte qu'à sa sortie, l'interpréteur trouve directement les données dont il a besoin, par exemple :
- position d'un repère attaché à un objet (capteur visuel),
- composante de force suivant une direction (capteur de force),
- pression moyenne sur les mors d'une pince (capteur du type peau sensible).

1. OPERATIONS SUR LES VECTEURS, LES TRANSFORMATIONS, LES REPERES

1.1. Représentation fonctionnelle des vecteurs, des transformations et des repères

a) *Les vecteurs*

Un vecteur peut être représenté par une matrice 3 × 1 :

$$\begin{Vmatrix} x \\ y \\ z \end{Vmatrix}$$

où x, y et z sont les composantes du vecteur.

Dans la suite, nous noterons [v] la représentation du vecteur v et $[v]_1$, $[v]_2$, $[v]_3$ ses composantes x, y, z.

b) *Les transformations*

Soit t une transformation dont l'application dans un repère r réalise :
- d'abord une rotation telle que

$$\begin{Vmatrix} a1 \\ a2 \\ a3 \end{Vmatrix} \qquad \begin{Vmatrix} b1 \\ b2 \\ b3 \end{Vmatrix} \qquad \begin{Vmatrix} c1 \\ c2 \\ c3 \end{Vmatrix}$$

représentent dans r les transformées des vecteurs unitaires portés par les axes de r,

- puis une translation suivant un vecteur dont les composantes dans r sont p1, p2 et p3.

Cette transformation peut être représentée en coordonnées homogènes [21] par une matrice 4 × 4 de la forme :

$$\begin{Vmatrix} a1 & b1 & c1 & p1 \\ a2 & b2 & c2 & p2 \\ a3 & b3 & c3 & p3 \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

On note que $\begin{Vmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{Vmatrix}$ est une matrice orthonormale.

Dans la suite, nous noterons [t] la représentation de la transformation t, et $[t]_{i,j}$ (i=1 à 4 ; j=1 à 4) l'élément de la ligne i et de la colonne j de cette matrice.

Remarque :

La représentation d'une transformation en coordonnées homogènes est fréquemment utilisée car elle possède plusieurs avantages. En particulier, les règles opératoires pour composer deux transformations et pour inverser une transformation sont très simples (cf. § 1.3). Mais la représentation de la rotation par 9 paramètres est très redondante. Cette redondance résulte en une occupation de mémoire inutile. Elle peut aussi conduire à des incohérences numériques. Les quaternions [25][20] constituent une autre représentation qui évite ces inconvénients.

c) Les repères

Un repère peut être représenté par la même matrice 4 × 4 que sa position absolue, i.e. la transformation TRANSF(STATION,r) (cf. § 2.2, chapitre 2). Cette représentation sera légèrement compliquée au § 2 pour permettre la gestion des liaisons entre repères.

La représentation de la constante STATION est donc la matrice identité 4×4.

Dans la suite, on notera [r] la représentation du repère r, et $[r]_{i,j}$ (i=1 à 4, j=1 à 4) l'élément de ligne i et de la colonne j de cette représentation.

1.2. Opérations sur les vecteurs

- VECTEUR (x,y,z) $\rightarrow \begin{Vmatrix} x \\ y \\ z \end{Vmatrix}$

- $X(v) \rightarrow [v]_1$; $Y(v) \rightarrow [v]_2$; $Z(v) \rightarrow [v]_3$

- v1 + v2 $\rightarrow \begin{Vmatrix} [v1]_1 + [v2]_1 \\ [v1]_2 + [v2]_2 \\ [v1]_3 + [v2]_3 \end{Vmatrix}$

,

- v1 - v2 $\rightarrow \begin{Vmatrix} [v1]_1 - [v2]_1 \\ [v1]_2 - [v2]_2 \\ [v1]_3 - [v2]_3 \end{Vmatrix}$

- - v $\rightarrow \begin{Vmatrix} -[v]_1 \\ -[v]_2 \\ -[v]_3 \end{Vmatrix}$

- r * v $\begin{Vmatrix} r[v]_1 \\ r[v]_2 \\ r[v]_3 \end{Vmatrix}$

- PSCAL(v1,v2) $\rightarrow \sum_{i=1}^{3} [v1]_i \, [v2]_i$

- PVECTO(v1,v2) $\rightarrow \begin{Vmatrix} [v1]_2[v2]_3 - [v1]_3[v2]_2 \\ [v2]_1[v1]_3 - [v1]_1[v2]_3 \\ [v1]_1[v2]_2 - [v1]_2[v2]_1 \end{Vmatrix}$

- LONG (V) → $\left(\sum_{i=1}^{3} [v]_i^2 \right)^{1/2}$

- t * v → $\left\| \begin{array}{l} \sum_{i=1}^{3} [t]_{1,i}\ [v]_i \\ \sum_{i=1}^{3} [t]_{2,i}\ [v]_i \\ \sum_{i=1}^{3} [t]_{3,i}\ [v]_i \end{array} \right\|$

1.3. Opérations sur les transformations

- ROTATION (v,x) →

$$\left\| \begin{array}{cccc} u_1^2+(1-u_1^2)\cos x & u_1u_2(1-\cos x)-u_3\sin x & u_1u_3(1-\cos x)+u_2\sin x & 0 \\ u_1u_2(1-\cos x)+u_3\sin x & u_2^2+(1-u_2^2)\cos x & u_2u_3(1-\cos x)-u_1\sin x & 0 \\ u_1u_3(1-\cos x)+u_2\sin x & u_2u_3(1-\cos x)+u_1\sin x & u_3^2+(1-u_3^2)\cos x & 0 \\ 0 & 0 & 0 & 1 \end{array} \right\|$$

où :

$$u_i = \frac{[v]_i}{\text{LONG}(v)} \ , \ i = 1,2,3.$$

- TRANSLATION(v,x) → $\left\| \begin{array}{cccc} 1 & 0 & 0 & u_1 \\ 0 & 1 & 0 & u_2 \\ 0 & 0 & 1 & u_3 \\ 0 & 0 & 0 & 1 \end{array} \right\|$

où :

$$u_i = x \frac{[v]_i}{\text{LONG}(v)} \ , \ i = 1,2,3.$$

- $t_1 * t_2 \rightarrow [t_1][t_2]$ (produit matriciel)

- VECT.ROT(t) → $\left\| \begin{array}{c} \alpha \\ \beta \\ \gamma \end{array} \right\|$

où :

$\alpha = \frac{\theta}{h}([t]_{3,2}-[t]_{2,3})$, $\beta = \frac{\theta}{h}([t]_{1,3}-[t]_{3,1})$ et $\gamma = \frac{\theta}{h}([t]_{2,1}-[t]_{1,2})$

avec :

$$\theta = \text{ARCOS}\ \frac{\sum_{i=1}^{3}[t]_{i,j}-1}{2}$$

$$h = 2\sin\theta$$

[si $\theta = \pi$:

$$\alpha = \frac{\pi}{\sqrt{2}}\left(\frac{[t]_{2,1}[t]_{3,1}}{[t]_{3,2}}\right)^{1/2}$$

$$\beta = \frac{\pi}{\sqrt{2}}\left(\frac{[t]_{2,1}[t]_{3,2}}{[t]_{3,1}}\right)^{1/2}$$

$$\gamma = \frac{\pi}{\sqrt{2}}\left(\frac{[t]_{3,2}[t]_{3,1}}{[t]_{2,1}}\right)^{1/2}$$]

- VECT.TRSL(τ) → $\begin{Vmatrix} [t]_{1,4} \\ [t]_{2,4} \\ [t]_{3,4} \end{Vmatrix}$

- INV(t) → $[t]^{-1} = \begin{Vmatrix} [t]_{1,1} & [t]_{2,1} & [t]_{3,1} & -\sum_{i=1}^{3}[t]_{i,1}[t]_{i,4} \\ [t]_{1,2} & [t]_{2,2} & [t]_{3,2} & -\sum_{i=1}^{3}[t]_{i,2}[t]_{i,4} \\ [t]_{1,3} & [t]_{2,3} & [t]_{3,3} & -\sum_{i=1}^{3}[t]_{i,3}[t]_{i,4} \\ 0 & 0 & 0 & 1 \end{Vmatrix}$

(car la matrice de rotation 3×3 est orthonormale).

$$- \text{ROT}(t) \rightarrow \begin{Vmatrix} [t]_{1,1} & [t]_{1,2} & [t]_{1,3} & 0 \\ [t]_{2,1} & [t]_{2,2} & [t]_{2,3} & 0 \\ [t]_{3,1} & [t]_{3,2} & [t]_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

$$- \text{TRSL} \rightarrow \begin{Vmatrix} 1 & 0 & 0 & [t]_{1,4} \\ 0 & 1 & 0 & [t]_{2,4} \\ 0 & 0 & 1 & [t]_{3,4} \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

1.4. Opérations sur les repères

- t * r → [t][r]
- r * t → [r][t]
- TRANSF(r1,r2) → $[r1]^{-1}$ [r2]
- DISTANCE(r1,r2) → LONG(VECT.TRSL(TRANSF(r1,r2)))
- ANGLE(r1,r2) → LONG(VECT.ROT(TRANSF(r1,r2))).

2. GESTION DES LIAISONS ENTRE REPERES

2.1. Représentation des repères

L'instruction LIER permet de rendre plusieurs repères solidaires (cf. § 4.3, chapitre 2). Nous appelons *solide* un ensemble de repères solidaires entre eux, c'est-à-dire une composante connexe du graphe ayant les variables de type repère pour sommets et les liaisons entre ces variables pour arêtes.

La modification de la position absolue d'un repère d'un solide S doit entraîner une mise à jour appropriée des positions absolues des autres repères contenus dans S. Leur mise à jour explicite pourrait cependant être coûteuse et inutile. C'est pourquoi nous avons adopté une représentation des repères différente de celle présentée au § B.1.3.

Dans tout solide S, on considère un repère rp que l'on appelle *repère principal* de S. A chaque repère r de S (y compris rp), on associe un indicateur, noté I(r), qui a le nom rp pour valeur, et une matrice, notée {r}, qui représente la position de r si r = rp et la position de r relative à rp, soit $[rp]^{-1}[r]$, si r ≠ rp.

Soit r un repère quelconque. On obtient la matrice [r] utilisée au § 1 et représentant la position absolue de r, en exécutant l'algorithme :

$$\textit{si } I(r) = r \textit{ alors } [r] \leftarrow \{r\}$$
$$\textit{sinon } [r] \leftarrow \{I(r)\}\{r\}.$$

Si un solide S contient un repère d'état, celui-ci est toujours pris pour repère principal. De cette façon, pour un repère quelconque r de S, l'algorithme précédent permet d'obtenir à chaque instant la matrice [r] courante.

2.2. Mise à jour des représentations des repères

a) *Initiations au chargement*

Pour chaque variable r de type repère, on effectue $I(r) \leftarrow r$ au chargement d'un programme.

b) *Affectation d'une valeur à un repère*

Soit r une variable de type repère appartenant à un solide S. L'affectation d'une nouvelle valeur à r doit entraîner une mise à jour appropriée des positions des autres repères appartenant à S. Compte tenu de la représentation décrite au § 2.1, il suffit de modifier la position du repère principal de S. Ainsi, l'algorithme suivant permet d'affecter la valeur $[r_o]$ au repère r

$$\textit{si } I(r) = r \textit{ alors } \{r\} \leftarrow [r_o]$$
$$\textit{sinon } \{I(r)\} \leftarrow [r_o]\{r\}^{-1}.$$

Le retour d'une procédure peut aussi nécessiter des mises à jour. Celles-ci seront décrites au § 2.5.

2.3. Création d'une liaison entre deux repères

Soient r1 et r2 deux variables de type repère qui appartiennent respectivement aux solides S1 et S2. Si S1 et S2 sont distincts, l'instruction "LIER r1 r2" a pour effet de remplacer S1 et S2 par le solide S = S1 ∪ S2. L'algorithme suivant permet d'effectuer les traitements nécessaires :

```
rp1 ← I(r1)
rp2 ← I(r2)
/* on choisit (par exemple) rp1 comme repère principal de S */
t ← {rp1}^-1 {rp2}
pour tout r dans S2 faire
        si r ≠ rp2 alors {r} = t{r}
                   sinon {r} ← t
        I(r) ← rp1
finfaire
```

Le choix du repère principal de S est imposé si rp1 ou rp2 est un repère d'état (cf. § 2.1). Dans le cas contraire, on peut choisir rpi (i=1 ou 2) tel que Card(Si) > Card(Sj) (j=1 ou 2 ; j ≠ i).

2.4. Destruction de liaisons entre repères

Soient r1 et r2 deux variables de type repère directement liées entre elles ; r1 et r2 appartiennent donc au même solide S. L'instruction "DELIER r1 r2" a éventuellement pour effet de casser S en deux solides S1 et S2 auxquels appartiennent respectivement r1 et r2. On exécute alors l'algorithme suivant :

```
si I(r1) ∈ S1 alors i ← 2 sinon i ← 1
/* on choisit dans Si un repère rpi qui sera le repère principal de Si */
t ← {rpi}^-1
pour tout r ≠ rpi dans Si faire
                        {r} ← t{r}
                        I(r) ← rpi
                    finfaire
{rpi} ← [rpi]
I(rpi) ← rpi.
```

L'instruction "DELIER r" peut avoir pour effet de partager un solide S en plusieurs solides. On exécute alors un algorithme analogue au précédent.

2.5. Retour d'une procédure

Au retour d'une procédure, deux types de modifications doivent être envisagés.

a) Si un même solide S contient d'une part des repères locaux déclarés dans la procédure, et d'autre part des repères globaux ou passés en paramètres, et si le repère principal de S est un repère local déclaré dans la procédure, il est nécessaire de changer le repère principal de S.

Cette modification peut être effectuée par l'algorithme suivant, où L note l'ensemble des repères locaux à la procédure :

pour tout solide S dont le repère principal rp ∈ L faire
si S contient un repère r ∉ L alors
/ on choisit r comme nouveau repère principal de S */*
$t \leftarrow \{r\}^{-1}$
pour tout r' ≠ r dans S-L faire
$\{r'\} \leftarrow t\{r\}$
$I(r') \leftarrow r$
finfaire
$\{r\} \leftarrow [r]$
$I\{r\} \leftarrow r$
finfaire

b) Si dans un solide S, deux repères r1 et r2 ∉ L sont liés indirectement par l'intermédiaire d'un repère r ∈ L, par exemple :

$$r_1 \text{ —— } r \text{ —— } r_2$$

il est nécessaire de conserver la liaison entre r1 et r2 au retour de la procédure, bien que le repère r disparaisse. La liaison devient indestructible, ce que l'on note :

$$r_1 \text{ ═══ } r_2$$

Cette mise à jour peut être effectuée par l'algorithme suivant :

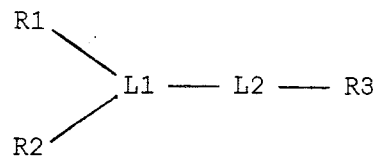
pour tout solide S tel que S ∩ L ≠ ∅ faire
pour tout r ∈ S ∩ L faire
R ← ensemble des repères liés directement à r
créer une liaison marquée indestructible entre tout couple de repères de R
détruire r et les liaisons adjacentes à r
finfaire
finfaire

Exemple :

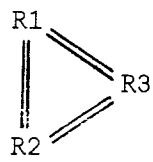
Soit la procédure LM :

```
PROCEDURE (R1 REPERE, R2 REPERE, R3 REPERE)
REPERE L1, L2 ;
DEBUT
     LIER R1 L1 ;
     LIER R2 L1 ;
     LIER L1 L2 ;
     LIER R3 L2 ;
     RETOUR
FIN
```

Avant l'exécution de l'instruction RETOUR, on a le solide S suivant :



Après le retour, ce solide devient :



2.6. Formation des solides

On peut choisir soit d'avoir une représentation explicite des solides, soit de former les solides dont on a besoin aux moments où on en a besoin. Dans le second cas, compte tenu de ce que peu de solides devraient contenir plus d'une dizaine de repères, on peut se contenter de l'algorithme simple suivant pour former le solide S contenant un repère r donné :

```
initialiser pile à pile vide
mettre r sur pile
index ← 1
tant que index ≤ hauteur de pile faire
        r ← pile(index)
        pour tout repère r' lié directement à r faire
                si r' ∉ pile alors mettre r' sur pile
        finfaire
        index ← index+1
finfaire.
```

Les éléments de la pile à la fin de l'exécution de cet algorithme sont ceux de S.

Pour déterminer si deux repères r1 et r2 appartiennent à un même solide, on peut de façon analogue exécuter :

```
initialiser pile à pile vide
mettre r1 sur pile
index ← 1
mêmesolide ← faux
tant que index ≤ hauteur de pile et ¬mêmesolide faire
        r ← pile (index)
        pour tout repère r' lié directement à r faire
            si r' = r2 alors mêmesolide ← vrai
            si mêmesolide alors exitfaire
            si r' ∉ pile alors mettre r' sur pile
        finfaire
        index ← index+1
finfaire.
```

3. CALCULS DES DEPLACEMENTS

3.1. Présentation des calculs

Il s'agit de déterminer les valeurs des coordonnées propres d'un manipulateur, qui doivent être successivement affichées en entrée des asservissements, pour réaliser un déplacement défini par une construction <déplacement> ::=

<var.repère> A <repère>| (type 1)
<var.repère> A <repère> VIA <repère> {,<repère>}*| (type 2)
<var.repère> DE <transformation>| (type 3)
<déplacement enregistré> (type 4)

(cf. § 4.4.b, chapitre 2).

On peut distinguer trois parties dans les calculs :

(1) L'échantillonnage de la trajectoire dans l'espace cartésien, ou *espace de la tâche* : on détermine les positions successives du repère à déplacer, la période d'échantillonnage étant de l'ordre de 100 ms (ce calcul ne doit être effectué que pour les déplacements des types 2 et 3).

(2) Le changement de coordonnées permettant de passer de l'espace de la tâche à l'espace des coordonnées propres du manipulateur, ou *espace du manipulateur*.

(3) L'échantillonnage dans l'espace du manipulateur : on détermine les valeurs des coordonnées à afficher successivement en entrée des asservissements, la période d'échantillonnage étant de l'ordre de 10 ms.

Dans les paragraphes suivants, nous analysons d'abord le changement de coordonnées, puis nous présentons les calculs d'échantillonnage pour chaque type de déplacement.

Note : Bien que cela ne soit pas explicité, il est facile d'utiliser les calculs présentés dans la suite pour piloter des asservissements ayant des entrées de position et de vitesse (cas des asservissements en mode dynamique).

3.2. Le changement de coordonnées

Pour tous les déplacements, il est nécessaire de faire appel à une procédure de changement de coordonnées. Cette procédure permet de passer de la matrice 4 × 4 représentant une position dans STATION le repère r à déplacer au vecteur des N coordonnées propres du manipulateur concerné.

Ce passage comporte deux parties :

(1) Soient p_1 et p_2 les matrices qui représentent les positions du repère r relativement à STATION et à ROBOTi respectivement (r est soit le repère ROBOTi, auquel cas la matrice p_2 est la matrice identité, soit un repère lié à ROBOTi). Soient p la matrice qui représente la position de ROBOTi relativement à un repère attaché au bâti fixe du manipulateur concerné et b la matrice qui représente la position de ce repère dans STATION. Au moment du changement de coordonnées, r1, r2 et b sont connus (b est déterminé par un calibrage du manipulateur) ; il s'agit de calculer p.

On a : $b.p.p_2 = p_1$

d'où : $p = b^{-1}.p_1.p_2^{-1}$

(2) La formule précédente permet de donner une valeur numérique aux coefficients de la matrice p. Ces coefficients peuvent aussi être exprimés analytiquement en fonction des coordonnées propres du manipulateur. Les expressions obtenues dépendent de la structure mécanique et géométrique de ce manipulateur. Les valeurs des coordonnées propres peuvent, dans certains cas, être extraites de la comparaison des formes analytique et numérique de p.

Toutefois, lorsque N=6, on utilise souvent l'approximation $\Delta\Theta = J^{-1}\Delta X$ [2], où :

- X est le vecteur des coordonnées de l'origine de ROBOTi et des trois angles d'Euler de ce repère dans le repère attaché au bâti du manipulateur ; on passe facilement de p à X et réciproquement ;
- Θ est le vecteur des 6 coordonnées propres du manipulateur ;
- J est le Jacobien de l'application f : $\Theta \rightarrow X$ que l'on sait toujours exprimer analytiquement.

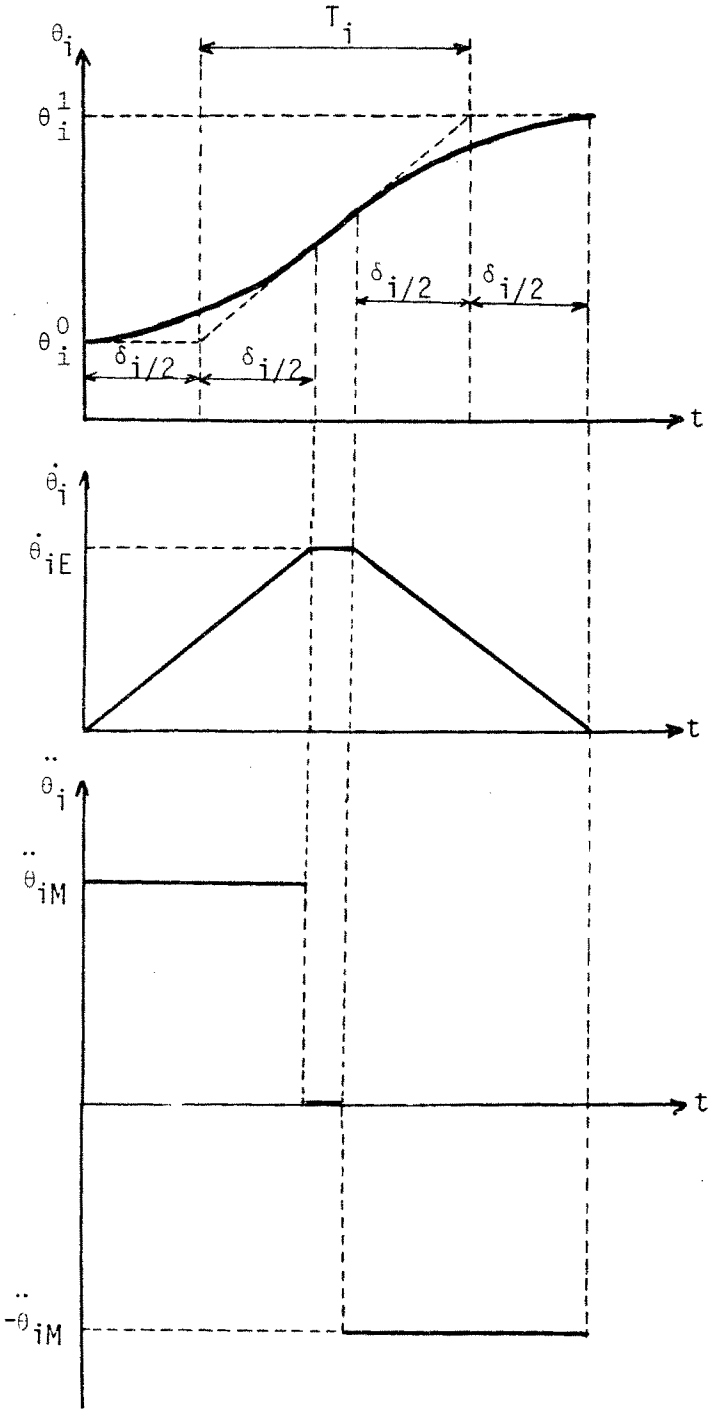

T_i
θ_i
θ_i^1
θ_i^0
$\delta_{i/2}$
$\delta_{i/2}$
$\delta_{i/2}$
$\delta_{i/2}$
t
$\dot{\theta}_i$
$\dot{\theta}_{iE}$
t
$\ddot{\theta}_i$
$\ddot{\theta}_{iM}$
t
$-\ddot{\theta}_{iM}$

Figure 1

Plusieurs méthodes existent pour déterminer J^{-1} [1][20][28].

La formule $\Delta\Theta = J^{-1}\Delta X$ requiert que J soit inversible, ce qui n'est en général pas le cas pour un petit nombre de positions singulières. A ces positions, on peut essayer de chercher les mineurs inversibles de J [14]. Une méthode plus générale utilisant les pseudo-inverses a été récemment développée [6] et facilite la résolution de cette difficulté.

Une autre fonction de la procédure de changement de coordonnées est de vérifier que l'on ne dépasse pas les butées associées à chaque degré de liberté.

3.3. Calcul des déplacements du premier type

Il s'agit de déplacer un repère d'une position initiale à une position finale. La trajectoire suivie est indifférente. Les évolutions des degrés de liberté concernés doivent être synchronisés (démarrages et arrêts simultanés).

Soient θ^0 et θ^1 les valeurs des N coordonnées propres du manipulateur correspondant aux positions initiale et finale du déplacement. Le calcul du déplacement consiste à faire varier chaque coordonnée θ_i (i=1 à N) de θ_i^0 à θ_i^1 linéairement en fonction du temps, sauf aux extrémités où l'on prévoit des transitions à accélération constante, de façon à ce que les vitesses soient continues. La figure 1 illustre la variation de θ_i en fonction du temps.

Nous supposons que pour un manipulateur donné, nous connaissons les valeurs absolues des vitesses et accélérations maximales pour l'évolution de chaque degré de liberté. Soient $\dot{\theta}_{iM}$ et $\ddot{\theta}_{iM}$ ces valeurs affectées du signe de $\theta_i^1-\theta_i^0$. Dans les calculs qui suivent, nous prenons $\dot{\theta}_{iE} = k.\dot{\theta}_{iM}$, où k est le coefficient de vitesse courant (cf. § 4.4.c, chapitre 2).

La durée δ_i de chaque transition est donnée par $\delta_i = \frac{\dot{\theta}_{iE}}{\ddot{\theta}_{iM}}$. Elle correspond au temps mis pour atteindre la vitesse $\dot{\theta}_{iE}$ à l'accélération constante $\ddot{\theta}_{iM}$.

Nous posons $T_i = \frac{\theta_i^1 - \theta_i^0}{\dot{\theta}_{iE}}$. Si $T_i \geq \delta_i$, $\theta_i(t)$ s'exprime de la façon suivante :

- pour $t \in [0,\delta_i]$: $\theta_i(t) = \theta_i^0 + \frac{1}{2}\ddot{\theta}_{iM} t^2$ (1)

- pour $t \in [\delta_i,T_i]$: $\theta_i(t) = \theta_i^0 + \dot{\theta}_{iE}(t - \frac{\delta_i}{2})$ (2)

- pour $t \in [T_i,T_i+\delta_i]$: $\theta_i(t) = \theta_i^0 + \dot{\theta}_{iE}(t - \frac{\delta_i}{2}) - \frac{1}{2}\ddot{\theta}_{iM}(t-T_i)^2$ (3)

Si $T_i < \delta_i$, on ne peut pas atteindre la vitesse $\dot{\theta}_{iE}$. Nous posons alors $\eta_i = \left(\sqrt{\frac{\theta_i^1 - \theta_i^0}{\ddot{\theta}_{iM}}}\right)^{1/2}$. η_i est le temps mis pour effectuer la moitié de la trajectoire. $\theta_i(t)$ s'exprime de la façon suivante :

- pour $t \in [0,\eta_i]$: $\theta_i(t) = \theta_i^0 + \frac{1}{2}\ddot{\theta}_{iM} t^2$ (4)

- pour $t \in [\eta_i,2\eta_i]$: $\theta_i(t) = \frac{\theta_i^0 + \theta_i^1}{2} + \eta_i \ddot{\theta}_{iM}(t-\eta_i) - \frac{1}{2}\ddot{\theta}_{iM}(t-\eta_i)^2$ (5)

Soit λ_i la durée d'évolution du $i^{ème}$ degré de liberté. On a $\lambda_i = T_i+\delta_i$ ou $\lambda_i = 2\eta_i$. Soit $\lambda_j = \underset{i\in[1,N]}{\text{Max}}(\lambda_i)$. Pour synchroniser les évolutions des N degrés de liberté, il suffit de faire en sorte que la durée d'évolution de chaque degré soit λ_j. Une équation (1), (2), (3), (4) ou (5) de la forme :

$$\text{pour } t \in [a_i,b_i] : \theta_i(t) = f_i(t)$$

devient alors

$$\text{pour } t \in [\alpha_i a_i, \alpha_i b_i] : \theta_i(t) = f_i(\frac{t}{\alpha_i})$$

où $\alpha_i = \frac{\lambda_j}{\lambda_i}$.

Compte tenu de la simplicité des calculs à effectuer à chaque pas, on peut réaliser l'échantillonnage des $\theta_i(t)$ au fur et à mesure des déplacements.

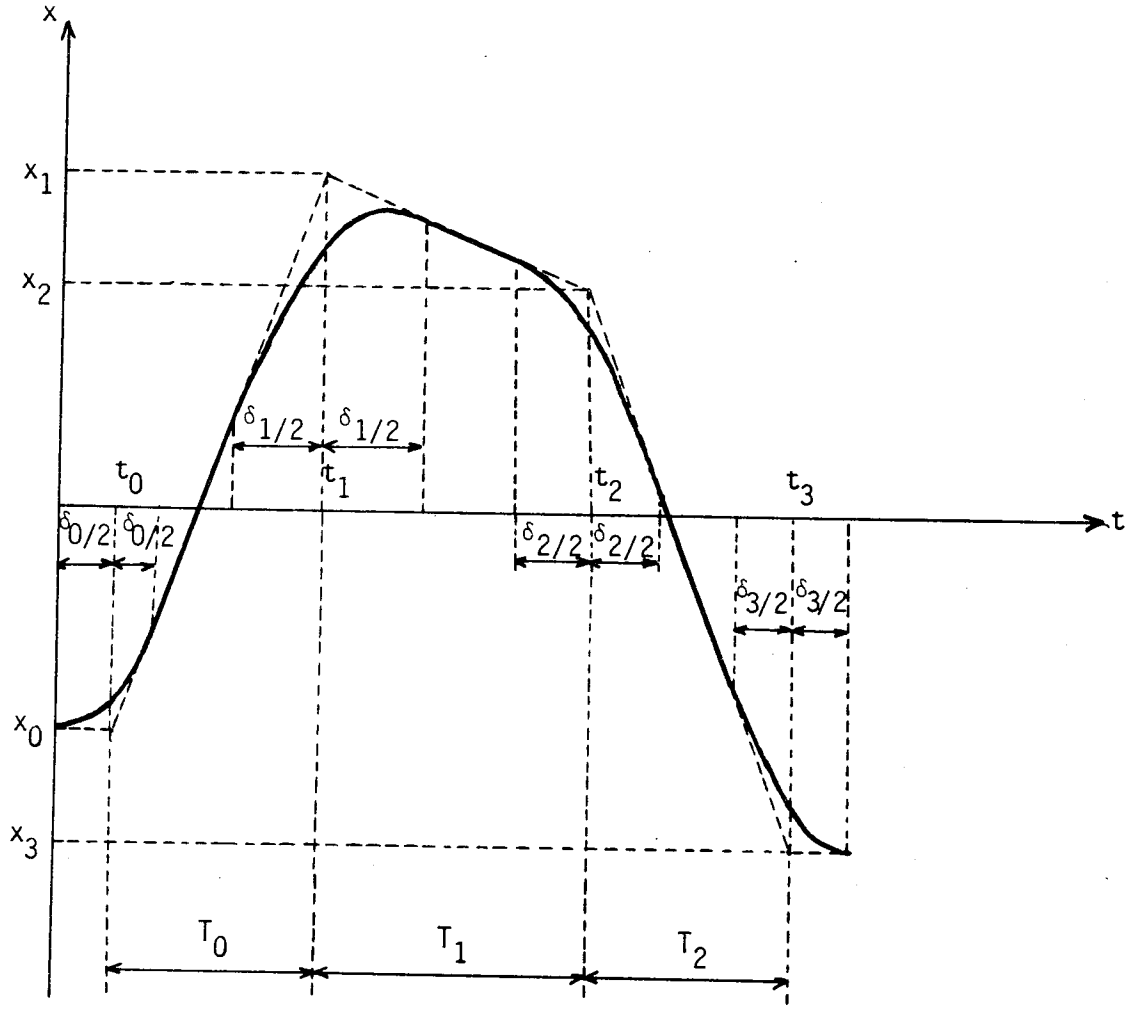


Figure 2

3.4. Calcul des déplacements du deuxième type

Il s'agit de déplacer un repère entre deux positions suivant une trajectoire passant à proximité de positions intermédiaires. Nous notons $P_0, P_1, \ldots, P_z$ les matrices 4×4 qui définissent ces positions ; P_0 et P_z définissent respectivement les positions initiale et finale du déplacement.

Le calcul de ce déplacement comporte deux phases d'échantillonnage : la première dans l'espace de la tâche et la seconde dans l'espace du manipulateur (cf. § 3.1).

3.4.1. Echantillonnage dans l'espace de la tâche [17][25]

Soit p(t) la matrice définissant la position du repère r à déplacer en fonction du temps. Nous écrivons $p(t) = \tau(t).\rho(t)$, où $\tau(t)$ et $\rho(t)$ sont les matrices qui définissent respectivement la translation et la rotation de r relativement au repère STATION. Nous présentons séparément les calculs de $\tau(t)$ et de $\rho(t)$.

a) Calcul de $\tau(t)$

La matrice de $\tau(t)$ est de la forme

$$\begin{Vmatrix} 1 & 0 & 0 & x(t) \\ 0 & 1 & 0 & y(t) \\ 0 & 0 & 1 & z(t) \\ 0 & 0 & 0 & 1 \end{Vmatrix}$$

où x(t), y(t) et z(t) sont les coordonnées de l'origine de r dans STATION pendant le déplacement. Le calcul de $\tau(t)$ consiste à faire varier chacune d'elles linéairement en fonction du temps, sauf aux extrémités et au voisinage des positions intermédiaires du déplacement où l'on prévoit des transitions à accélération constante, de façon à ce que les vitesses soient continues. La figure 2 illustre la variation de x(t) pour z = 3.

Nous notons (cf. figure 2) :

- τ_k la matrice de la translation de p_k dans STATION (k = 0 à z),
- t_k l'instant théorique du passage de r par p_k, si l'origine de r se déplaçait à vitesse constante sur chaque segment $[p_j,p_{j+1}]$ (j = 0 à z-1) et passait instantanément d'un segment à un autre (k = 0 à z),
- $T_k = t_{k+1}-t_k$ (k = 0 à z-1),
- δ_k la durée de la transition au voisinage de p_k (k = 0 à z),
- $\vec{u}_k$ le vecteur unitaire qui définit la direction et le sens de la translation représentée par $\tau_k^{-1}.\tau_{k+1}$ (k = 0 à z-1),
- d_k la longueur de cette translation,
- translation $(\vec{u},d)$ la matrice de la translation de longueur d suivant le vecteur unitaire $\vec{u}$.

Les grandeurs $\vec{u}_k$ et d_k sont déterminées à l'aide de calculs présentés au § 1. Les durées δ_k seront calculées plus loin. Les durées T_k (qui, pour une origine des temps donnée, fixent les instants t_k) seront déterminées au § c.

La matrice $\tau(t)$ s'exprime alors de la façon suivante :

- pour $t \in [t_k + \frac{\delta_k}{2},\ t_{k+1} - \frac{\delta_{k+1}}{2}]$ (k = 0 à z-1) :

$$\tau(t) = \tau_k.\text{translation}\ (\vec{u}_k,\ d_k\ \frac{t-t_k}{T_k}) \qquad (6)$$

- pour $t \in [t_k - \frac{\delta_k}{2},\ t_k + \frac{\delta_k}{2}]$ (k = 0 à z) :

$$\tau(t) = \tau_k.\text{translation}(-\vec{u}_{k-1},d_{k-1}\ \frac{(t-t_k-\delta_{k/2})^2}{2\delta_k T_{k-1}}).\text{translation}(\vec{u}_k,d_k\ \frac{(t-t_k+\delta_{k/2})^2}{2\delta_k T_k}) \qquad (7)$$

avec $d_{-1} = d_z = 0$.

La formule (7) montre que la transition au voisinage de p_k est obtenue en combinant deux accélérations :

- l'accélération $-\frac{d_{k-1}}{\delta_k T_{k-1}}\vec{u}_{k-1}$ qui accélère le déplacement le long de $[p_{k-1},p_k]$ de la vitesse $\frac{d_{k-1}}{T_{k-1}}\vec{u}_{k-1}$ (instant $t_k - \frac{\delta_k}{2}$) à la vitesse nulle (instant $t_k + \frac{\delta_k}{2}$),

- l'accélération $\frac{d_k}{k T_k} \vec{u}_k$ qui accélère le déplacement le long de $[p_k, p_{k+1}]$ de la vitesse nulle (instant $t_k - \frac{\delta_k}{2}$) à la vitesse $\frac{d_k}{T_k} \vec{u}_k$ (instant $t_k + \frac{\delta_k}{2}$).

L'accélération résultante est $\vec{\Gamma}_V = \frac{1}{\delta_k} [\frac{d_k}{T_k} \vec{u}_k - \frac{d_{k-1}}{T_{k-1}} \vec{u}_{k-1}]$. En supposant que, pour un manipulateur donné, nous connaissons $\Gamma_V = |\vec{\Gamma}_V|$, δ_k est donné par :

$$\delta_k = \frac{1}{\Gamma_V} |\frac{d_k}{T_k} \vec{u}_k - \frac{d_{k-1}}{T_{k-1}} \vec{u}_{k-1}| \qquad (8)$$

b) Calcul de $\rho(t)$

En nous inspirant des formules (6) et (7), nous exprimons $\rho(t)$ de la façon suivante :

- pour $t \in [t_k + \frac{\delta'_k}{2}, t_{k+1} - \frac{\delta'_{k+1}}{2}]$ $(k = 0 \text{ à } z-1)$:

$$\rho(t) = \rho_k . \text{rotation}(\vec{v}_k, \psi_k \frac{t-t_k}{T_k}) \qquad (9)$$

- pour $t \in [t_k - \frac{\delta'_k}{2}, t_k + \frac{\delta'_k}{2}]$ $(k = 0 \text{ à } z)$:

$$\rho(t) = \rho_k . \text{rotation}(-\vec{v}_{k-1}, \psi_{k-1} \frac{(t-t_k-\delta'_{k/2})^2}{2\delta'_k T_{k-1}}) . \text{rotation}(\vec{v}_k, \psi_k \frac{(t-t_k+\delta'_{k/2})^2}{2\delta'_k T_k}) \qquad (10)$$

avec $\psi_{-1} = \psi_z = 0$.

Dans ces formules, rotation$(\vec{v},\psi)$ note la matrice de la rotation d'angle autour de l'axe qui passe par l'origine de r, et dont l'orientation et le sens sont ceux du vecteur unitaire $\vec{v}$. Ainsi, la formule (9) traduit une rotation du repère r autour d'un axe fixe défini par $\vec{v}_k$; l'angle de cette rotation est une fonction linéaire du temps. Le vecteur $\vec{v}_k$ est le vecteur unitaire qui définit l'axe de la rotation représentée par la matrice $\rho_k^{-1} \rho_{k+1}$ $(k = 0 \text{ à } z-1)$; ψ_k est l'angle de cette rotation. Les grandeurs $\vec{v}_k$ et ψ_k sont déterminées à l'aide de calculs présentés au § 1.

La formule (10) correspond à une transition au voisinage de p_k de durée δ'_k. Cette transition combine deux accélérations angulaires,

$-\frac{\psi_{k-1}}{\delta'_k T_{k-1}} \vec{v}_{k-1}$ et $\frac{\psi_k}{\delta'_k T_k} \vec{v}_k$, qui font passer la vitesse angulaire $\frac{\psi_{k-1}}{T_{k-1}} \vec{v}_{k-1}$ à $\frac{\psi_k}{T_k} \vec{v}_k$ sans discontinuité. L'accélération angulaire résultante $\vec{\Gamma}_\Omega$ n'est pas rigoureusement constante (car l'axe de rotation n'est pas fixe), sauf si $\vec{v}_{k-1}$ et $\vec{v}_k$ sont parallèles, ou si ψ_{k-1} ou ψ_k est nul. En supposant que, pour un manipulateur donné, nous connaissons $\Gamma_\Omega = |\vec{\Gamma}_\Omega|$, nous pouvons calculer δ'_k par :

$$\delta'_k = \frac{1}{\Gamma_\Omega} \left(\frac{\psi_{k-1}}{T_{k-1}} + \frac{\psi_k}{T_k}\right) \qquad (11)$$

c) *Synthèse des calculs*

Les principaux pas de calculs conduisant à l'échantillonnage du déplacement dans l'espace de la tâche sont les suivantes :

- Pour chaque segment $[p_k, p_{k+1}]$, k = 0 à z-1, on extrait les grandeurs $\vec{u}_k$, d_k, $\vec{v}_k$ et ψ_k de la matrice $p_k^{-1} p_{k+1}$.
- Soient V_M et Ω_M les modules des vitesses linéaire et angulaire maximale souhaitées pour le manipulateur concerné. On calcule $V = kV_M$ et $\Omega = k\Omega_M$, où k est le coefficient de vitesse courant. On détermine $T_k = \max\left(\frac{d_k}{V}, \frac{\psi_k}{\Omega}\right)$ pour k = 0 à z-1.
- On calcule les durées δ_k et δ'_k, k = 0 à z, des transitions par les formules (8) et (11), avec d_{-1}, ψ_{-1}, d_z, ψ_z nuls.
- L'équation du déplacement dans l'espace de la tâche est alors $p(t) = \tau(t).\rho(t)$ où $\tau(t)$ et $\rho(t)$ sont définis par les équations (6), (7), (9) et (10). On échantillonne p(t) à une fréquence de l'ordre de 10 Hz pour garantir une précision suffisante du déplacement.

Pour permettre une réaction rapide du manipulateur aux instructions DEPLACER (notamment lorsqu'elles comportent la clause IMMEDIAT), il est souhaitable de ne pas exécuter ces différentes phases séquentiellement. On peut en effet commencer à échantillonner p(t) dès que les paramètres relatifs au début du déplacement ont été déterminés. La période d'échantillonnage permet de calculer les paramètres de la suite du déplacement au fur et à mesure de

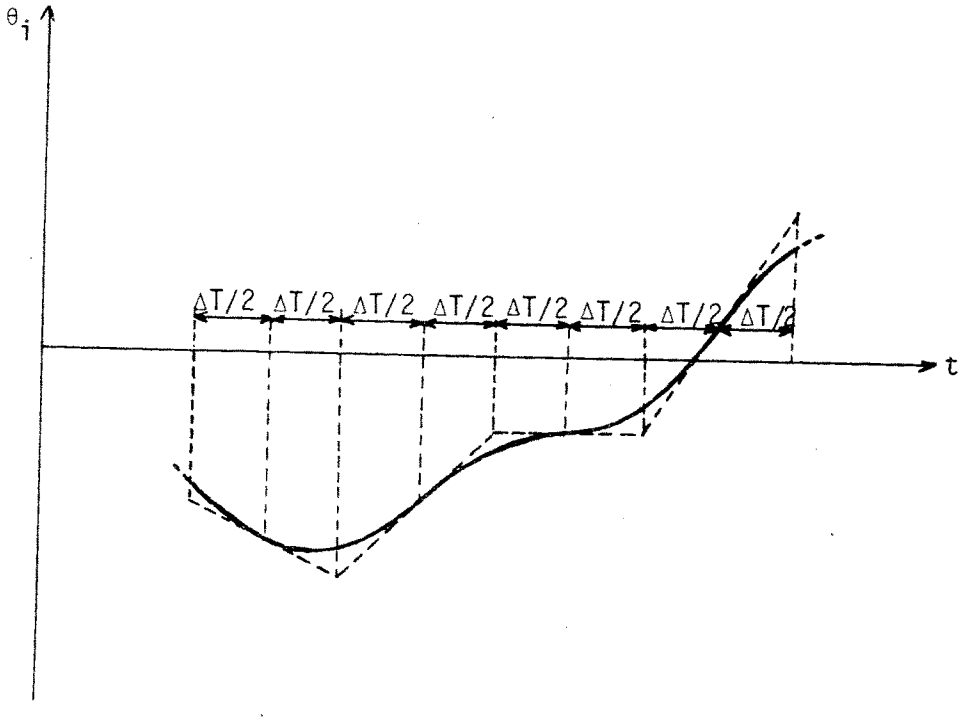


Figure 3

l'échantillonnage. La structure de l'interpréteur proposée au § 4 exploite cette possibilité.

Au cours de ces calculs, une difficulté peut apparaître si, pour une valeur de k, on a $\delta_k+\delta_{k+1} > 2T_k$ ou $\delta'_k+\delta'_{k+1} > 2T_k$. Pour résoudre complètement cette difficulté, il serait nécessaire de planifier complètement le déplacement, ce que nous voulons éviter. Plusieurs traitements simplifiés sont envisageables. On peut par exemple empêcher qu'une transition dépasse la moitié du second segment. Pour cela, chaque fois qu'on calcule δ_k et δ'_k à l'aide des formules (8) et (11), on vérifie que $\frac{\delta_k}{2} \leq \max(\frac{T_k}{2}, T_{k-1} - \frac{\delta_{k-1}}{2})$ et $\frac{\delta'_k}{2} \leq \max(\frac{T_k}{2}, T_{k-1} - \frac{\delta'_{k-1}}{2})$; si $\delta_k > \max(\frac{T_k}{2}, T_{k-1} - \frac{\delta_{k-1}}{2})$, on prend δ_k égal à $\max(\frac{T_k}{2}, T_{k-1} - \frac{\delta_{k-1}}{2})$; on procède de même pour δ'_k. Ce traitement, qui assure la continuité des vitesses linéaire et angulaire, ne garantit cependant pas que les modules des accélérations restent inférieurs à une valeur fixée a priori.

3.4.2. Echantillonnage dans l'espace du manipulateur [17]

Les calculs précédents fournissent, après application de la procédure de changement de coordonnées, une série de vecteurs $\theta^0, \theta^1, \ldots, \theta^Z$ distants dans le temps d'une période ΔT de l'ordre de 100 ms. La fréquence d'alimentation des asservissements étant de l'ordre de 100 Hz, il est nécessaire de réaliser une interpolation entre ces vecteurs.

Une interpolation possible consiste à passer du mileu du segment $[\theta_{\ell-1}, \theta_\ell]$ au milieu du segment $[\theta_\ell, \theta_{\ell+1}]$ à accélération constante (cf. figure 3).

Soit t_ℓ l'instant théorique du passage par θ_ℓ. L'équation de $\theta_i(t)$, pour $t \in [t_\ell - \frac{\Delta T}{2}, t_\ell + \frac{\Delta T}{2}]$, $\forall \ell = 0,1,\ldots,Z$, est alors :

$$\theta_i(t) = \frac{\theta_i^{\ell-1}+\theta_i^{\ell}}{2} + \frac{\Delta\theta_i^{\ell-1}}{\Delta T}(t-t_\ell+\frac{\Delta T}{2}) + \frac{1}{2}\frac{\Delta\theta_i^{\ell}-\Delta\theta_i^{\ell-1}}{\Delta T^2}(t - t_\ell + \frac{\Delta T}{2})^2$$

avec :

. $\Delta\theta_i^{\ell} = \theta_i^{\ell+1} - \theta_i^{\ell}$,

. $\theta_i^{-1} = \theta_i^{0}$,

. $\theta_i^{Z+1} = \theta_i^{Z}$.

On peut commencer à échantillonner les fonctions $\theta_i(t)$ dès que l'on connaît Θ_0 et Θ_1, puis réaliser la suite de l'échantillonnage au fur et à mesure du déplacement.

3.5. Calcul des déplacements du troisième type

Il s'agit de déplacer un repère r d'une position initiale p_0 à une position finale p_1, de telle sorte que l'origine de r suive un chemin rectiligne et que la rotation de r se fasse autour d'un axe unique passant par cette origine.

Le calcul de ce déplacement comporte deux phases d'échantillonnage.

Dans l'espace de la tâche, on exprime $p(t) = \tau(t).\rho(t)$ avec (cf. § 3.4.1) :

$$\tau(t) = \begin{cases} \tau_0.\text{translation}(\vec{u}, d\,\dfrac{(t-t_0+\delta/2)^2}{2\delta T}) & \text{pour } t \in [t_0 - \frac{\delta}{2},\ t_0 + \frac{\delta}{2}] \\ \tau_0.\text{translation}(\vec{u}, d\,\dfrac{t-t_0}{T}) & \text{pour } t \in [t_0 + \frac{\delta}{2},\ t_1 - \frac{\delta}{2}] \\ \tau_1.\text{translation}(-\vec{u}, d\,\dfrac{(t-t_1-\delta/2)^2}{2\delta T}) & \text{pour } t \in [t_1 - \frac{\delta}{2},\ t_1 + \frac{\delta}{2}] \end{cases}$$

$$\rho(t) = \begin{cases} \rho_0.\text{rotation}(\vec{v}, \psi\,\dfrac{(t-t_0+\delta'/2)^2}{2\delta' T}) & \text{pour } t \in [t_0 - \frac{\delta'}{2},\ t_0 + \frac{\delta'}{2}] \\ \rho_0.\text{rotation}(\vec{v}, \psi\,\dfrac{t-t_0}{T}) & \text{pour } t \in [t_0 + \frac{\delta'}{2},\ t_1 - \frac{\delta'}{2}] \\ \rho_1.\text{rotation}(-\vec{v}, \psi\,\dfrac{(t-t_1-\delta'/2)^2}{2\delta' T}) & \text{pour } t \in [t_1 - \frac{\delta'}{2},\ t_1 + \frac{\delta'}{2}] \end{cases}$$

où :

- $\vec{u}$, d, $\vec{v}$ et ψ sont extraits de la matrice $p_0^{-1}.p_1$,
- $T = \max(\frac{d}{V}, \frac{\psi}{\Omega})$,
- $\delta = \frac{1}{\Gamma}\frac{d}{V}$,
- $\delta' = \frac{1}{\Gamma_\Omega}\frac{\psi}{\Omega}$.

Les modules des vitesses linéaire et angulaire, V et Ω, sont pris égaux à k V_M et k Ω_M, où k est le coefficient de vitesse courant.

L'échantillonnage dans l'espace du manipulateur peut être effectué comme au § 3.4.2.

3.6. Calcul des déplacements du quatrième type

Il s'agit de déplacer un repère r le long d'une trajectoire déjà échantillonnée dans l'espace de la tâche (à une position relative près).

L'échantillonnage dans l'espace du manipulateur peut être effectué comme au § 3.4.2.

3.7. Traitements particuliers

a) Modification de la vitesse d'un déplacement

La vitesse d'un déplacement en cours peut être réduite par une instruction VITESSE comportant la clause IMMEDIATE.

La réduction peut être effectuée au niveau de l'échantillonnage dans l'espace du manipulateur. Pour cela, il suffit de dilater l'échelle du temps en remplaçant t par k't dans les formules donnant $\theta_i(t)$, où k' est le coefficient spécifié par l'instruction VITESSE IMMEDIATE.

b) Arrêt d'un déplacement

Un déplacement peut devoir être arrêté avant son arrivée à destination pour différents motifs : lors d'un changement de coordonnées, on constate que le déplacement devient impossible (dépassement d'une butée par exemple), la condition d'arrêt du déplacement est devenue vraie, une instruction ARRETER DEPLACEMENT est exécutée.

L'arrêt d'un déplacement peut être réalisé simplement au niveau de l'échantillonnage des coordonnées propres du manipulateur en dilatant progressivement l'échelle du temps dans les formules donnant les $\theta_i(t)$ [14].

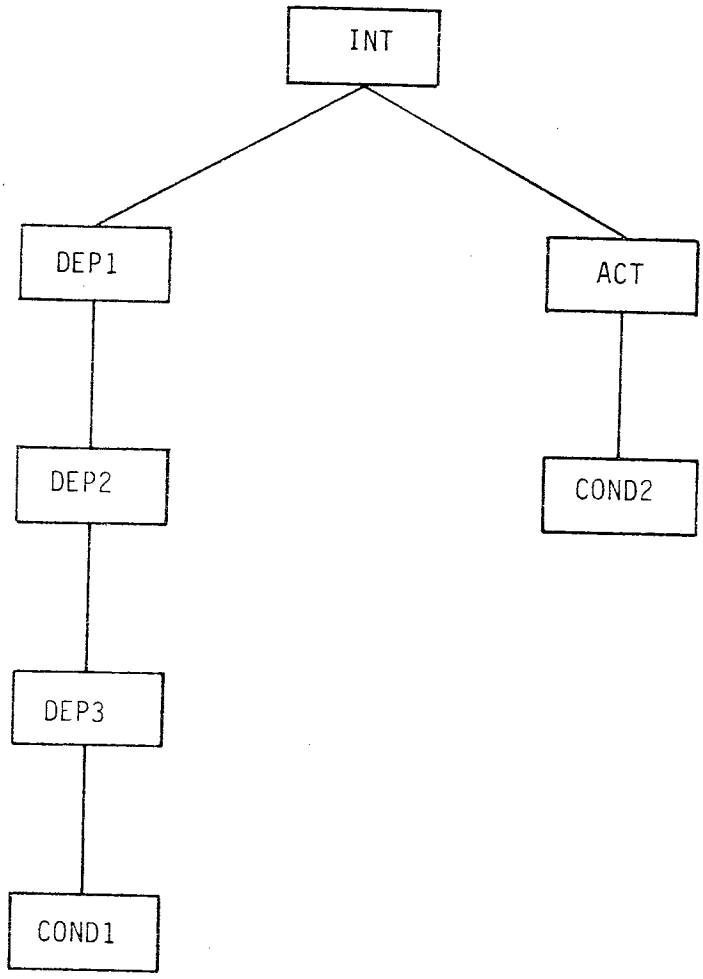


Figure 4

c) *Modification d'un déplacement*

Un déplacement en cours peut être modifié par une instruction DEPLACER comportant la clause IMMEDIAT.

On évalue alors la durée nécessaire au calcul du premier point du nouveau déplacement. Cette durée dépend principalement du type du nouveau déplacement. On peut alors connaître approximativement la position du manipulateur au moment où l'on sera capable d'afficher la première position du nouveau déplacement en entrée des asservissements. On calcule le nouveau déplacement avec cette position pour position initiale.

4. STRUCTURE FONCTIONNELLE DE L'INTERPRETEUR

4.1. Structure générale

Pendant l'exécution d'un programme en LM, plusieurs processus peuvent se dérouler en parallèle. Nous proposons ici une organisation fonctionnelle de l'interpréteur pour un manipulateur (on peut la généraliser facilement pour un nombre quelconque de manipulateurs). Elle repose sur l'interaction de 7 processus notés INT, DEP1, DEP2, DEP3, COND1, ACT et COND2 (cf. figure 4). Nous présentons brièvement ces processus ci-dessous :

- INT réalise toutes les opérations nécessaires à l'interprétation d'un programme en LM, à l'exception du calcul et de la conduite des déplacements du manipulateur et des actions de son outil ;
- DEP1, DEP2 et DEP3 effectuent les calculs des déplacements, à savoir l'échantillonnage dans l'espace de la tâche, les chargements de coordonnées et l'échantillonnage dans l'espace du manipulateur ;
- COND1 surveille la condition d'arrêt éventuelle de chaque déplacement ;
- ACT effectue les calculs nécessaires à l'exécution des actions de l'outil ;
- COND2 surveille la condition d'arrêt éventuelle de chaque action de l'outil.

Dans les paragraphes suivants, nous présentons les interactions entre ces processus.

4.2. Les interactions entre INT, DEP1, DEP2, DEP3 et COND1

Elles ont lieu lors de l'interprétation des instructions DEPLACER, VITESSE IMMEDIATE et ARRETER DEPLACEMENT.

4.2.1. Instruction DEPLACER

Quatre cas peuvent se produire :

a) *Premier cas* : DEP1, DEP2 et DEP3 sont en attente lorsque INT lit une instruction DEPLACER.

Le processus INT range alors :
- la définition déplacement du déplacement dans une mémoire partagée avec DEP1, après avoir évalué les expressions contenues dans cette définition,
- la condition d'arrêt éventuelle dans une mémoire partagée avec COND1, après avoir remplacé les variables déclarées contenues dans cette expression par leurs valeurs courantes.

Puis il active DEP1.

A partir de cet instant, INT ne participe plus à l'interprétation de l'instruction DEPLACER. Si celle-ci ne se termine pas par le symbole "/", il interprète les instructions qui suivent dans le programme.

Lorsqu'il a été activé, DEP1 réalise, s'il y a lieu, l'échantillonnage du déplacement dans l'espace de la tâche. Il transmet à DEP2 les positions successives qu'il détermine, par paquets, dans une mémoire partagée par les deux processus. DEP1 active DEP2 après la transmission du premier paquet, et fait suivre le dernier par une marque de fin de déplacement. Il se met en attente lorsqu'il a transmis cette marque.

Après son activation, DEP2 applique la procédure de changement de coordonnées aux positions transmises par DEP1, jusqu'à la marque de fin de déplacement. Au fur et à mesure, il transmet à DEP3 ces positions exprimées dans les coordonnées propres du manipulateur, par paquets, dans une mémoire partagée par les deux processus. DEP2 active DEP3 après la transmission du premier paquet, et fait suivre le dernier par une marque de fin de déplacement. Il se met en attente lorsqu'il a transmis cette marque.

Une fois activé, DEP3 réalise l'échantillonnage du déplacement dans l'espace des coordonnées propres du manipulateur. Il affiche les positions successives en entrée des asservissements, et il se met en attente lorsque la dernière position a été affichée. Il active COND1 avant d'afficher la première position.

Si la procédure exécutée par DEP2 découvre que le déplacement est impossible (butées, singularités), DEP2 alerte DEP3 qui exécute alors la procédure d'arrêt (cf. § 3.5.b). Il met DEP1, COND1, puis lui-même, en attente. Lorsque l'exécution de la procédure d'arrêt est terminée, DEP3 se met à son tour en attente. Pour permettre un tel arrêt, sans incident mécanique, les calculs de DEP2 doivent avoir une avance suffisante sur les calculs de DEP3. Cette avance, qui dépend du manipulateur et de la vitesse de déplacement, détermine approximativement la taille des paquets transmis entre DEP1, DEP2 et DEP3.

Pendant le déplacement, COND1 (qui a été activé par DEP3) surveille la condition d'arrêt. Si cette condition prend la valeur VRAI, COND1 alerte DEP3 qui exécute alors la procédure d'arrêt. Il met DEP1, DEP2, puis lui-même en attente. Comme précédemment, DEP3 se met en attente lorsque l'exécution de la procédure d'arrêt est terminée.

Certaines définitions de déplacement ne font pas intervenir la procédure d'échantillonnage dans l'espace de la tâche. DEP1 se contente alors de transmettre à DEP2 ces définitions.

b) *Deuxième cas* : au moins un des processus DEP1, DEP2 et DEP3 est actif lorsque INT lit une instruction DEPLACER, et cette instruction ne comporte pas la clause IMMEDIAT.

Le processus INT se met en attente jusqu'à ce que DEP1, DEP2 et DEP3 soient en attente. On est alors ramené au premier cas.

c) *Troisième cas* : DEP3 exécute la procédure d'arrêt lorsque INT lit une instruction DEPLACER.

Que l'instruction comporte la clause IMMEDIAT ou non, le processus INT se met en attente jusqu'à ce que DEP3 soit en attente. On est alors ramené au premier cas.

d) *Quatrième cas* : au moins un des processus DEP1, DEP2 et DEP3 est actif lorsque INT lit une instruction DEPLACER, cette instruction comporte la clause IMMEDIAT et DEP3 n'exécute pas la procédure d'arrêt.

Le traitement est le même que pour le premier cas, à la différence suivante près : l'activation de chaque processus DEP1, DEP2, DEP3 et COND1 est précédée d'une interruption du processus. De cette façon, DEP1, DEP2, DEP3 et COND1 peuvent poursuivre le traitement du déplacement en cours tant que les données nécessaires au traitement du nouveau déplacement ne sont pas prêtes.

4.2.2. Instruction VITESSE IMMEDIATE

Lorsqu'il rencontre une instruction VITESSE IMMEDIATE, le processus INT range le coefficient de vitesse fourni dans une mémoire qu'il partage avec le processus DEP3. La procédure d'échantillonnage exécutée par DEP3 utilise ce coefficient.

Ce coefficient est initialisé à 1 par le processus INT lorsqu'il rencontre une instruction DEPLACER correspondant aux trois premiers cas d'interprétation du § 4.2.2.

4.2.3. Instruction ARRETER DEPLACEMENT

Lorsqu'il rencontre une instruction ARRETER DEPLACEMENT, le processus INT alerte DEP3 qui exécute la procédure d'arrêt. Il met alors DEP1, DEP2 et COND1 en attente.

4.3. Les interactions entre INT, ACT et COND2

Elles ont lieu lors de l'interprétation des instructions ACTIONNER et ARRETER ACTION.

4.3.1. Instruction ACTIONNER

Deux cas peuvent se produire :

a) *Premier cas* : ACT est en attente lorsque INT lit une instruction ACTIONNER.

Le processus INT range alors :
- les paramètres de l'action dans une mémoire partagée avec ACT, après les avoir évalués,
- la condition d'arrêt éventuelle dans une mémoire partagée avec COND2, après avoir remplacé les variables déclarées contenues dans cette expression par leurs valeurs courantes.

Puis il active ACT.

A partir de cet instant, INT ne participe plus à l'interprétation de l'instruction ACTIONNER. Si celle-ci ne se termine pas par le symbole "/", il interprète les instructions qui suivent dans le programme.

Lorsqu'il a été activé, ACT traduit les paramètres de l'action en consignes pour les asservissements des moteurs de l'outil. Il active COND2 avant d'envoyer la première consigne. Il met COND2 et lui-même en attente après l'envoi de la dernière consigne.

Pendant l'action de l'outil, COND2 surveille la condition d'arrêt. Lorsque celle-ci prend la valeur VRAI, COND2 alerte ACT qui exécute alors une procédure d'arrêt de l'action, puis se met en attente. ACT se met en attente lorsque l'exécution de la procédure d'arrêt est terminée.

b) *Deuxième cas* : ACT est actif lorsque INT lit une instruction ACTIONNER.

Le processus INT se met en attente jusqu'à ce que ACT soit en attente.
On est alors ramené au premier cas.

4.3.2. Instruction ARRETER ACTION

Lorsqu'il rencontre une instruction ARRETER ACTION, le processus INT alerte ACT qui exécute la procédure d'arrêt de l'action. Puis il met COND2 en attente.

CHAPITRE 4

LE MATERIEL MIS EN OEUVRE

Dans ce chapitre, nous décrivons le matériel que nous avons mis en oeuvre et sur lequel nous avons réalisé une implantation de LM. Ce matériel est destiné à servir de support expérimental aux recherches en Intelligence Artificielle effectuées dans le cadre du projet PANDORE. Il possède une structure modulaire pour permettre des extensions futures.

La mise en oeuvre du matériel a bénéficié d'une aide technique importante de l'Atelier de Micro-Informatique.

1. VUE D'ENSEMBLE

Le schéma de la figure 1 montre les principaux sous-ensembles fonctionnels du matériel mis en oeuvre et leurs connexions. On y distingue :

- *Deux systèmes mécaniques articulés*. Ils ont respectivement quatre et deux degrés de liberté, une disposition adéquate rendant ces six degrés indépendants. Chaque système est muni d'une pince de préhension. La motorisation (degrés de liberté et pinces) est assurée par des moteurs pas à pas.
- *La commande de puissance*. Elle se compose d'actionneurs pilotant les moteurs pas à pas. Chaque actionneur engendre automatiquement des rampes d'accélération et de décélération, et réalise l'alimentation des moteurs.
- *Deux systèmes à base de microprocesseurs Zilog Z80*. Nous avons défini ces deux systèmes (un par système mécanique) pour qu'ils assurent un ensemble de fonctions de base permettant d'accroître la portabilité du logiciel de plus haut niveau. A chacun de ces deux microprocesseurs est associée une commande manuelle des systèmes mécaniques.
- *Un calculateur PLESSEY Micro 1/03*. Ce calculateur apparaît comme le calculateur central du matériel mis en oeuvre. Son rôle principal est de supporter l'interpréteur du langage LM. Il est prévu qu'il soit prochainement (1981) connecté à l'ordinateur HB-68 du CICG par l'intermédiaire du réseau local de l'IMAG [18].

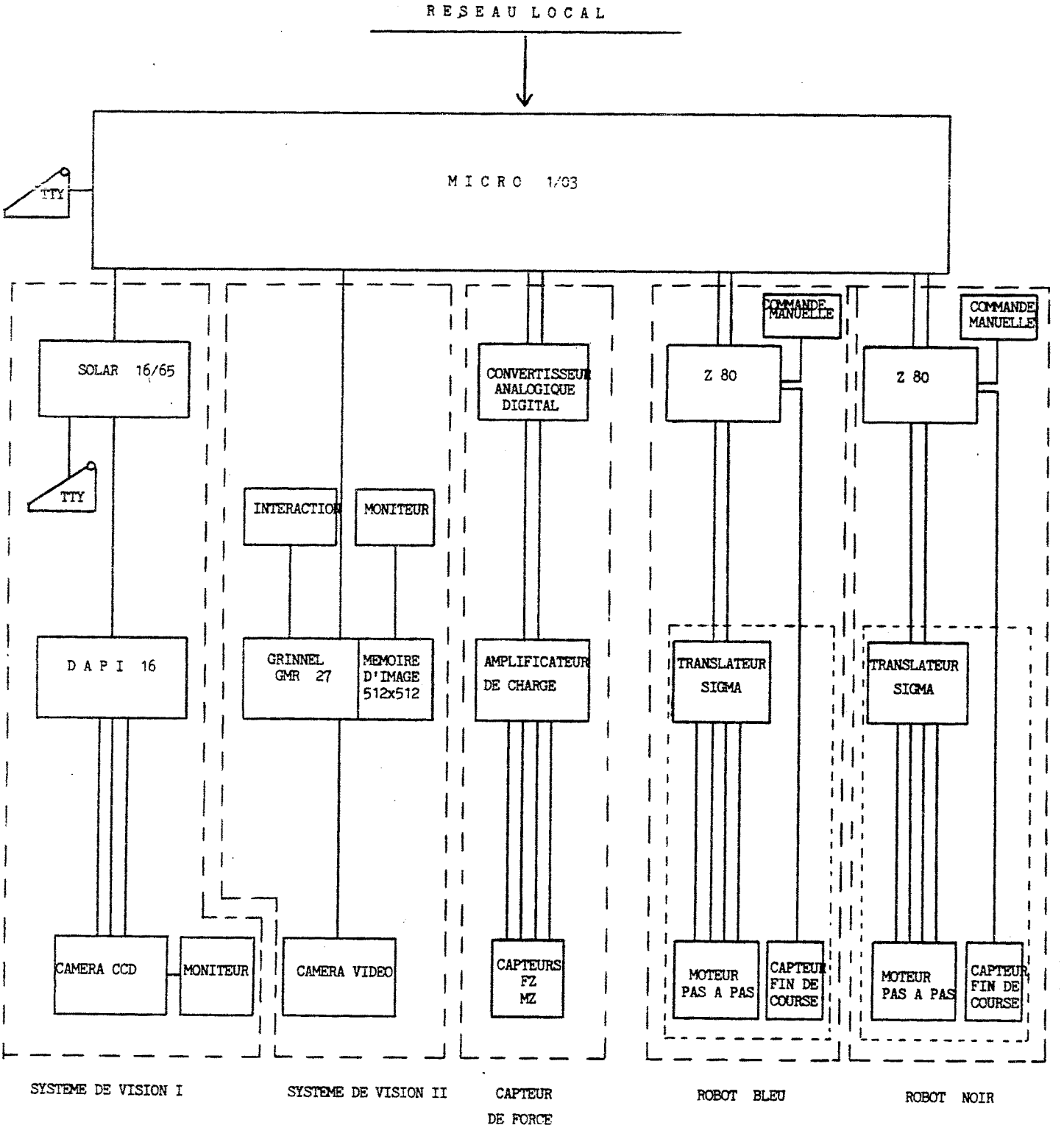

RESEAU LOCAL
MICRO 1/03
TTY
SOLAR 16/65
TTY
D A P I 16
CAMERA CCD
MONITEUR
INTERACTION
MONITEUR
GRINNEL GMR 27
MEMOIRE D'IMAGE 512x512
CAMERA VIDEO
CONVERTISSEUR ANALOGIQUE DIGITAL
AMPLIFICATEUR DE CHARGE
CAPTEURS FZ MZ
COMMANDE MANUELLE
Z 80
TRANSLATEUR SIGMA
MOTEUR PAS A PAS
CAPTEUR FIN DE COURSE
COMMANDE MANUELLE
Z 80
TRANSLATEUR SIGMA
MOTEUR PAS A PAS
CAPTEUR FIN DE COURSE
SYSTEME DE VISION I
SYSTEME DE VISION II
CAPTEUR DE FORCE
ROBOT BLEU
ROBOT NOIR

Figure 1

- *Un capteur de force*. Ce capteur, placé entre la pince et le dernier segment du système mécanique à quatre degrés de liberté, mesure la composante de force le long de l'axe de la pince (FZ) et la composante de moment autour de cet axe (MZ). Une extension du capteur est prévue (début 1981) aux deux composantes latérales de force (FX, FY).
- *Un système de vision I*. Il comporte une caméra CCD connectée à un calculateur SEMS SOLAR 16/65 par un coupleur SEMS DAPI-16. La résolution de l'image obtenue est de 100x100 points digitalisés sur 8 bits.
- *Un système de vision II*. Il comporte une caméra vidéo couplée à un calculateur PLESSEY Micro 2/23 par l'intermédiaire d'un système GRINNELL GMR 27. Ce dernier, qui inclut sa propre mémoire d'image, réalise diverses fonctions d'acquisition et de visualisation d'images. L'installation du Micro 2/23 n'est prévue que pour Janvier 1981 et la connexion a été effectuée provisoirement au Micro 1/03.

Dans les paragraphes suivants, nous donnons une présentation plus détaillée de ces sous-ensembles.

2. LES SYSTEMES MECANIQUES

Il nous semble important de pouvoir disposer, au niveau de la recherche, d'un ensemble mécanique offrant six degrés de liberté indépendants. Dans le matériel mis en oeuvre, nous avons choisi de répartir ces degrés sur deux systèmes complémentaires. Ce choix, qui est en partie dû à des contraintes financières, nous a conduits toutefois à un résultat permettant des travaux intéressants sur la coopération de plusieurs robots.

Les deux systèmes mécaniques sont appelés BLEU et NOIR d'après leurs couleurs respectives.

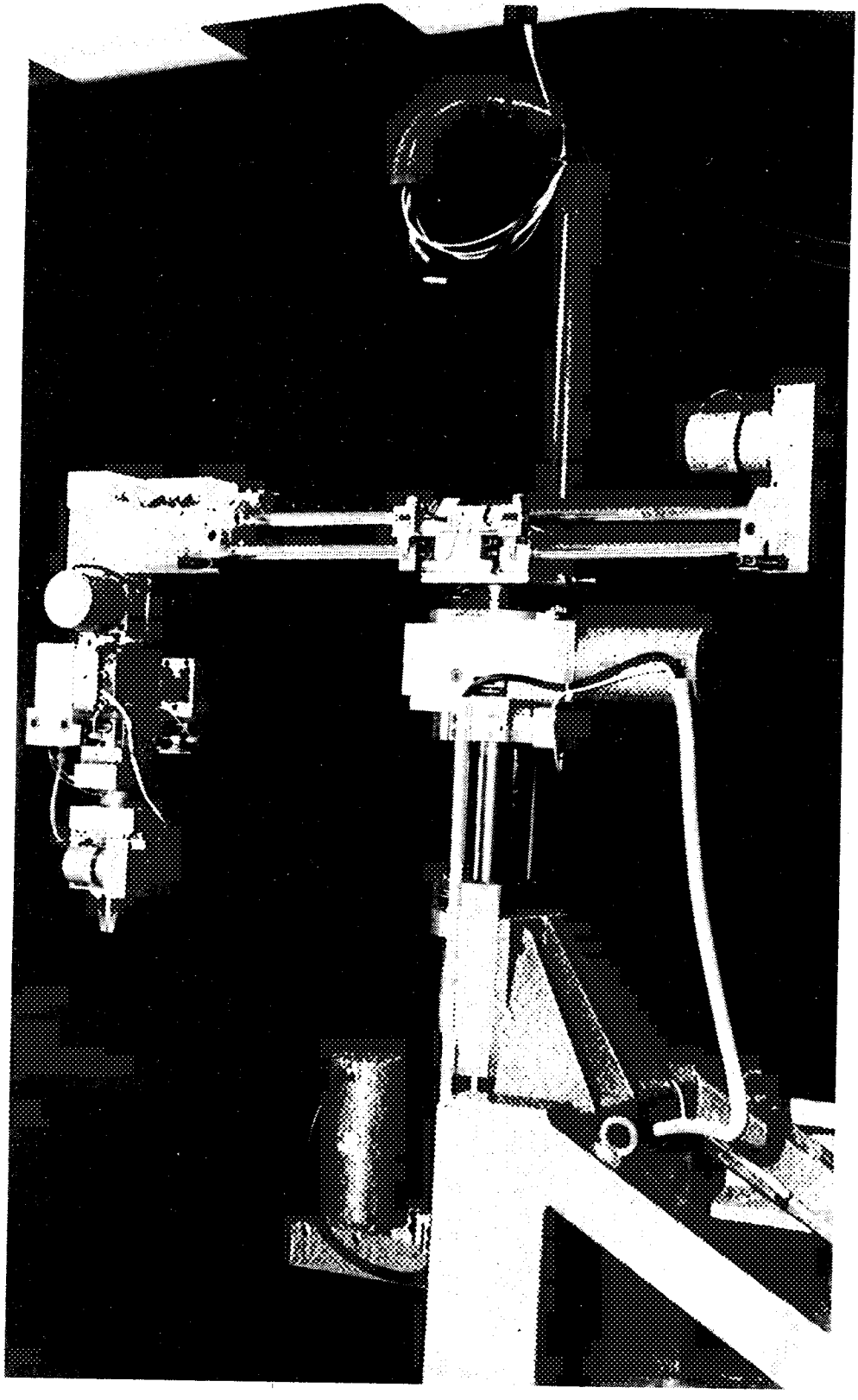


Figure 2

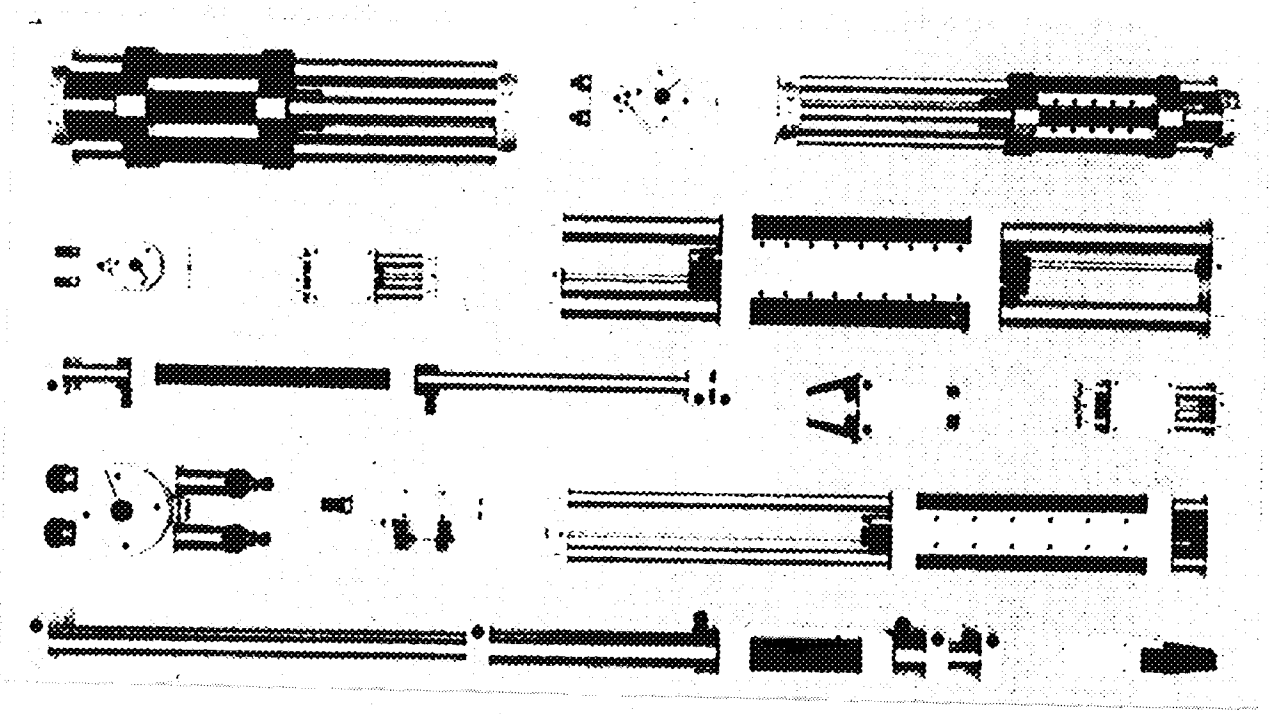


Figure 3

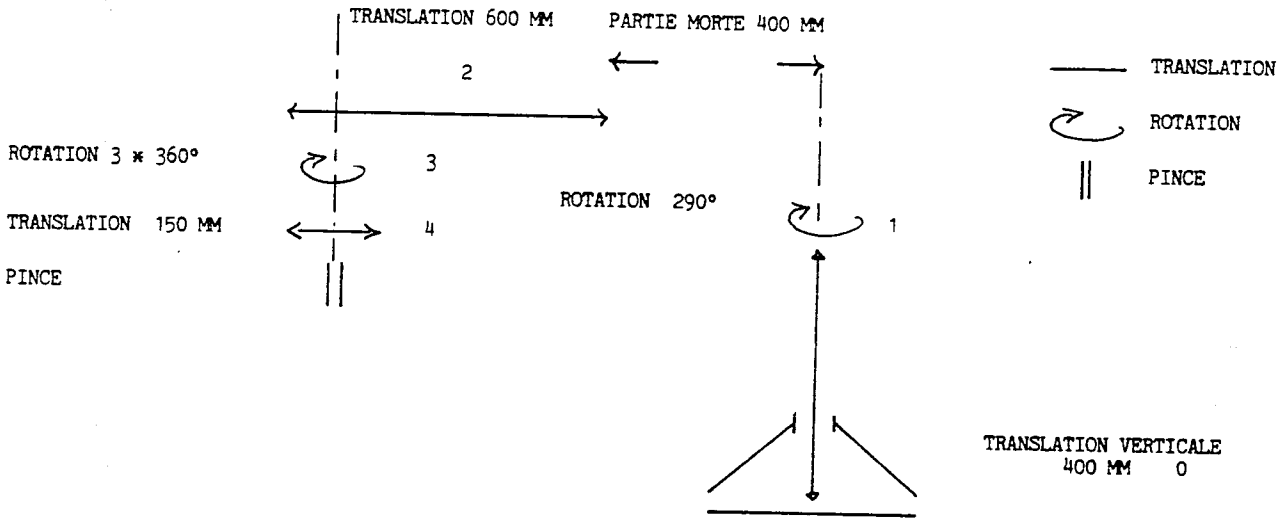


Figure 4

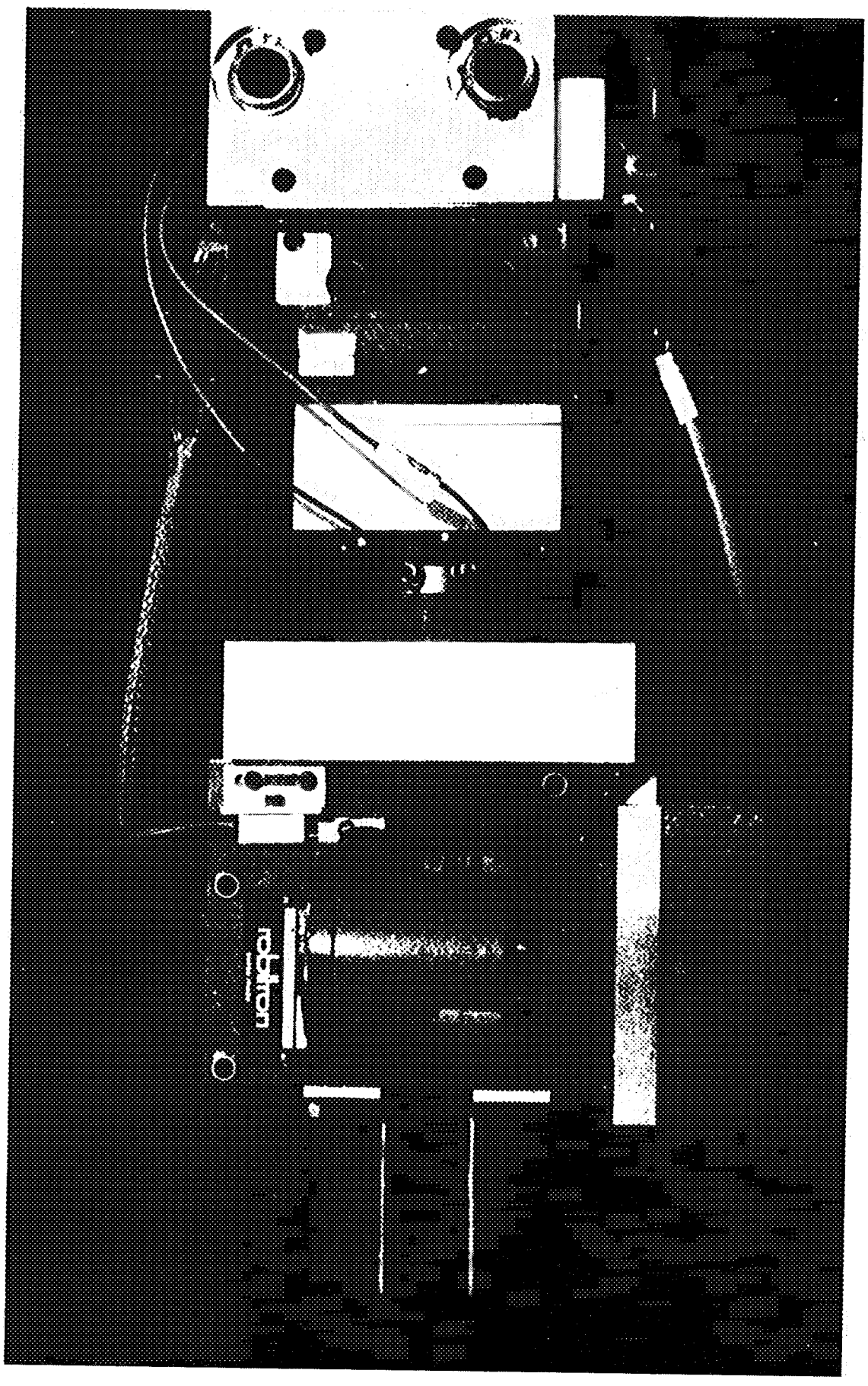


Figure 5

2.1. BLEU (figure 2)

Les composants de BLEU sont dérivés d'éléments modulaires standard construits par la Société ROBITRON. Ces éléments sont de trois types (figure 3) : éléments linéaires (translations), éléments de rotation et pinces. La Société ROBITRON les a mis au point pour construire à la demande des machines automatiques destinées notamment au chargement-déchargement d'autres machines (presses à injecter, fours, machines-outils, ...).

Sur la base des éléments du catalogue ROBITRON, nous avons défini un bras manipulateur à quatre degrés de liberté indépendants. Ces degrés sont réalisés, dans l'ordre (à partir du bâti fixe), par une translation T1 verticale, une rotation R1 autour d'un axe vertical, une translation T2 horizontale et une rotation R2 autour d'un axe vertical (figure 4). Compte tenu des restrictions imposées par les alimentations des moteurs, les courses de T1, R1, T2 et R2 sont 400 mm, 280°, 600 mm et 3x360°.

Une pince de préhension est fixée sur la partie mobile de la rotation R2. Elle comporte deux mors parallèles mobiles disposés verticalement et s'ouvrant vers le bas (figure 5). Les déplacements des deux mors sont simultanés et symétriques par rapport à un axe aligné avec l'axe de la rotation R2. La course relative des deux mors est de 60 mm.

Les trois premiers degrés de liberté T1, R1 et T2 déterminent donc un système de coordonnées cylindriques permettant de donner à un point de la pince une position quelconque dans un certain volume. Le quatrième degré de liberté R2 permet de faire tourner la pince autour de l'axe des mors.

Pour simplifier la commande, nous avons choisi d'équiper chaque degré de liberté de BLEU, ainsi que sa pince, d'un moteur pas à pas. Les opérations du manipulateur peuvent ainsi être effectuées en boucle ouverte (cf. § 3), c'est-à-dire sans avoir besoin de capteurs de position. Certains éléments standard (éléments de rotation et pince) n'avaient été conçus qu'en vue d'une commande "tout ou rien" par des actionneurs hydrauliques ou pneumatiques. La Société ROBITRON les a modifiés pour les adapter à la motorisation électrique.

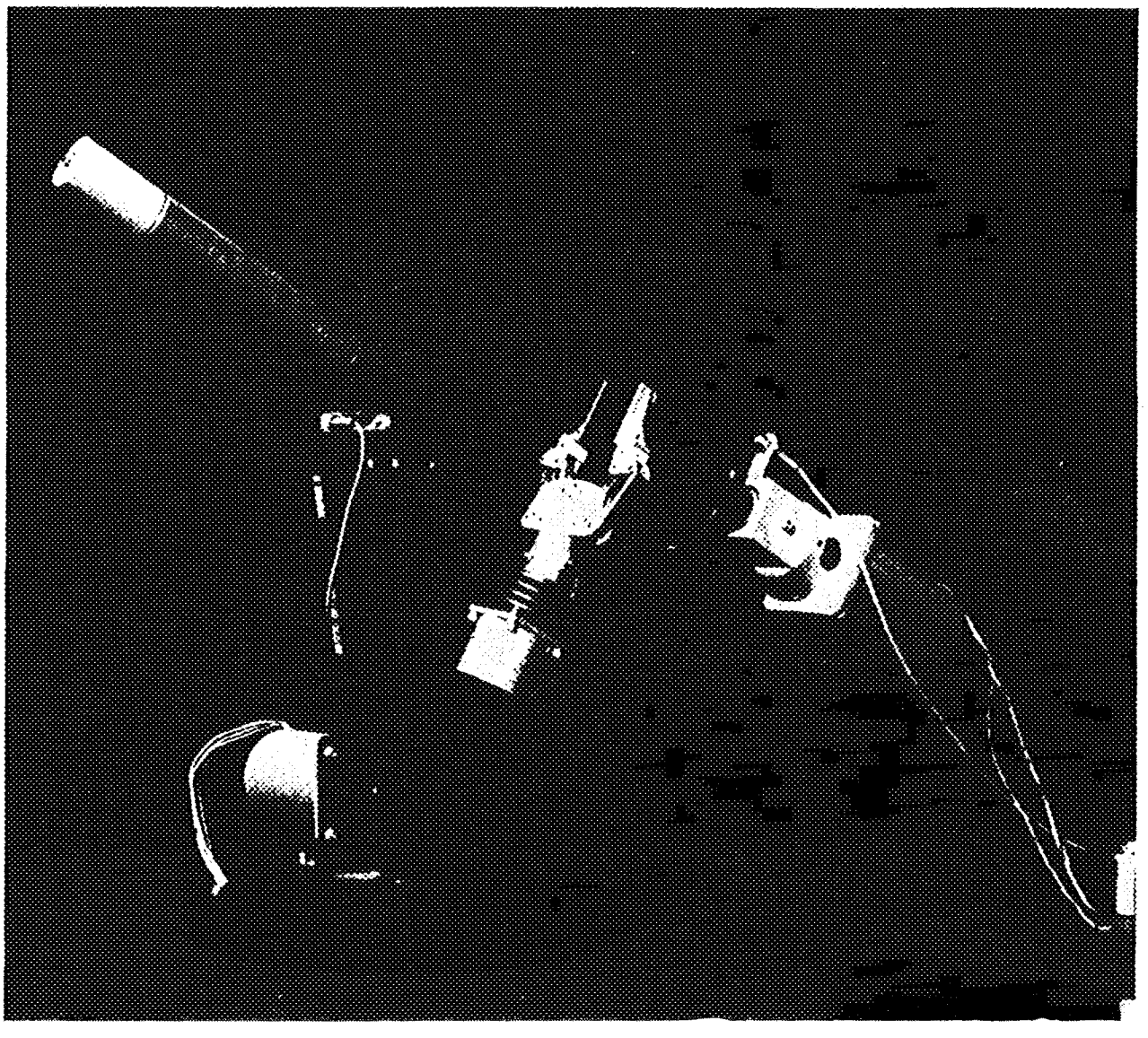


Figure 6

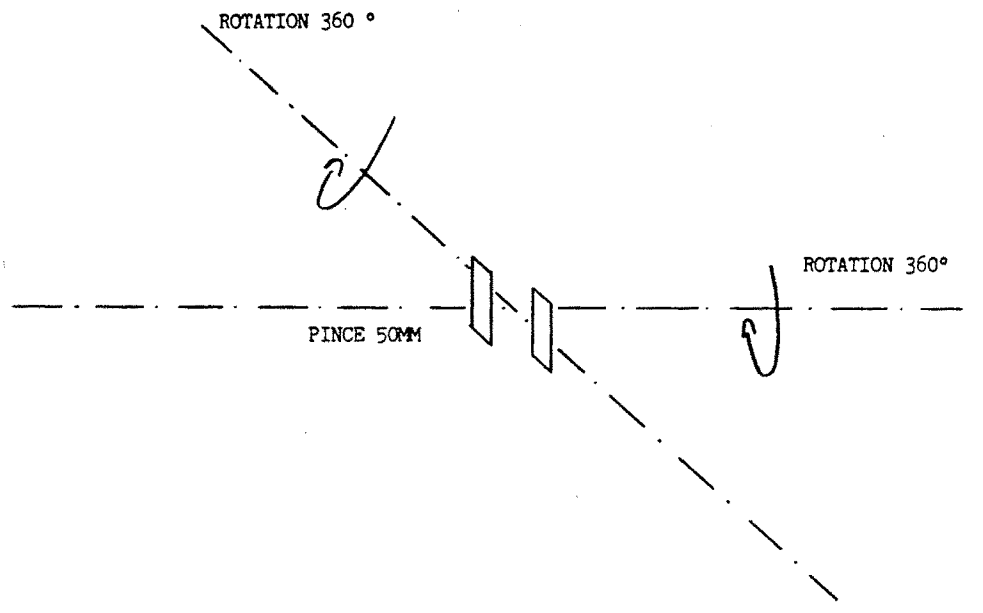
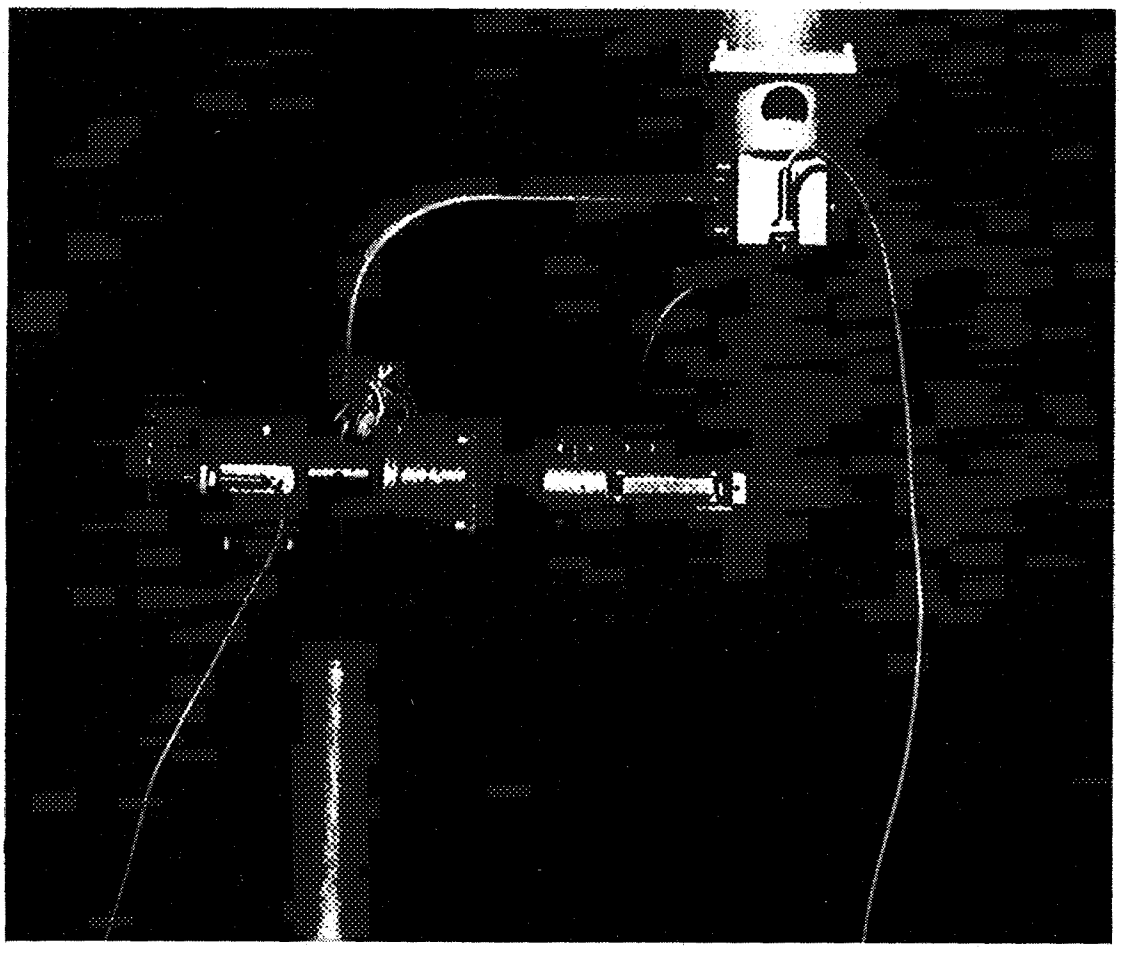


Figure 7



Les moteurs utilisés sont des moteurs SIGMA à 200 pas par tour. Commandés en 1/2 pas, et compte tenu des démultiplications sur les transmissions, ils permettent les résolutions de déplacement théoriques suivantes : 0,02 mm pour T1, 0,02 degré pour R1, 0,02 mm pour T2 et 0,01 degré pour R2. Les vitesses maximales prévues sont 20 cm/s sur T1, 0,2 tour/s sur R1, 20 cm/s sur T2, 1 tour/s sur R2 et 10 cm/s sur la pince. Bien qu'assez faibles, ces vitesses nous paraissent provisoirement suffisantes pour un support expérimental de recherche. La charge maximale prévue dans la pince est de 50 N.

Les degrés de liberté et la pince sont munis de contacts de "fin de course" (capteurs de proximité à induction magnétique). Ces contacts sont utilisés pour assurer la sécurité de BLEU en cours de fonctionnement : ils commandent directement l'ouverture du contacteur d'alimentation des moteurs. Cette action peut toutefois être inhibée par le système Z80 (cf. § 4.1) pour permettre l'initialisation des degrés de liberté de BLEU et de sa pince (cf. § 6, chapitre 5).

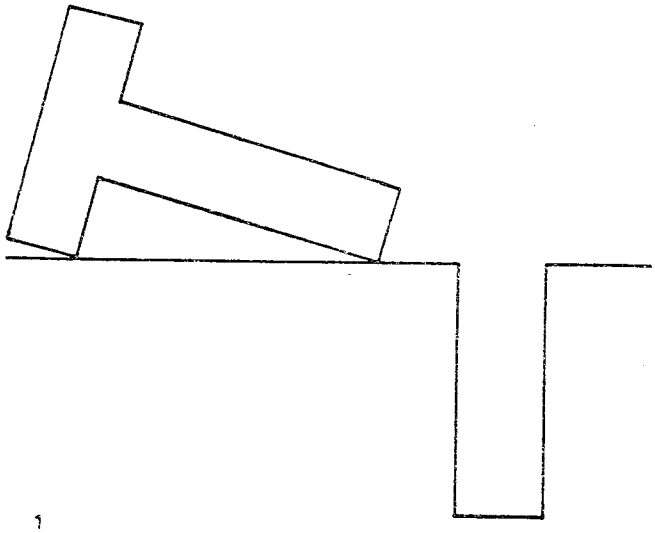
2.2. NOIR (figure 6)

NOIR est un système mécanique qui a été construit par la Société BARRAS-Provence d'après nos spécifications. Il comporte deux degrés de liberté réalisés par deux rotations R'1 et R'2 de 360° chacune (figure 7). Les axes de R'1 et R'2 sont perpendiculaires, celui de R'1 étant fixe et horizontal.

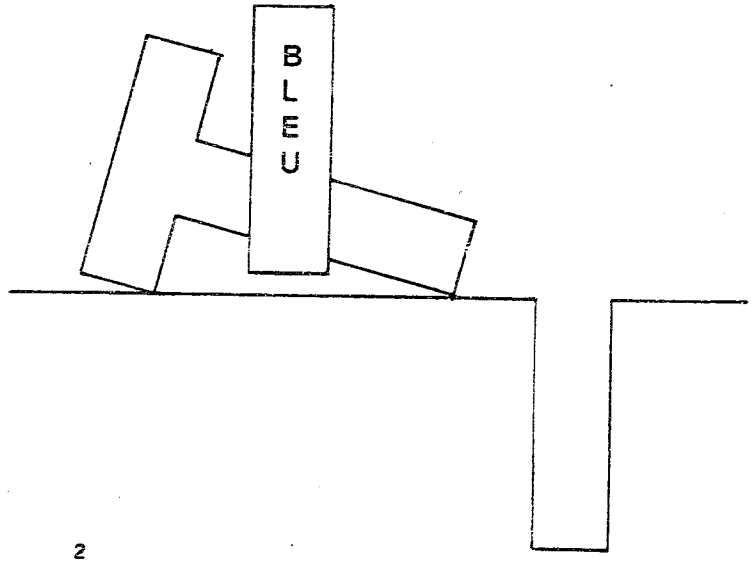
Une pince de préhension à deux mors parallèles (figure 8) est montée sur la partie mobile de R'2. Comme pour la pince de BLEU, les mors se déplacent simultanément et symétriquement par rapport à un axe perpendiculaire à l'axe de R'2. La course relative des deux mors est de 60 mm.

Les deux axes des rotations R'1 et R'2 sont concourants en un point situé sur l'axe de la pince entre les deux mors. Tous les mouvements de NOIR laissent donc ce point immobile.

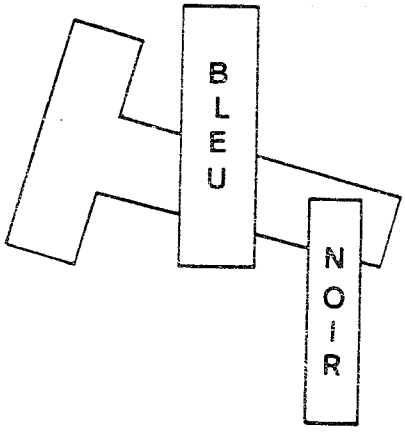
Comme pour BLEU, nous avons choisi d'équiper les deux rotations et la pince de NOIR de moteurs pas à pas SIGMA à 200 pas/tour. Commandés en



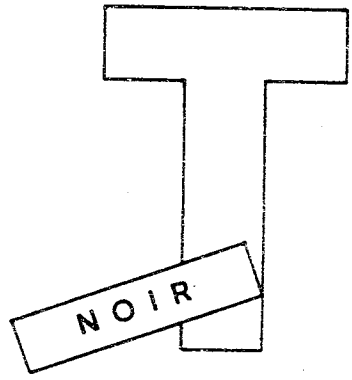
1



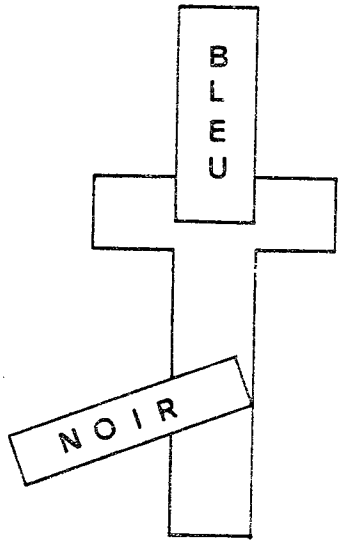
2



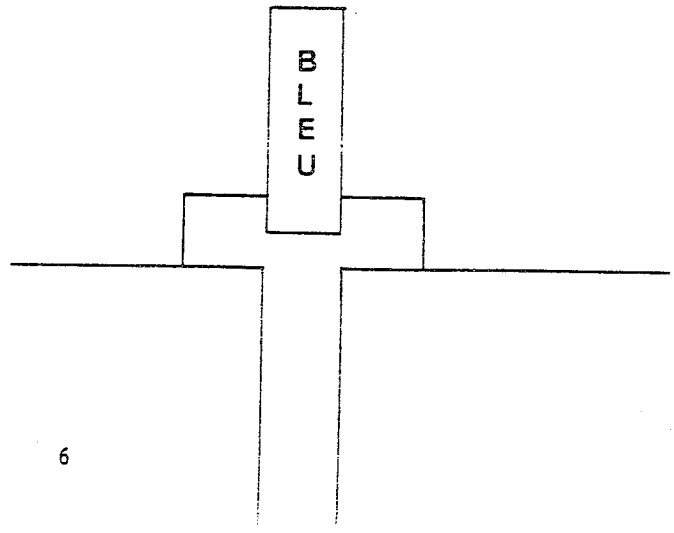
3



4



5



6

Figure 9

1/2 pas, ces moteurs fournissent les résolutions de déplacement suivantes : 0,2 degré sur R'1 et 0,9 degré sur R'2. Les vitesses maximales prévues sont 0,5 tour/s pour R'1 et 0,5 tour/s pour R'2. La charge maximale dans la pince est de 50 N.

A chaque rotation est associé un capteur de proximité à induction magnétique. De plus, la pince est munie de deux micro-interrupteurs (figure 8) actionnés par un système mécanique à base d'une rondelle "Belleville" lorsque la pince arrive en ouverture maximale ou lorsque la force de serrage exercée par les mors dépasse une certaine valeur (de l'ordre de 150 N). Ces capteurs et contacts commandent directement l'ouverture du contacteur d'alimentation des moteurs. Cette action peut être inhibée par le système Z80 pour permettre d'initialiser les degrés de liberté et la pince.

Nous avons disposé NOIR de telle sorte que sa pince soit accessible par celle de BLEU. Les degrés de liberté de NOIR étant indépendants de ceux de BLEU (sauf dans la position singulière où les axes de R2 et de R'2 sont alignés), une coopération des deux systèmes nous permet de disposer ainsi d'un ensemble de six degrés de liberté indépendants. Il est donc possible, dans les limitations du volume accessible par la pince de BLEU, de donner une position et une orientation quelconques à un objet. La figure 9 illustre un exemple de coopération entre BLEU et NOIR.

3. LA COMMANDE DE PUISSANCE

Elle sert d'interface entre la commande programmable assurée par les calculateurs (cf. § 4 et 5) et les moteurs pas à pas de BLEU et NOIR. Elle se compose de huit actionneurs (un par moteur) : cinq d'entre eux (ceux de BLEU) sont des actionneurs SIGMA ; les trois autres (ceux de NOIR) sont des actionneurs ATAC.

Chaque actionneur assure la commande d'un moteur en boucle ouverte. Pour un déplacement à réaliser, il reçoit deux types d'informations : d'une part, une donnée binaire qui spécifie le sens de rotation du moteur, et d'autre part un train d'impulsions. Le nombre d'impulsions définit le nombre de 1/2 pas dont doit tourner le moteur, c'est-à-dire, à un facteur

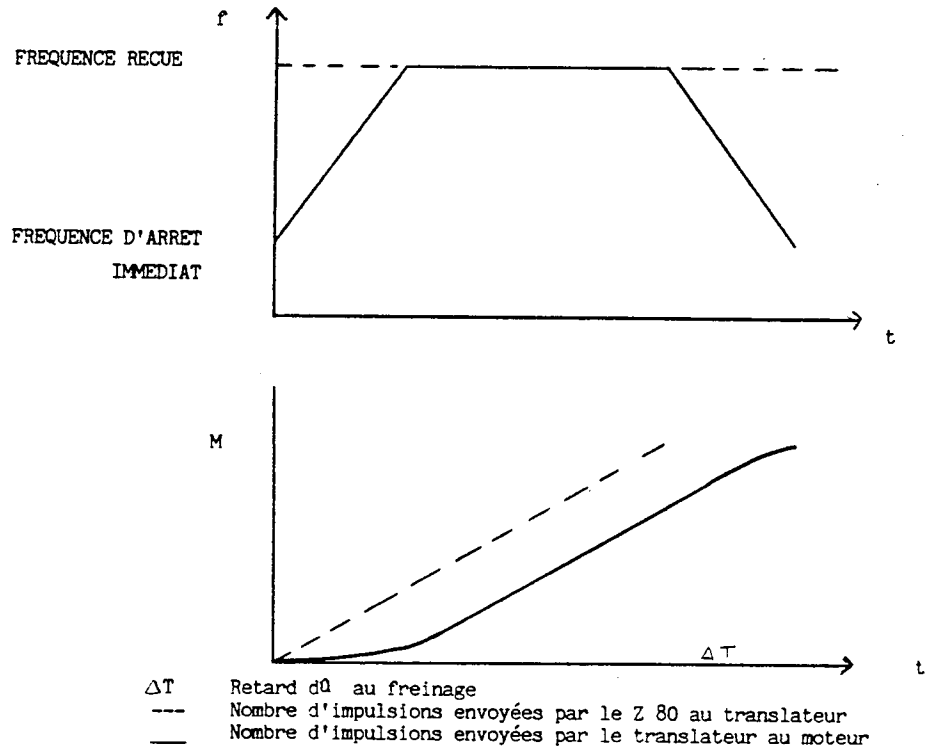
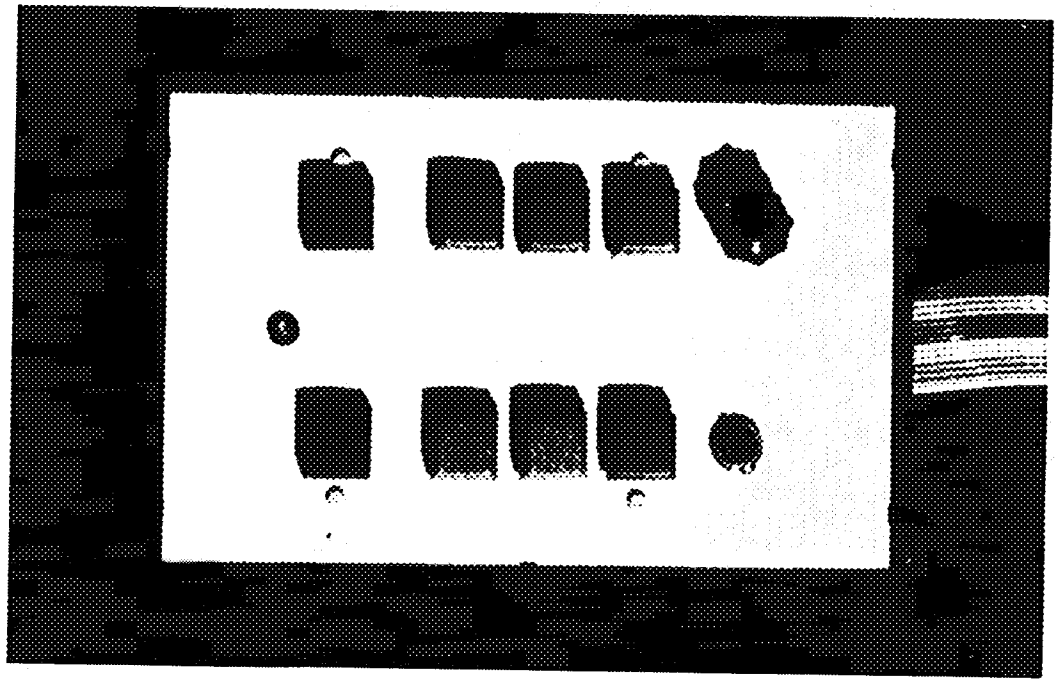


Figure 10

Retour du contrôle T2 R1 T1 Vitesse



Arrêt d'urgence Pince T3 R2 Sens

Figure 11

près dépendant des démultiplications, l'amplitude du déplacement du degré de liberté correspondant. La fréquence des impulsions détermine la vitesse souhaitée pour le moteur.

Simultanément avec la réception des impulsions, l'actionneur réalise les deux fonctions suivantes :

- il transforme le train d'impulsions pour inclure des rampes d'accélération et de décélération évitant les changements brusques de vitesse (notamment au démarrage et à l'arrêt),
- il détermine à chaque instant les bobinages du moteur à alimenter en fonction du sens de rotation demandé et réalise l'alimentation.

Tous les actionneurs utilisés commandent les moteurs en 1/2 pas.

La figure 10 illustre la génération de rampes d'accélération et de décélération par un actionneur recevant un train d'impulsions à fréquence constante. L'actionneur modifie la fréquence des impulsions au début et à la fin du déplacement pour produire une accélération progressive du moteur depuis une vitesse minimale dite de démarrage (ou d'arrêt) immédiat, jusqu'à la vitesse spécifiée par la fréquence du train d'impulsions reçu (si le déplacement est suffisamment long pour permettre de l'atteindre), puis une décélération symétrique en fin de déplacement. Ces rampes ont pour but d'éviter que le moteur ne "décroche" sur un couple inertiel trop important. Elles résultent en un retard de la position du moteur sur le train d'impulsions reçu par l'actionneur. Toutefois, le nombre total des impulsions n'est pas modifié, de telle sorte que la position finale du moteur soit celle demandée.

Chacun des actionneurs peut recevoir un train d'impulsions à fréquence non constante. Si les variations de fréquence sont trop brusques (accélérations importantes), il engendre alors automatiquement des transitions équivalentes aux rampes présentées ci-dessus. Cette possibilité permet théoriquement de contrôler les accélérations moyennes des moteurs par le logiciel et donc de faire effectuer des trajectoires précalculées aux systèmes articulés.

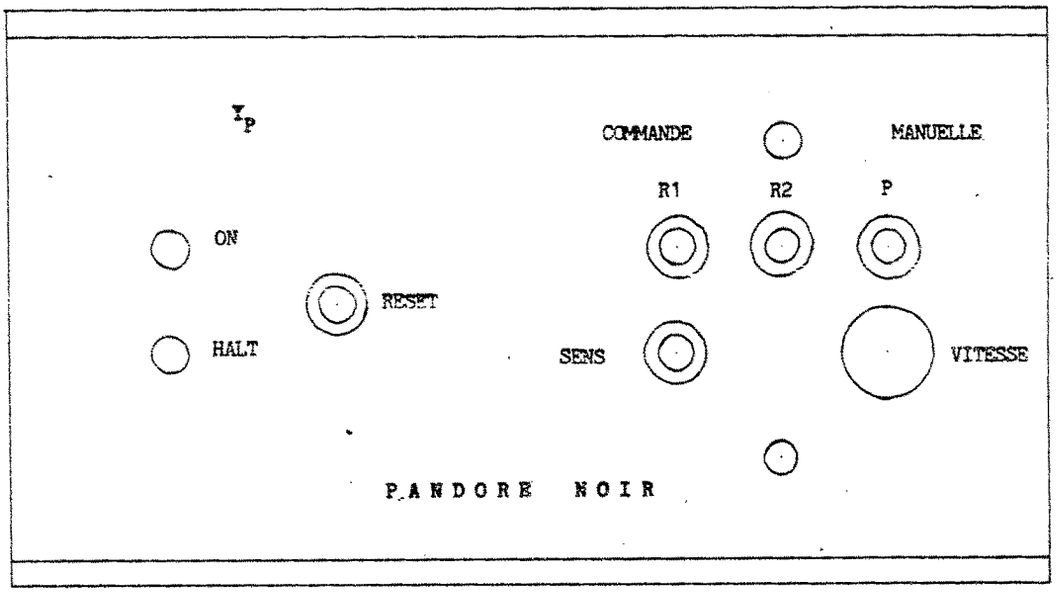


Figure 12

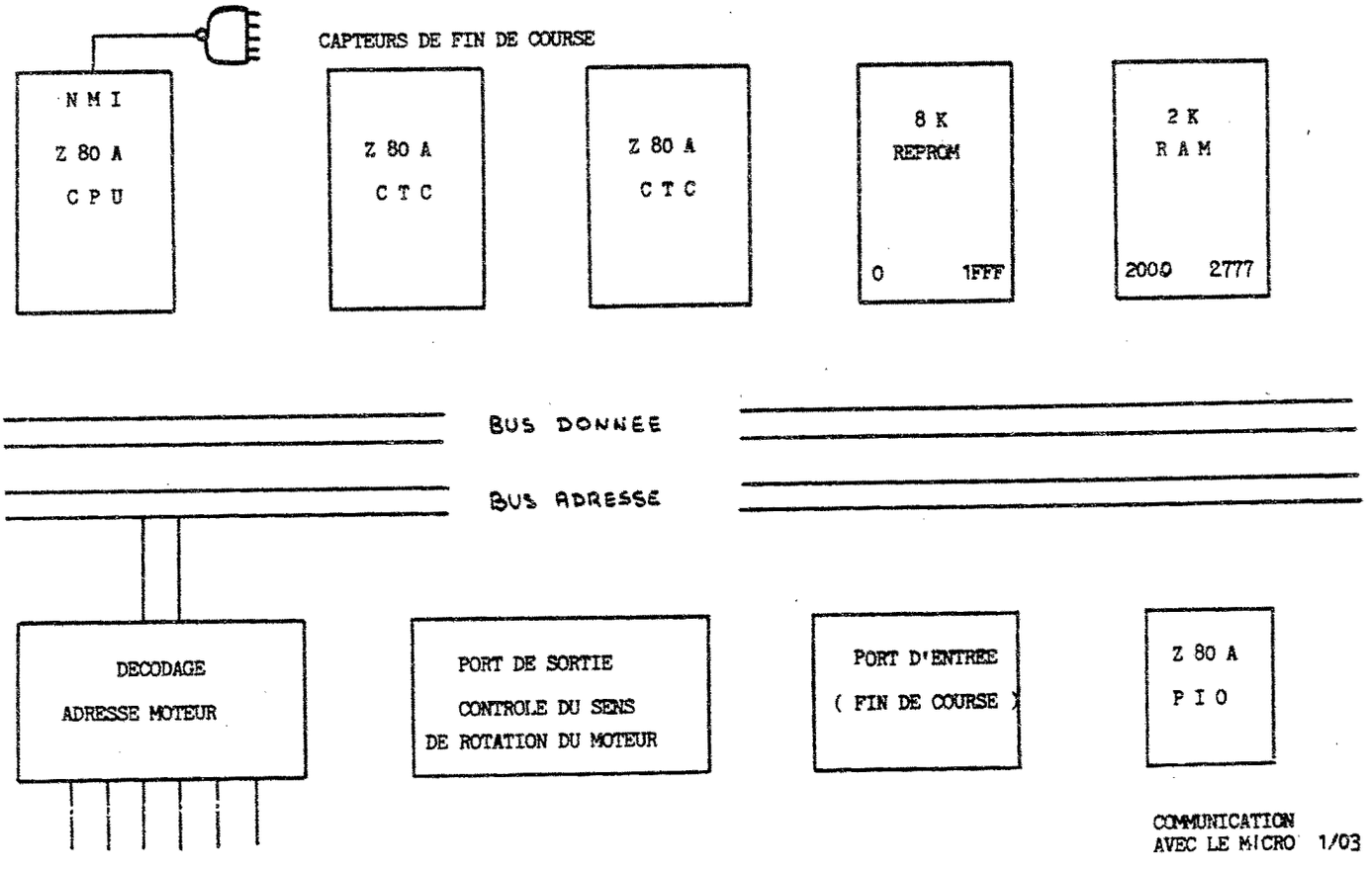


Figure 13

On peut régler manuellement les pentes des rampes sur tous les actionneurs utilisés. Les réglages ont été effectués avec une marge de sécurité suffisante pour garantir la validité de la commande en boucle ouverte tant qu'un effort anormal n'est pas exercé. Un tel effort, qui pourrait endommager BLEU et NOIR, doit être prévenu à un niveau de commande plus élevé en particulier en utilisant les informations transmises par le capteur de force (cf. § 6).

4. LES SYSTEMES Z80

Deux systèmes identiques à base de microprocesseurs ZILOG Z80 assurent respectivement la commande des cinq actionneurs de BLEU et celle des trois actionneurs de NOIR. Le rôle de ces systèmes est double :

(1) Nous les avons conçus pour qu'ils assurent des fonctions de base (cf. Annexe A) dont la réalisation dépend étroitement des systèmes mécaniques BLEU et NOIR, et de leur motorisation. Ils contribuent donc à accroître la portabilité du logiciel implanté sur le Micro 1/03 (§ 5).

(2) Ils peuvent travailler en parallèle avec le Micro 1/03 et déchargent celui-ci lorsque plusieurs traitements doivent être exécutés "simultanément". Nous les utilisons notamment pour contrôler les déplacements de BLEU et NOIR pendant que le Micro 1/03 continue l'interprétation du programme LM en cours d'exécution.

De plus, chaque système Z80 est muni d'un dispositif permettant de commander manuellement le système mécanique correspondant. Pour BLEU, ce dispositif est un boîtier portable (figure 11). Pour NOIR, il est disposé en face avant du système Z80 (figure 12).

La figure 13 montre un schéma de principe d'un système Z80. Outre le microprocesseur, ce système comporte 8K de mémoire morte reprogrammable et 2K de mémoire vive, un interface avec le Micro 1/03, des circuits d'horloge (2 CTC) et des circuits d'entrées-sorties.

Les circuits d'entrées-sorties permettent de commander les actionneurs, de connaître l'état des boutons du dispositif de commande manuelle et de

gérer les sécurités (contacts de "fin de course"). Ils comportent des entrées-sorties supplémentaires permettant la synchronisation de BLEU et NOIR avec d'autres systèmes et la commande de dispositifs mécaniques simples (par exemple un étau de serrage pneumatique).

Les trains d'impulsions envoyés aux actionneurs sont engendrés par programme. Pour cela, nous utilisons les deux circuits CTC (Control Timer Circuit), qui comportent chacun quatre décompteurs programmables. Chaque décompteur est connecté à une horloge de référence et envoie une interruption au microprocesseur à tous ses passages par zéro. Il est ainsi possible de choisir par programme l'intervalle de temps séparant deux interruptions, donc la fréquence d'envoi des impulsions.

Une description plus détaillée des systèmes Z80 est donnée dans l'Annexe B.

5. LE CALCULATEUR PLESSEY MICRO 1/03

Son rôle principal est de supporter l'interpréteur du langage LM.

Il s'agit d'un calculateur à mots de 16 bits construit autour d'un LSI-11/2 de DEC. Notre configuration comporte 32K mots de mémoire centrale et une unité de disques souples de 2x256 Koctets. L'unité centrale est munie d'un circuit de calcul en virgule flottante.

Ce calculateur est relié aux systèmes Z80 par une ligne parallèle partagée de 16 bits, au SOLAR 16/65 (système de vision I, § 7) par une ligne série et au capteur de force (§ 6) par un convertisseur analogique/digital placé directement sur son bus. De plus, il sera prochainement connecté à un calculateur PLESSEY Micro 2/23 (système de vision II, § 8) par une ligne série et au calculateur HB-68 du Centre Interuniversitaire de Calcul de Grenoble par le réseau local de l'IMAG.

Cette dernière connexion permettra d'expérimenter les méthodes de synthèse automatique de programmes d'assemblages en cours de développement sur l'HB-68 (cf. chapitre 1).

6. LE CAPTEUR DE FORCE

Il s'agit d'un capteur KISTLER constitué de rondelles de quartz piezo-électrique placées dans un boîtier métallique. Il est monté entre le dernier segment du manipulateur BLEU et sa pince (figure 5).

Lorsqu'un effort est exercé sur la pince, des charges électriques s'accumulent sur certaines rondelles. La quantité de charges formées sur une rondelle dépend d'une part de la nature de l'effort exercé (force ou moment) et de sa direction, et d'autre part du sens de découpage de la rondelle relativement aux cristaux de quartz.

Actuellement, le capteur ne comporte que deux rondelles permettant de mesurer FZ (force suivant l'axe de la pince) et MZ (moment autour de cet axe) respectivement. Il est prévu d'ajouter deux nouvelles rondelles permettant de mesurer les composantes latérales de force FX et FY (début 1981). Par la suite, il serait également utile de pouvoir mesurer les composantes de moments MX et MY, de façon à disposer du torseur de force complet.

Le choix d'un capteur piézo-électrique, plutôt que d'un capteur à jauges de contraintes [22], résulte principalement des deux facteurs suivants :

- Le capteur piézo-électrique peut supporter sans dommage des surcharges de plusieurs tonnes. Au contraire, le capteur à jauges est assez fragile et requiert une protection mécanique. La gamme de mesure du capteur piézo-électrique est donc plus étendue.
- Tant qu'on ne cherche pas à mesurer MX et MY, le capteur piézo-électrique est une pile de rondelles de quartz permettant chacune de mesurer une composante FX, FY, FZ ou MZ. Il est donc possible de calibrer chaque composante indépendamment des autres. Le calibrage d'un capteur à jauges est plus complexe [23].

Chacune des deux sorties du capteur actuel est reliée à un amplificateur de charges dont la sortie est un signal analogique. Malgré la haute impédance d'entrée des amplificateurs (10^{14} Ω) et l'utilisation de cables spéciaux pour le transport des charges, les mesures doivent être faites en mode dynamique ou "quasi-statique". Chaque amplificateur comporte donc

une entrée binaire permettant d'effectuer la remise à zéro des charges sur la rondelle de quartz correspondante. A tout moment, le signal de sortie est proportionnel à la variation de la grandeur FZ ou MZ depuis l'instant de la dernière remise à zéro.

Les amplificateurs utilisés permettent de mesurer les variations maximales suivantes : ± 100N pour FZ, ± 200 N/cm pour MZ.

Les sorties analogiques des amplificateurs sont reliés à un convertisseur analogique/digital. Celui-ci possède 16 entrées multiplexées échantillonnées à la fréquence de 30 KHz. Sa sortie digitalisée est directement connectée au bus du Micro 1/03. Son gain est réglable par programme.

Une ligne de contrôle disponible sur l'interface parallèle reliant le Micro 1/03 aux systèmes Z80 est utilisée pour commander la remise à zéro simultanée des amplificateurs de charges.

Pour permettre une mesure progressive des efforts, nous avons intercalé une plaque de caoutchouc entre le capteur de force et le dernier segment de BLEU (figure 14).

7. LE SYSTEME DE VISION I

Il se compose d'une caméra Fairchild CCD 202.A, d'un coupleur SEMS DAPI 16, d'un calculateur SEMS SOLAR 16/65 et d'un moniteur TEKTRONIX 620.

La caméra comporte une matrice CCD (Charged Coupled Device) 100x100, une carte réalisant le balayage de cette matrice à une vitesse réglable, et une carte d'interface avec le DAPI 16. En sortie on dispose d'une part d'un signal analogique comportant 112x112 points incluant les 100x100 points de l'image et des signaux de contrôle, et d'autre part le signal d'horloge pour l'échantillonnage de ce signal.

L'acquisition est faite en 1/5 de seconde. La carte d'interface avec le DAPI 16 permet de synchroniser une acquisition d'image sur un début de balayage et réalise le comptage des points. Elle est décrite en détail dans l'Annexe C.

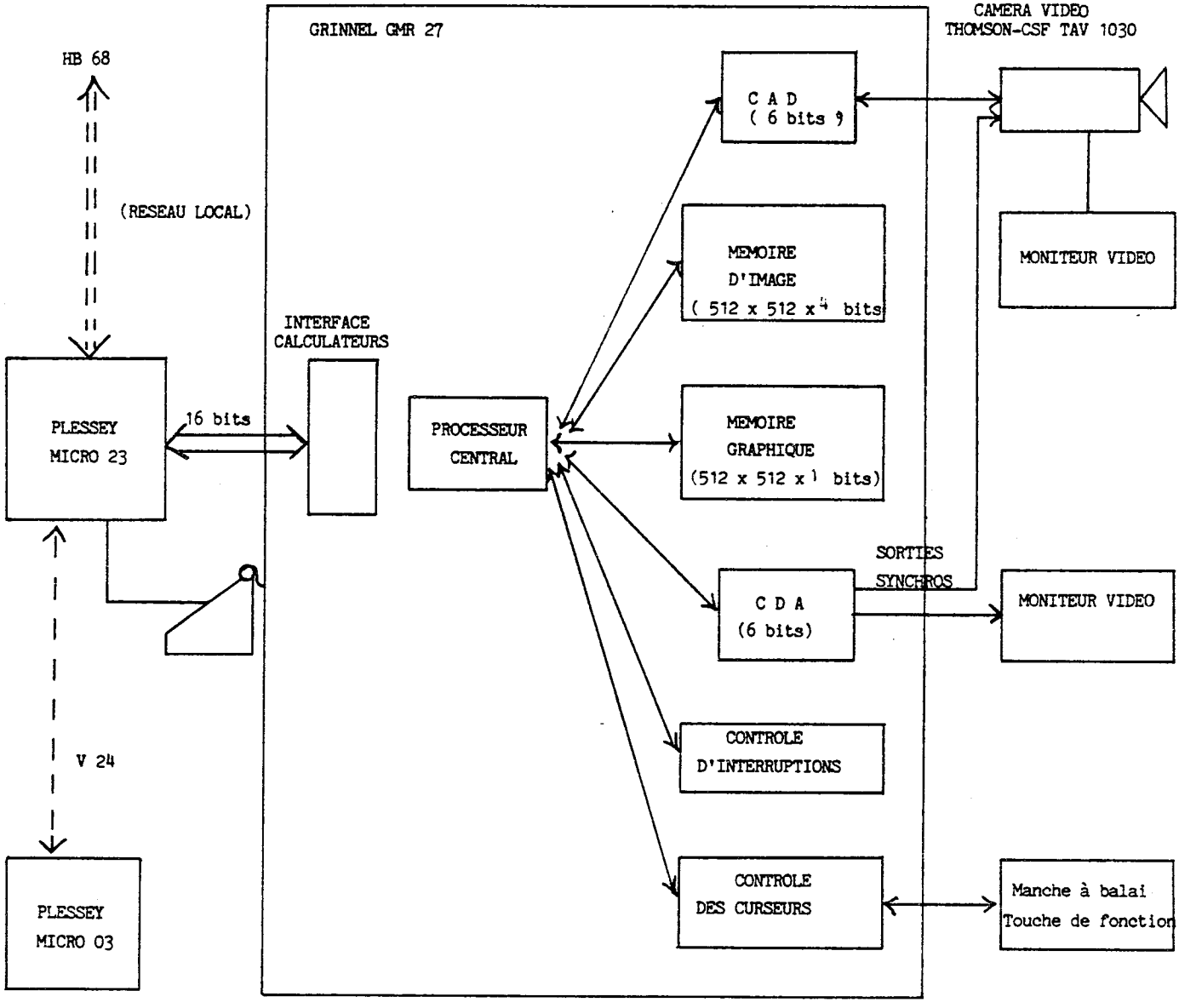

GRINNEL GMR 27
CAMERA VIDEO
THOMSON-CSF TAV 1030
HB 68
(RESEAU LOCAL)
C A D
(6 bits)
MEMOIRE
D'IMAGE
(512 x 512 x 4 bits)
MONITEUR VIDEO
INTERFACE
CALCULATEURS
PLESSEY
MICRO 23
16 bits
PROCESSEUR
CENTRAL
MEMOIRE
GRAPHIQUE
(512 x 512 x 1 bits)
SORTIES
SYNCHROS
C D A
(6 bits)
MONITEUR VIDEO
V 24
CONTROLE
D'INTERRUPTIONS
CONTROLE
DES CURSEURS
Manche à balai
Touche de fonction
PLESSEY
MICRO 03

Figure 14

Le signal analogique est digitalisé par le DAPI 16 sur 8 bits. L'image est réorganisée en mémoire centrale du SOLAR par le logiciel d'acquisition.

8. LE SYSTEME DE VISION II

Il se compose d'une caméra vidéo noir et blanc à balayage entrelacé (Thomson-CSF TAV 1030), d'un matériel GRINNELL GMR 27, d'un calculateur PLESSEY Micro 2/23 et de deux moniteurs vidéo noir et blanc (figure 14).

Le GMR 27 est constitué de plusieurs sous-ensembles : un convertisseur analogique/digital (6 bits), un convertisseur digital/analogique (6 bits), une mémoire d'image (512x512x4 bits), une mémoire graphique (512x512x1 bits) et un dispositif d'interaction (touches de fonctions et manche à balais commandant 4 curseurs indépendants). Comme le montre la figure 15, ces sous-ensembles sont gérés par un processeur central qui est relié au calculateur par une ligne parallèle de 16 bits capables d'atteindre des vitesses d'échanges d'environ 30 K Hz (mot/s).

Le calculateur peut commander un certain nombre de fonctions au GMR 27, notamment :

- la digitalisation à la volée du signal transmis par la caméra et la mémorisation immédiate du résultat en mémoire d'image (acquisition d'une image 512×512×4 bits en 1/25ème de seconde),
- l'addition et la soustraction dans la mémoire d'image de n images successives,
- la visualisation du contenu de la mémoire d'image et/ou du contenu de la mémoire graphique sur le moniteur,
- l'accès aux mémoires (image et graphique) en lecture et en écriture (1,5µs par pixel en moyenne),
- la génération (câblée) de vecteurs et de caractères alphanumériques en mémoire graphique,
- la lecture des positions des curseurs et de l'état des touches de fonction.

Le GMR 27 est provisoirement connecté au Micro 1/03. Au début de 1981, cette connexion sera remplacée par une connexion à un calculateur PLESSEY Micro 2/23 construit autour d'un LSI-11/23 de DEC. Ce calculateur sera à son tour relié à l'HB-68 du CICG par le réseau local de l'IMAG de façon à pouvoir expérimenter les modèles d'analyse de scènes développées sur cet ordinateur.

9. PERSPECTIVES D'EVOLUTION

Nous avons vu dans les paragraphes précédents que plusieurs extensions du matériel actuellement en place seront réalisées au début de l'année 1981. Notamment :

- le capteur de force sera complété de telle sorte qu'il puisse mesurer les composantes latérales de force FX et FY (cf. § 6),
- un calculateur Micro 2/23 sera installé et connecté à la sortie du système GRINNELL (cf. § 8),
- les deux Micro 1/03 et 2/23 seront reliés à l'HB-68 du CICG par le réseau local de l'IMAG.

Par ailleurs, un contrat de recherche avec la Société SCEMI prévoit la fourniture pour Mars 1981 d'un bras manipulateur à 6 degrés de liberté muni de ses asservissements de position et d'un calculateur de commande à base d'un LSI-11/23 de DEC.

CHAPITRE 5

LE LOGICIEL IMPLANTE

Dans ce chapitre, nous décrivons le logiciel que nous avons conçu et implanté. Ce logiciel est essentiellement l'interpréteur d'un langage syntaxiquement élémentaire, mais permettant d'exprimer simplement la sémantique de LM. Ce langage, que nous appelons LM-A (LM-Assembleur) est au symbolisme près celui d'une machine destinée à faciliter l'interpréteur de programmes en LM et appelée dans la suite *machine LM*.

Le logiciel implanté est destiné à donner au matériel décrit dans le chapitre précédent les qualités de programmablité nécessaires aux recherches effectuées dans le cadre du projet PANDORE.

1. VUE D'ENSEMBLE

Comme le montre le schéma de la figure 1, le logiciel implanté comporte quatre parties : un *analyseur lexicographique*, un *chargeur*, une *machine LM* et un ensemble de *primitives actionneurs-capteurs*.

L'élément principal de ce logiciel est la machine LM. Il s'agit d'une machine à pile implantée en Fortran sur le Micro 1/03 (cf. § 5). Elle interprète directement un langage composé d'instructions codées chacune sur un mot de 16 bits. Actuellement, l'intérêt de cette machine est surtout méthodologique [12]. Toutefois, on peut envisager une réalisation micro-programmée de cette machine de façon à disposer d'un processeur spécialisé capable d'interpréter plus efficacement les programmes écrits dans le langage LM.

L'analyseur lexicographique joue pour cette machine un rôle analogue à celui d'un assembleur (§ 3). Le langage qu'il accepte est le langage LM-A. Chaque instruction de la machine LM s'exprime dans ce langage par un mnémonique symbolique (§ 2).

Le rôle du langage LM-A est de séparer le traitement de la syntaxe du langage LM du traitement de sa sémantique. Nous l'avons donc conçu pour que :

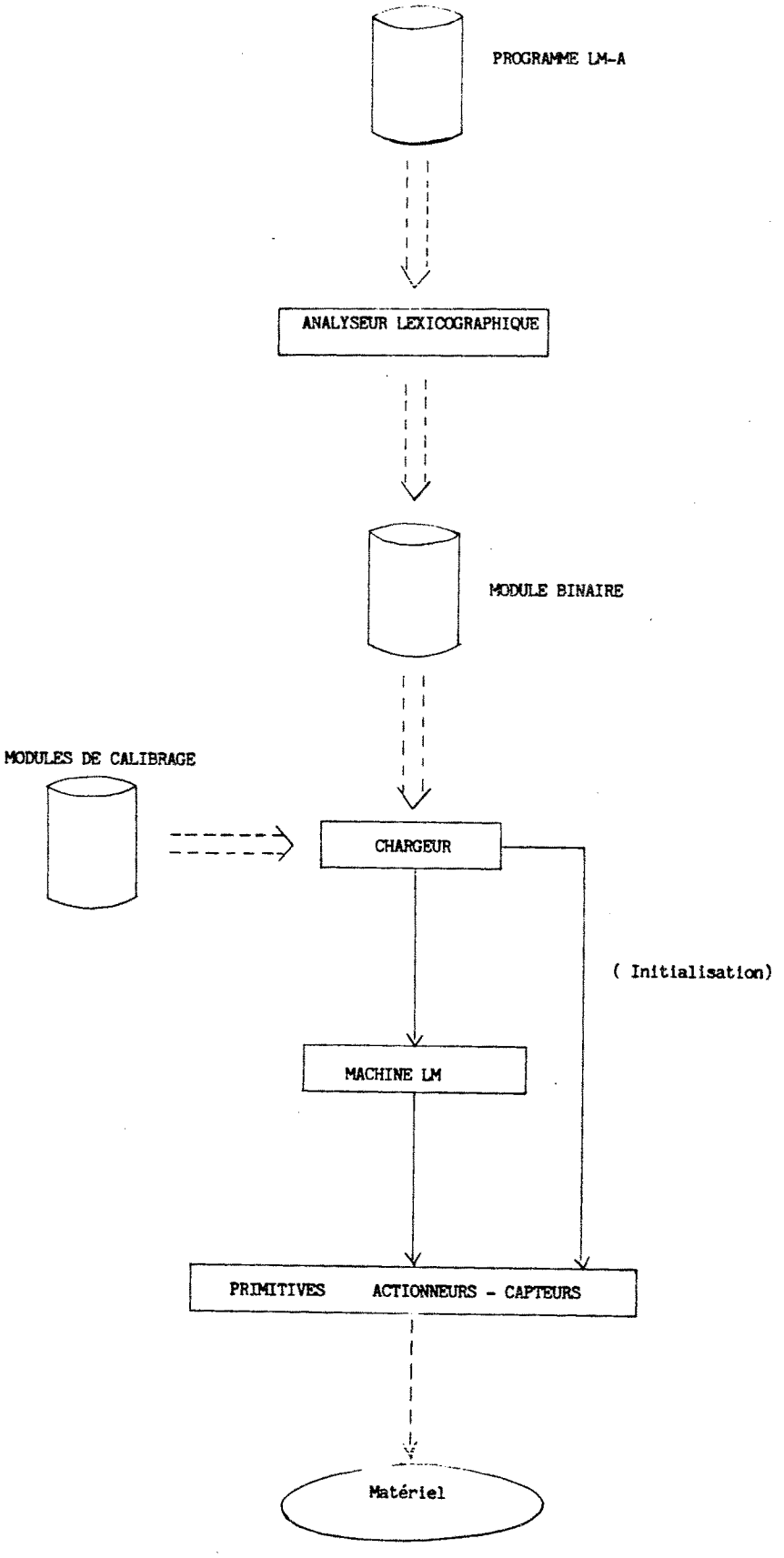

PROGRAMME LM-A
ANALYSEUR LEXICOGRAPHIQUE
MODULE BINAIRE
MODULES DE CALIBRAGE
CHARGEUR
(Initialisation)
MACHINE LM
PRIMITIVES
ACTIONNEURS - CAPTEURS
Matériel

Figure 1

- d'une part, il permette d'exprimer simplement la sémantique de LM,
- d'autre part, son implantation soit plus facile et plus efficace que celle de LM.

La machine LM interagit avec les systèmes mécaniques et les capteurs par l'intermédiaire de primitives actionneurs-capteurs. Ces primitives permettent notamment de commander des déplacements des systèmes mécaniques, de connaître leurs positions, de mesurer une composante de force et de localiser un objet à l'aide d'une caméra (§ 6). Elles constituent un interface dont le rôle principal est de dissocier la machine LM des particularités du matériel utilisé (nombre de degrés de liberté des systèmes mécaniques, nature des asservissements, ...) et des méthodes utilisées pour interpréter les données provenant de certains capteurs (caméras notamment).

Outre sa fonction classique de chargeur de programmes, le chargeur réalise aussi certaines opérations d'initialisation et de calibrage nécessaires au bon fonctionnement du matériel (§ 4).

Les programmes de l'analyseur lexicographique, du chargeur et de la machine LM sont écrits en Fortran et sont opérationnels sur le Micro 1/03 sous le système RT-11. Ceux des primitives actionneurs-capteurs sont écrits partiellement en Fortran et en Assembleur ; leur implantation est répartie sur le Micro 1/03, les systèmes Z80 et le SOLAR 16/65.

Relativement au langage LM décrit dans le chapitre 2, le logiciel actuellement opérationnel présente quelques limitations. Les principales sont les suivantes :

- le traitement des procédures en LM-A est seulement en cours d'implantation (ce qui explique l'absence d'un éditeur de lien dans la figure 1) ; il en est de même du traitement des tableaux et des chaînes de caractères ;
- le seul type de déplacement actuellement exécutable consiste à lancer simultanément et à arrêter simultanément tous les degrés de liberté concernés.

L'ensemble du logiciel implanté a été conçu en vue d'une réalisation complète d'un système de programmation LM. Cette réalisation, qui est en cours, conservera les modules existants et leur adjoindra de nouveaux modules (analyseur syntaxique, vérification sémantique, moniteur d'exécution, ...).

2. LE LANGAGE LM-A

Les différences entre les langages LM et LM-A sont essentiellement d'ordre syntaxique (cf. l'exemple de la figure 2). Pour éviter trop de répétitions avec le chapitre 2, nous ne donnons donc ci-dessous qu'une description succincte du langage LM-A. Les exemples de programmes présentés au chapitre 6 complètent cette description.

Un programme en LM-A est composé de *directives* pour l'analyseur lexicographique et d'*instructions* pour la machine LM.

Note : Les unités choisies dans le langage LM-A sont le millimètre pour les longueurs, le radian pour les angles, la seconde pour les temps, le newton pour les forces et le newton-centimètre pour les moments.

2.1. Les directives

Elles sont de quatre types :

- *Les déclarations de variables*
  Les variables (autres que les variables d'état) doivent être déclarées avec leurs types au début du programme. Les six types de données de LM sont possibles.
  Exemples : REEL FORCE HAUTEUR
  REPERE PRISE DEST REPOS

- *Les définitions d'étiquettes*
  Une étiquette ETIQ est définie par ETIQ: (concaténation du nom de l'étiquette et du caractère ":"). Elle identifie l'instruction qui suit immédiatement cette définition.

- *La directive C*
  Elle indique que le reste de la ligne est à prendre comme un commentaire.

- *La directive FIN*
  Elle provoque l'arrêt de l'analyse lexicographique.

```
REPERE DEST
REEL LONG
100.0 =LONG
BLEU REP VX !LONG TRANSLAT PRTRS =DEST
BLEU !DEST DEPL
RETOUR
FIN
```

(a) Programme en LM-A

```
REPERE DEST ;
REEL LONG ;
LONG = 100.0 ;
DEST = BLEU*TRANSLATION(VX,LONG) ;
DEPLACER BLEU A DEST ;
RETOUR ;
FIN ;
```

(b) Programme en LM

Figure 2 - Exemple de programmes équivalents en LM-A et LM

2.2. Les instructions

Les instructions du langage LM-A reflètent la structure à pile de la machine LM. En particulier, nombre d'entre elles ne contiennent pas leurs arguments ; ceux-ci doivent se trouver sur la pile au moment de l'exécution d'une telle instruction et ils en sont alors retirés.

Toutes les instructions sont exprimées par des chaînes de symboles sans blanc.

Nous distinguons les instructions classiques que l'on s'attend à trouver dans une machine à pile et les instructions plus spécifiques de la commande de manipulateurs.

a) Instructions classiques

- *Chargement d'une constante sur la pile*
  L'instruction est la constante elle-même, par exemple : 10 2.05 VRAI VX STATION (la liste des constantes est donnée dans le § 2.3).

- *Chargement sur la pile de la valeur d'une variable déclarée VAR*
  L'instruction est !VAR (concaténation du caractère "!" et du nom de la variable).

- *Chargement sur la pile de l'adresse d'une variable déclarée REP de type repère*[1]
  L'instruction est @REP.

- *Chargement sur la pile de la valeur d'une variable d'état autre qu'un repère d'état*
  L'instruction est le nom de la variable, par exemple FZ MZ ENDEPO (la liste des variables d'état est donnée dans le § 2.3).

- *Chargement de l'adresse d'un repère d'état sur la pile*[1]
  L'instruction est le nom du repère, par exemple BLEU NOIR (repères attachés aux extrémités utiles des systèmes mécaniques BLEU et NOIR).

---

(1) Ces instructions donnent une "existence" aux repères indépendamment de leurs valeurs. Elles permettent de charger sur la pile les arguments d'instructions telles que LIER, DEPL, ...

- *Chargement de la valeur d'un repère d'état sur la pile*

L'instruction est REP. Elle doit trouver l'adresse du repère d'état correspondant sur la pile. La séquence d'instructions :

```
BLEU  REP
```

permet donc de charger la valeur courante du repère d'état BLEU sur la pile.

- *Opérations sur les données*

Un certain nombre d'instructions exprimées par des mnémoniques simples permettent d'exécuter des opérations sur les données, par exemple +ENT additionne deux entiers, >REEL compare deux réels, INVTRS calcule l'inverse d'une transformation. La liste de ces instructions est donnée au § 2.3. Chacune de ces instructions doit trouver ses arguments sur la pile et range le résultat de l'opération exécutée sur la pile. Par exemple, la séquence d'instructions :

```
!REP1   !REP2   DISTANCE
```

calcule la distance entre les origines des repères REP1 et REP2. Les expressions LM doivent donc être exprimées en LM-A sous une forme postfixée. Par exemple, l'expression LM :

```
(R1+R2)*A < B
```

où R1, R2, A et B sont des réels s'expriment en LM-A par :

```
!R1  !R2  +REEL  !A  *REEL  !B  <REEL
```

- *Affectation d'une valeur à une variable déclarée VAR*

L'instruction =VAR affecte la valeur située au sommet de la pile à la variable VAR. Cette valeur est retirée de la pile.

- *Branchements*

. L'instruction $ETIQ déroute inconditionnellement l'exécution du programme à l'instruction désignée par l'étiquette ETIQ.

. L'instruction $$ETIQ déroute l'exécution du programme à l'instruction désignée par ETIQ si la valeur VRAI se trouve au sommet de la pile ; dans le cas contraire, l'exécution se poursuit séquentiellement. Par exemple, la séquence d'instructions

```
FZ  !FZMAX  >REEL  $$STOP
```

réalise un branchement à l'étiquette STOP si FZ (variable d'état) est supérieur à FZMAX.

[*Remarque : Ces deux instructions permettent d'exprimer les instructions SI, TANTQUE et PAUSE du langage LM*].

- *Entrées-sorties sur le terminal*
  . L'instruction ECRENT (resp. ECRREE, ECRBOO, ECRVEC, ECRTRS) imprime la valeur d'un entier (resp. réel, booléen, vecteur, transformation/repère) sur le terminal. Cette valeur doit se trouver sur la pile.
  . L'instruction LIRENT (resp. LIRREE, LIRBOO, LIRVEC, LIRTRS) lit la valeur d'un entier (resp. réel, booléen, vecteur, transformation/repère) sur le terminal. Elle place cette valeur sur la pile.

- *Fin de l'exécution du programme*
  L'instruction RETOUR termine l'exécution du programme.

b) Instructions spécifiques

- *Création d'une liaison entre deux repères*
  L'instruction est LIER. Elle doit trouver les adresses des repères sur la pile. Par exemple, les instructions

```
BLEU   @R1   LIER
```

  rendent solidaires les repères BLEU (repère d'état) et R1.

- *Destruction d'une liaison entre deux repères*
  L'instruction est DELIER. Elle doit trouver les adresses des repères sur la pile.

- *Déplacement d'un système mécanique*
  L'instruction est DEPL. Elle doit trouver sur la pile l'adresse du repère à déplacer et la destination du déplacement (valeur d'un repère). Par exemple, les instructions

```
BLEU          BLEU  REP  VZ  100.0  TRANSLAT  PRTRS      DEPL
```

adresse du repère à déplacer — *destination du déplacement*

  commandent de déplacer le repère BLEU à une destination se déduisant de la position courante de BLEU par une translation de 100 mm suivant l'arc Z de ce repère.
  [*Remarque : Le déplacement est effectué avec démarrages et arrêts simultanés des degrés de liberté concernés*].

- *Contrôle de la vitesse d'un déplacement*

  . L'instruction CVIT permet de choisir un coefficient de vitesse compris entre 0.1 et 1. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné et la valeur du coefficient. Celle-ci sera utilisée pour les déplacements postérieurs à l'instruction CVIT.

  . L'instruction CVITI permet de multiplier le coefficient de vitesse pour un déplacement en cours par un autre coefficient compris entre 0.1 et 1.

- *Arrêt d'un déplacement*

  L'instruction est ARRDEP. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné. Par exemple, les instructions

  NOIR ARRDEP

  arrêtent le déplacement en cours du système NOIR.

- *Attente de la fin d'un déplacement*

  L'instruction ATTDEP met l'exécution du programme en attente jusqu'à ce qu'un déplacement soit terminé. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné. Ainsi, l'instruction LM

  DEPLACER NOIR A REPOS/

  peut être traduite en LM-A par :

  NOIR !REPOS DEPL NOIR ATTDEP.

- *Action (ouverture-fermeture) d'une pince*

  L'instruction OUVR permet de commander l'ouverture-fermeture d'une pince. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné et la valeur d'un réel définissant l'écart voulu entre les deux mors de la pince. Ainsi, après l'exécution de

  BLEU 45.0 OUVR

  l'écart entre les mors de la pince de BLEU sera 45 mm.

- *Arrêt de l'action d'une pince*

  L'instruction est ARROUV. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné.

- *Attente de la fin de l'action d'une pince*

  L'instruction ATTOUV met l'exécution du programme en attente jusqu'à ce qu'une action soit terminée. Elle doit trouver sur la pile l'adresse du repère (BLEU ou NOIR) correspondant au système mécanique concerné.

- *Exécution parallèle de sous-programmes*

  Les instructions //ETIQ, où ETIQ est le nom d'une étiquette, et FIN// permettent d'exécuter des sous-programmes en parallèle avec le processus principal d'interprétation. Bien qu'elles soient plus générales, elles ont pour fonction essentielle d'exprimer la clause JUSQUA des instructions DEPLACER et ACTIONNER du langage LM.

  . L'instruction //ETIQ active un processus qui exécute les instructions situées derrière l'étiquette ETIQ. Ce processus doit se dérouler pendant le déplacement ou l'action spécifié par deux arguments que l'instruction //ETIQ prend sur la pile : l'un est l'adresse du repère associé au système mécanique concerné, l'autre est la valeur d'un entier (0 pour un déplacement, 1 pour une action de la pince).

  . L'instruction FIN// indique que le sous-programme exécuté par le processus parallèle est terminé. Elle commande la réactivation du processus pour qu'il exécute à nouveau le sous-programme à partir du même point d'entrée.

Ainsi, l'instruction LM

```
DEPLACER  BLEU  A  ICI  JUSQUA  FZ > FZMAX/
```

peut s'exprimer en LM-A par :

```
BLEU  !ICI  DEPL
BLEU  0  //TESTFZ
BLEU  ATTDEP
$SUITE
TESTFZ:  FZ  !FZMAX  <REEL  $$REACT
         BLEU  ARRDEP
         REACT:  FIN//
SUITE: ...
```

Le branchement à l'étiquette SUITE peut être évitée en plaçant le sous-programme dont l'exécution est commandée par l'instruction //TESTFZ (en italique) à la fin du programme.

[*Note : Les instructions //ETIQ et FIN// seront décrites plus précisément par leur interprétation au § 5.4*].

- *Commande manuelle des systèmes mécaniques*

L'instruction MANU donne à un opérateur le contrôle manuel des déplacements des systèmes mécaniques et des actions de leurs pinces.

- *Remise à zéro du capteur de force*

Le capteur de force utilisé est constitué d'éléments piezo-électriques et ne permet d'effectuer que des mesures dynamiques ou "quasi-statiques" (cf. § 6 Chapitre 4). L'instruction RAZF commande la remise à 0 de toutes les composantes de force et de moment mesurées par ce capteur.

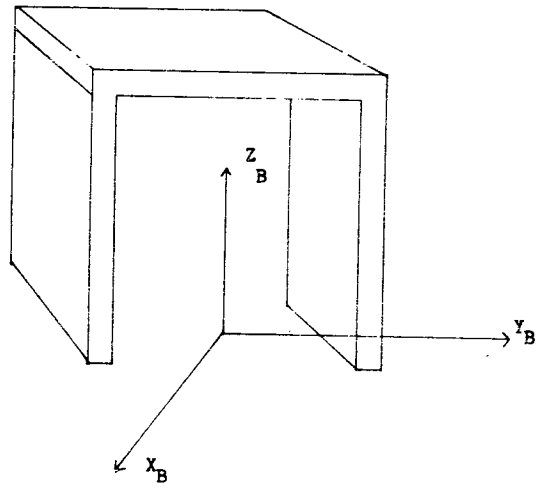
- *Localisation d'un objet à l'aide d'une caméra*

Les instructions LOCVIS1 et LOCVIS2 permettent de localiser le barycentre de la vue de dessus d'un objet. Elles doivent trouver sur la pile l'aire de cette vue (qui est supposée suffisante pour identifier l'objet). En retour, elles rangent sur la pile les coordonnées X et Y du barycentre. Si LOCVIS1 est utilisée, l'aire doit être exprimée en mm^2, et les coordonnées X et Y sont relatives au plan OXY du repère STATION. Si LOCVIS2 est utilisée, l'aire doit être exprimée en nombre de pixels, et les coordonnées X et Y sont relatives à un repère attachée à la caméra. LOCVIS2 n'est utile que pour écrire des programmes de calibrage.

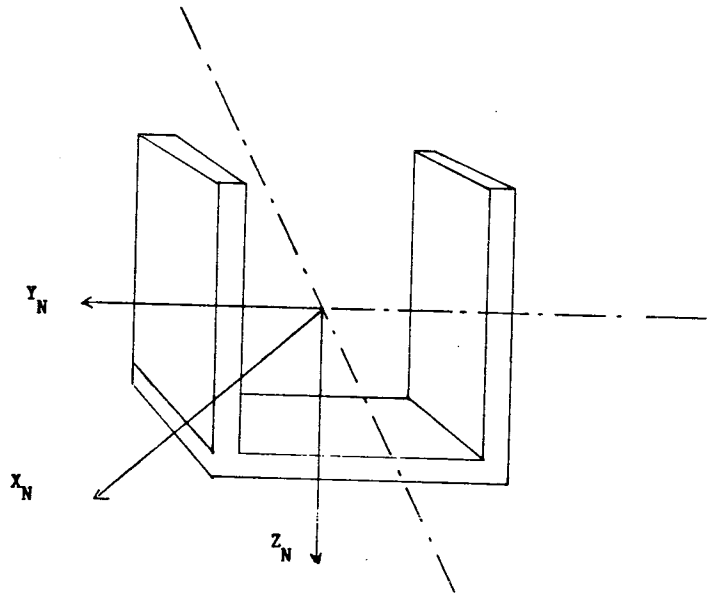
[*Note : Ces instructions sont évidemment très rudimentaires. Nous les avons implantées pour illustrer l'utilisation d'un système de vision avec le langage LM-A*].

- *Synchronisation avec des processus externes*

L'instruction ENVOI affiche une valeur logique sur l'une des sorties disponibles des systèmes Z80 (cf. § 1.3.b, Annexe B). Le numéro de la sortie (nombre entier compris entre 1 et 5), l'adresse du repère (BLEU ou NOIR) correspondant au système Z80, et la valeur logique (VRAI ou FAUX) doivent se trouver sur la pile.



POSITION DU REPERE BLEU DANS LA PINCE DU ROBOT BLEU



POSITION DU REPERE NOIR DANS LA PINCE DU ROBOT NOIR

Figure 3

- *Arrêt généralisé*

L'instruction ABORT commande l'arrêt de tout déplacement et de toute action en cours. Elle permet à l'utilisateur d'examiner le contenu de la pile de la machine LM.

2.3. Constantes, variables d'état et opérations du langage LM-A

a) Constantes

Ce sont les constantes numériques entières et réelles, PI (nombre 3.14159265), VRAI, FAUX, VX, VY, VZ et STATION.

[Dans l'implantation actuelle, le repère STATION est attaché à la base du manipulateur BLEU (il n'est donc pas défini par calibrage). Son axe Z est supporté par l'axe de la rotation R1 et est orienté vers le haut].

b) Variables d'état

- Variables d'état réelles :
  - . ECART0 (resp. ECART1) : écart des mors de la pince de BLEU (resp. NOIR),
  - . TEMPS : temps depuis le début de l'exécution du programme,
  - . TDEP0 (resp. TDEP1) : temps depuis le début du dernier déplacement de BLEU (resp. NOIR),
  - . TACT0 (resp. TACT1) : temps depuis le début de la dernière action de la pince de BLEU (resp. NOIR),
  - . FZ : composante de force suivant l'axe de la pince de BLEU,
  - . MZ : composante de moment autour de l'axe de la pince de BLEU.

- Variables d'état booléennes :
  - . ENDEP0 (resp. ENDEP1) : VRAI si BLEU (resp. NOIR) est en déplacement, FAUX sinon,
  - . ENACT0 (resp. ENACT1) : VRAI si la pince de BLEU (resp. NOIR) est en action, FAUX sinon,
  - . BOOL1,...,BOOL16 : variables sans signification imposée a priori.

- Variables d'état de type repère :
  - . BLEU (resp. NOIR) : repère attaché à l'extrémité utile du système mécanique BLEU (resp. NOIR).

  [Chacun de ces repères a été choisi de telle sorte que son origine soit située sur l'axe de la pince, l'axe Z étant supporté par l'axe de la pince et orienté vers l'intérieur de la pince, l'axe X étant perpendiculaire aux

mors (cf. figure 3)].

c) Opérations

- Opérations sur les entiers :
  +ENT (addition de deux entiers), -ENT (soustraction de deux entiers), -ENT1 (changement de signe d'un entier), *ENT (multiplication de deux entiers), /ENT (division de deux entiers), **ENT (élévation d'un entier à une puissance entière), <ENT, <=ENT, ENT=, >=ENT, >ENT (comparaison de deux entiers), ENTREEL (conversion d'un entier en un réel), VALAEN (valeur absolue d'un entier).

- Opérations sur les réels :
  +REEL (addition de deux réels), -REEL (soustraction de deux réels), -REEL1 (changement de signe d'un réel), *REEL (multiplication de deux réels), /REEL (division de deux réels), **REEL (élévation d'un réel à une puissance réelle), <REEL, <=REEL, REEL=, >=REEL, >REEL (comparaison de deux réels), REELENT (partie entière d'un réel), VALARE (valeur absolue d'un réel), SIN, COS, TAN, ARSIN, ARCOS, ARTAN (fonctions trigonométriques), EXP, LOG (fonctions exponentielle et logarithme).

- Opérations sur les booléens :
  OU, ET, NON.

- Opérations sur les vecteurs :
  GENVEC (génération d'un vecteur), XVEC, YVEC, ZVEC (extraction des composantes d'un vecteur), +VEC (addition de deux vecteurs), -VEC (soustraction de deux vecteurs), -VEC1 (changement de signe d'un vecteur), REEVEC (multiplication d'un vecteur par un réel), PSCAL (produit scalaire de deux vecteurs), PVECTO (produit vectoriel de deux vecteurs), LONG (longueur d'un vecteur), TRSVEC (application d'une transformation à un vecteur).

- Opérations sur les transformations et les repères :
  [*Note : La valeur d'un repère est la transformation qui fait passer de STATION à ce repère. Une fois sur la pile la valeur d'un repère ne se distingue donc pas de celle d'une transformation. Les opérations de LM-A sont donc les mêmes pour les transformations et les repères*].
  ROTATION (génération d'une rotation), TRANSLAT (génération d'une translation), PRTRS (produit de deux transformations), VECT.ROT (extraction du vecteur rotation), VECT.TRSL (extraction du vecteur translation),

INVTRS (inversion d'une transformation), EXTROT (extraction de la rotation), EXTTRSL (extraction de la translation), TRANSF (extraction de la transformation entre deux repères), DISTANCE (distance entre les origines de deux repères), ANGLE (distance angulaire entre deux repères).

3. L'ANALYSEUR LEXICOGRAPHIQUE

L'analyseur lexicographique transforme un programme en LM-A en un module numérique. La transformation est effectuée en une seule passe de gauche à droite, la résolution de certaines étiquettes étant éventuellement réalisée à la fin de l'analyse.

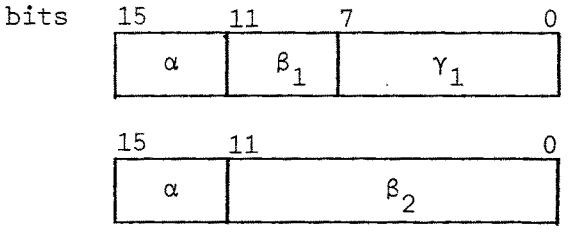
Le module engendré comporte trois zones :

(1) La *zone d'entête*, qui contient les informations nécessaires au chargeur : nombre d'instructions, nombre de variables locales de chaque type, ...

(2) La *zone de programme*, qui contient les instructions pour la machine LM.

(3) La *zone des constantes*, qui contient (sans répétition) les constantes entières et réelles figurant dans le programme source.

A chaque instruction du programme en LM-A, l'analyseur lexicographique fait correspondre une instruction unique de la machine LM. Cette instruction est codée sur un mot de 16 bits décomposé en champs suivant deux formats possibles :



- Une instruction LM-A exprimée par une constante entière ou réelle, ou de la forme !VAR, @VAR ou =VAR est transformée en une instruction à 3 champs α, β_1 et γ_1 :
  . α code le type de l'instruction (chargement d'une donnée sur la pile, chargement de l'adresse d'un repère sur la pile, déchargement du sommet de pile dans une variable),

. β_1 spécifie les caractéristiques de la donnée manipulée (constante entière, constante réelle, variable entière, variable réelle, variable booléenne, ...),

. γ_1 détermine l'adresse de la donnée relativement à une base qui sera connue au chargement.

[*Note : Dans le cas d'une constante, γ_1 est le numéro d'apparition de la constante dans la zone des constantes. Dans le cas d'une variable, il s'agit du numéro d'apparition de la variable dans les déclarations parmi les variables du même type. L'analyseur lexicographique est ainsi indépendant du format des données*].

- Une instruction LM-A de la forme $ETIQ, $$ETIQ ou //ETIQ est traduite en une instruction à 2 champs α et β_2 :
  . α code le type de l'instruction (branchement inconditionnel, branchement conditionnel, exécution parallèle d'une séquence d'instructions),
  . β_2 code le déplacement (positif ou négatif) en mots qui sépare cette instruction de l'instruction repérée par l'étiquette ETIQ.

- Toutes les autres instructions sont traduites en une instruction à 3 champs α, β_1 et γ_1. L'instruction engendrée ne contenant aucun paramètre, les trois champs sont utilisés pour spécifier le type de l'instruction.

L'analyseur lexicographique a été implanté en Fortran sous le système RT-11 du Micro 1/03. Le traitement des chaînes de caractères utilise des fonctions appropriées de la bibliothèque FORTRAN (SYSLIB) de RT-11.

4. LE CHARGEUR

Il réalise quatre fonctions principales.

a) Initialisation des systèmes BLEU et NOIR

Les systèmes mécaniques BLEU et NOIR ne sont pas munis de capteurs de position. Leurs positions respectives sont connues relativement à des positions initiales par comptage des impulsions envoyées aux moteurs.

Lors de la mise en route du matériel, les positions absolues de ces systèmes ne sont donc connues d'aucun programme. C'est pourquoi le chargeut fait appel à une primitive actionneur-capteur qui utilise les capteurs de fin

de course comme références pour donner à chaque degré de liberté des systèmes BLEU et NOIR, y compris leurs pinces, des positions initiales bien déterminées (cf. § 6.1).

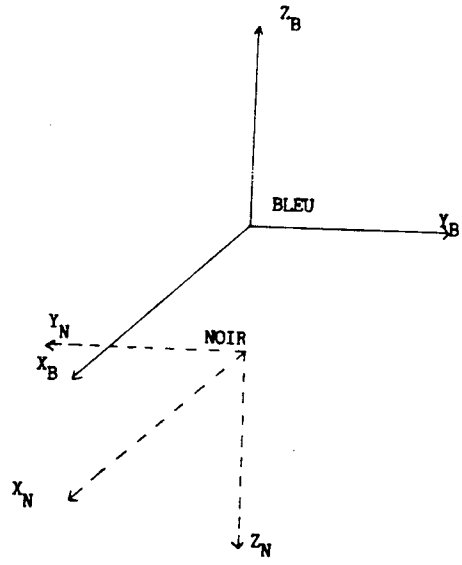
L'initialisation de chaque système est facultative. L'utilisateur peut l'éviter s'il estime qu'il se trouve déjà dans sa position initiale. Les programmes font alors l'hypothèse de cette position.

b) Calibrage de NOIR

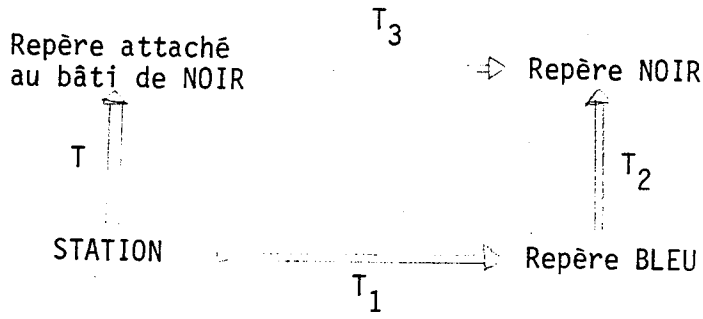
La machine LM représente les positions de tous les repères, y compris celle du repère NOIR, relativement au repère STATION (qui est attaché au bâti du manipulateur BLEU, cf. § 2.3). Pour effectuer les changements de coordonnées nécessaires à la commande du système NOIR et au calcul de sa position à chaque instant, elle utilise un repère implicite attaché au bâti de NOIR (cf. Chapitre 3, § 3.2). Elle doit donc connaître la transformation qui définit la position de ce repère relativement à STATION.

En position initiale, l'axe Z du repère NOIR est vertical et dirigé vers le bas. Le calibrage de NOIR consiste à donner le contrôle manuel du manipulateur BLEU à l'utilisateur. Celui-ci doit positionner la pince de BLEU de telle sorte que les repères BLEU et NOIR soient comme le montre la figure 4.a, puis rendre le contrôle au chargeur. La connaissance des transformations T1, T2 et T3 (cf. Figure 4.b) permet au chargeur de déterminer les paramètres de la transformation T liant STATION au repère attaché au bâti du système NOIR.

Le calibrage de NOIR est facultatif. L'utilisateur peut l'éviter si NOIR n'est pas utilisé. Il peut aussi choisir d'entrer les paramètres du calibrage sur le terminal (cette possibilité est utile lorsque la position relative des systèmes BLEU et NOIR n'a pas été modifiée depuis le dernier calibrage).



(a)



$$T \circ T_3 = T_1 \circ T_2 \Rightarrow T = T_1 \circ T_2 \circ T_3^{-1}$$

(b)

Figure 4 - Calibrage de NOIR

c) Calibrage du système de vision I

Le système de vision I peut être utilisé pour localiser des objets dans un espace bidimensionnel d'après leurs vues de dessus. La caméra CCD est placée à la verticale au dessus du plan de travail de BLEU et un repère ΩUV lui est implicitement attaché. Ce repère est parallèle au plan OXY du repère STATION. La localisation d'un objet dans STATION nécessite de connaître la transformation dans R^2 qui fait passer de ΩUV à OXY.

Le calibrage du système de vision I est réalisé en déposant (sous le contrôle de l'utilisateur) un objet cylindrique à deux endroits différents dans le champ de vision de la caméra CCD. Les coordonnées du barycentre (supposé correspondre au centre de la pince) sont déterminées dans le plan OXY de STATION (positions de la pince lors des déposes de l'objet) et dans le repère ΩUV de la caméra. On obtient ainsi des relations desquelles on extrait les quatre paramètres du calibrage :
- les coordonnées de Ω dans OXY,
- l'angle de ΩV avec OX,
- le facteur d'échelle entre ΩUV et OXY.

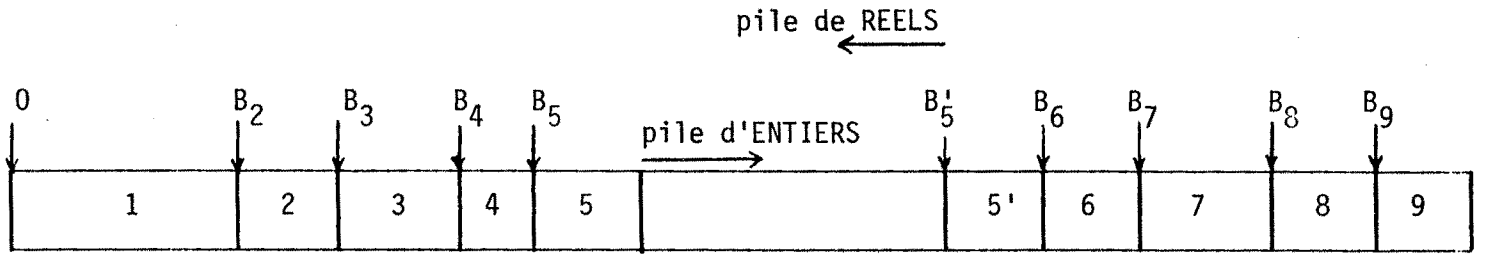
Le calibrage du système de vision I est facultatif. L'utilisateur peut l'éviter si ce système n'est pas utilisé. Il peut aussi choisir d'entrer les paramètres du calibrage sur le terminal.

Remarques :

- *Les calibrages sont réalisés par des programmes écrits en LM-A et transformés par l'analyseur lexicographique. Il est ainsi facile de les modifier.*
- *Les calibrages nécessitent des déplacements du manipulateur BLEU. Le chargeur remet celui-ci dans sa position initiale.*
- *Le logiciel actuellement opérationnel ne permet pas d'utiliser le système de vision II.*

d) Chargement du module utilisateur

Le chargeur utilise les données contenues dans le module numérique produit par l'analyseur lexicographique pour structurer une zone de mémoire suivant le modèle de la figure 5 et pour en remplir certaines parties



1 Programme

2 Constantes entières

3 Variables entières

4 Variables booléennes

5 et 5' Variables repères

6 Variables transformations

7 Variables vecteurs

8 Variables réelles

9 Constantes réelles

B_2 : base des constantes entières

B_3 : base des variables entières

B_4 : base des variables booléennes

B_5 : base des variables repères (zone entière)

$B_{5'}$: base des variables repères (zone réelle)

B_6 : base des variables transformations

B_7 : base des variables vecteurs

B_8 : base des variables réelles

B_9 : base des constantes réelles

Figure 5 - Structuration de la mémoire de la machine LM

(programme, constantes entières et réelles). La structuration de cette zone sera explicitée au § 5.2.

D'autre part, le chargeur donne une valeur à chacune des bases d'adressage nécessaires à la machine LM.

5. LA MACHINE LM

Les principaux éléments de la machine LM ont été présentés implicitement au chapitre 3. Nous ne décrivons donc ci-dessous que la simulation de cette machine en Fortran sous le système RT-11 du Micro 1/03.

Nous notons ENTIER (resp. REEL) un entier (resp. un réel) Fortran.

5.1. Représentation des données

Toutes les données, quels que soient leurs types, sont représentées en termes d'ENTIERS (1 mot) et de REELS (2 mots).

- Un *entier* est représenté par un ENTIER.
- Un *réel* est représenté par un REEL.
- Un *booléen* est représenté par un ENTIER.
  [*Note : On pourrait envisager de représenter jusqu'à 16 booléens avec un même ENTIER, moyennant un adressage plus complexe des données booléennes*].
- Un *vecteur* est représenté par 3 REELS consécutifs (les composantes du vecteur).
- Une *transformation* est représentée par 12 REELS consécutifs : les 9 premiers représentent les éléments de la matrice de rotation ; les 3 suivants représentent le vecteur de translation.
  [*Note : La représentation d'une rotation par 9 REELS est redondante (cf.§ 1.1.b,Chapitre 3). Elle permet toutefois d'utiliser directement les règles de calcul du Chapitre 3*].
- Un *repère* est représenté par 2+N ENTIERS consécutifs et par 12 REELS consécutifs ; pour un repère R appartenant à un solide S :

. le 1er ENTIER contient l'adresse absolue du bloc d'ENTIERS composant la représentation du repère principal de S (cf. § 2.1 Chapitre 3),
. le 2ème ENTIER contient l'adresse absolue du bloc de REELS de R,
. les N ENTIERS suivants servent à coder les adresses absolues des blocs d'ENTIERS composant les représentations des repères directement liés à R (dans l'implantation actuelle N = 10),
. les 12 REELS représentent la position de R relativement à STATION si R est le repère principal de S, et relativement au repère principal de S dans le cas contraire.

[*Note : Les 2+N ENTIERS sont initialisés par le chargeur*].

5.2. Structuration de la mémoire

La mémoire de la machine LM est implantée par un tableau d'ENTIERS (ZENTIER) et un tableau de REELS (ZREEL). Grâce à la directive Fortran EQUIVALENCE, ces deux tableaux recouvrent la même zone de mémoire du Micro 1/03. Dans l'implantation actuelle, celle-ci comporte 5000 mots.

Cette mémoire est structurée suivant le modèle de la figure 5. Les ENTIERS occupent sa partie gauche et les REELS sa partie droite. La zone intermédiaire (hachurée) est utilisée pour gérer une pile d'ENTIERS (à gauche) et une pile de REELS (à droite). Ces deux piles réalisent la pile vue par l'utilisateur du langage LM-A. Par exemple, le chargement de la valeur d'un repère sur la pile correspond à charger 12 REELS sur la pile des REELS. Initialement les deux piles sont vides.

L'adresse A d'une donnée dans cette mémoire est l'adresse du mot le plus à gauche qu'elle occupe. Cette donnée est accédée à travers le tableau ZENTIER s'il s'agit d'un ENTIER ou d'un ensemble d'ENTIERS ; l'indice du tableau est alors A+1. Elle est accédée à travers le tableau ZREEL s'il s'agit d'un REEL ou d'un ensemble de REELS ; l'indice du tableau est alors $\frac{A}{2}+1$.

Dans certaines instructions à trois champs (cf. § 3), le champ β_1 détermine la base d'adressage B et la longueur L d'une donnée ; le champ γ_1 fournit son numéro d'apparition N dans la mémoire à partir de cette base. L'adresse A de la donnée se calcule alors par A = B+L(N-1). Si la donnée est un repère,

l'adresse calculée est celle du bloc d'ENTIERS qui compose la représentation du repère (l'adresse du bloc de REELS étant contenue dans ce bloc).

Aucune place n'est réservée dans la mémoire pour les variables d'état. Leurs valeurs sont obtenues en exécutant des primitives actionneurs-capteurs. L'"adresse" d'un repère d'état (qui est rangée sur la pile par une instruction LM-A de la forme @REP) est un ENTIER permettant d'identifier la primitive correspondante.

5.3. Les instructions

Au § 3, nous avons vu que toutes les instructions de la machine LM sont codées sur un mot de 16 bits décomposé en 2 ou 3 champs. Nous avons également présenté comment chaque instruction LM-A est traduite en une instruction de la machine LM.

Toutes les instructions sont interprétées par des programmes écrits en FORTRAN. Pour l'essentiel, cette interprétation suit les règles opératoires et les algorithmes décrits au Chapitre 3. Elle ne pose aucune difficulté particulière sauf en ce qui concerne les instructions de la forme //ETIQ et FIN//. Ces instructions, qui mettent en jeu des processus parallèles, sont étudiées au paragraphe suivant.

5.4. Structure de la machine

Les instructions LM-A de la forme //ETIQ permettent d'activer des processus se déroulant en parallèle avec le processus principal d'interprétation. Chacun de ces processus est associé à un déplacement ou à une action de l'un des systèmes mécaniques BLEU et NOIR, et ne doit se dérouler que pendant ce déplacement ou cette action.

L'implantation de ces processus utilise la fonction ITIMER du système RT-11. Cette fonction, appelable par un programme FORTRAN, permet de commander l'exécution d'un sous-programme FORTRAN quelconque (appelé "completion routine") sur une interruption provoquée par un signal d'horloge à venir.

Plus précisément, quatre variables DRAP(i), i=1 à 4, sont associées respectivement aux déplacements et aux actions de BLEU et de NOIR. Par exemple, DRAP(1) est associée aux déplacements de BLEU. Chacune de ces variables est initialisée à 0.

L'interprétation de la séquence d'instructions LM-A :

BLEU 0 //ETIQ

consiste pour la machine LM à donner la valeur 1 à DRAP(1) et à mémoriser l'adresse de l'instruction identifiée par ETIQ.

Le programme principal de la machine LM est un programme itératif (cf. Figure 6). A chaque itération, il commence par interpréter une instruction du programme LM-A (correspondant au processus principal d'interprétation), puis il compare chaque variable DRAP(i) à 1. Si, par exemple, DRAP(1)=1, il redonne la valeur 0 à DRAP(1), puis, si un déplacement de BLEU est en cours, il exécute le sous-programme placé derrière l'étiquette ETIQ jusqu'à la rencontre d'une instruction FIN//. L'interprétation de cette instruction consiste à appeler la fonction ITIMER pour commander l'exécution du sous-programme SCHED1 1/50ème de seconde plus tard. L'exécution de SCHED1 a pour effet de mettre DRAP(1) à 1.

On peut ainsi considérer que la machine LM exécute cinq processus : un processus principal d'interprétation et quatre processus auxiliaires. Dans l'implantation réalisée, le processus principal ne peut être interrompu qu'entre les exécutions de deux instructions. Chacun des processus auxiliaires n'est interrompu que lorsque le sous-programme correspondant a été complètement exécuté ; sa réactivation n'est pas possible avant 1/50ème de seconde.

Ces cinq processus travaillent sur les mêmes piles d'ENTIERS et de REELS. En effet, l'exécution d'un sous-programme sémantiquement correct laisse ces deux piles dans leur état initial.

Remarques :

- *Cette implantation ne permet d'exécuter "simultanément" qu'un seul sous-programme en parallèle avec un même déplacement ou une même action. On peut toutefois changer ce sous-programme au cours d'un même déplacement par une nouvelle instruction //....*

- *Pour que l'interprétation des instructions ATTDEP et ATTACT ne bloquent pas le processus principal d'interprétation, leur interprétation ne fait appel aux primitives actionneurs-capteurs correspondantes qu'une seule fois. Les paramètres des instructions sont replacés sur la pile et le compteur de programme n'est pas incrémenté. Ainsi, entre deux appels aux primitives, les processus auxiliaires peuvent être activés.*
- *Avant de lancer un nouveau déplacement ou une nouvelle action, l'implantation de la machine LM fait appel à une primitive de RT-11 (ICMKT) pour annuler la commande ITIMER éventuelle relative à un déplacement antérieur du même système mécanique ou à une action de ce système. On s'assure ainsi que les effets d'une instruction //ETIQ ne se propagent pas d'un déplacement (ou d'une action) à un autre.*

6. LES PRIMITIVES ACTIONNEURS-CAPTEURS

Le rôle des primitives actionneurs-capteurs est de dissocier la machine LM des particularités du matériel utilisé et des méthodes qui servent à interpréter les données fournies par les capteurs. Elles contribuent donc à la portabilité de la machine LM.

L'implantation de ces primitives est répartie sur le Micro 1/03, les systèmes Z80 et le SOLAR 16/65. Un grand nombre d'entre elles font appel à des commandes interprétées sur le système Z80. Une description plus détaillée de ces commandes figurent dans l'Annexe B.

6.1. Initialisation des systèmes mécaniques BLEU et NOIR

Deux primitives permettent d'initialiser respectivement les systèmes mécaniques BLEU et NOIR (degrés de liberté et pinces).

L'initialisation de chaque système se fait séquentiellement actionneur par actionneur. Chaque actionneur est commandé en vitesse d'arrêt immédiat avec un sens de rotation fixé a priori jusqu'au changement d'état du contact de fin de course correspondant. La position d'arrêt (connue a priori) est prise comme position de référence. Chaque degré de liberté est alors initialisé à égale distance entre les deux fins de course. La pince est initialisée en position fermée (mors jointifs).

L'initialisation de chaque degré de liberté ou pince est effectuée en appelant la commande INIT implantée sur Z80 (cf. Annexe B).

6.2. Déplacements de BLEU et NOIR

La primitive de déplacement du système mécanique BLEU (resp. NOIR) reçoit en paramètre la transformation qui définit la destination du repère BLEU (resp. NOIR) relativement à sa position courante. Elle effectue les traitements suivants :

- vérification de la compatibilité de la transformation avec les degrés de liberté du système mécanique (indépendamment des butées),
- changement de coordonnées pour déterminer le déplacement à réaliser sur chaque degré de liberté,
- contrôle des butées (des butées "logicielles" plus restrictives que les butées matérielles ont été définies pour doubler les sécurités assurées par les contacts de fin de course),
- calcul des vitesses pour synchroniser les déplacements de chaque degré de liberté sur le déplacement le plus long (le calcul néglige les rampes d'accélération),
- appel des commandes Z80 DEP (description des mouvements des actionneurs concernés) et LANDEP (lancement des mouvements).

Le contrôle est rendu à la machine LM immédiatement après l'appel de la commande LANDEP.

6.3. Ouverture-fermeture des pinces des systèmes mécaniques

La primitive d'ouverture-fermeture d'une pince reçoit l'écart des pinces à réaliser. Elle vérifie que celui-ci ne dépasse pas une valeur maximale (butée logicielle), puis fait appel aux commandes Z80 ACT (description du mouvement de l'actionneur concerné) et LANACT (lancement du mouvement).

Le contrôle est rendu à la machine LM immédiatement après l'appel de la commande LANACT.

6.4. Tests de déplacement ou d'ouverture-fermeture

Pour chaque système mécanique, il existe deux primitives permettant de savoir si ce système est en déplacement et si son outil est en action. Ces deux primitives font appel aux commandes Z80 ENDEP et ENACT.

[*Remarque : Il conviendrait de distinguer l'arrêt logique d'un actionneur (lorsque l'actionneur ne reçoit plus d'impulsions du système Z80) et l'arrêt physique du moteur correspondant. Compte tenu des rampes d'accélération et de décélération éventuelles, le premier précède en général le second. Le matériel mis en oeuvre ne détecte pas la fin de l'envoi des pas aux moteurs, de telle sorte que les systèmes Z80 ne peuvent déterminer que leurs arrêt logiques. Pour tenir compte des rampes, ces systèmes introduisent une temporisation après l'arrêt logique d'un moteur, avant de le considérer comme physiquement arrêté. Cette temporisation est réglable par la commande Z80 RETARD. Elle est déterminée par les primitives de déplacements et d'actions des pinces en fonction du coefficient de vitesse courant*].

6.5. Détermination des positions des repères BLEU et NOIR

La primitive qui détermine la position du repère BLEU (resp. NOIR) fait appel aux commandes Z80 suivantes :

- ARRDEP qui arrête l'envoi des pas aux actionneurs correspondants,
- POSI (pour chaque degré de liberté concerné) qui fournit la distance ou l'angle restant à faire (0 si aucun déplacement n'est en cours) sur le degré de liberté interrogé,
- LANDEP qui réactive l'envoi des pas aux actionneurs.

A partir des informations fournies par POSI et d'informations mémorisées avant chaque déplacement, la primitive détermine la position de chacun des degrés de liberté du système considéré, puis réalise le changement de coordonnées qui calcule la transformation entre les repères STATION et BLEU (resp. entre le repère attaché à la base du système mécanique NOIR et le repère NOIR).

[*Note : POSI fournit des informations sur la base du nombre d'impulsions restant à envoyer aux actionneurs concernés. Compte tenu de rampes de décélération éventuelles, les positions calculées sont donc celles que les systèmes mécaniques atteindraient si on arrêtait immédiatement l'envoi des impulsions*].

6.6. Détermination de l'écart entre les mors d'une pince

La primitive qui détermine l'écart entre les mors d'une pince fait appel à la commande Z80 POSI pour l'actionneur concerné. Elle utilise l'information fournie par cette commande (distance restant à parcourir) et des informations mémorisées avant chaque action de la pince pour calculer l'écart courant entre les mors de la pince.

6.7. Arrêt d'un déplacement

La primitive qui permet d'arrêter un déplacement du système BLEU (resp. NOIR) en cours fait appel aux commandes Z80 ARRDEP (cf. § 6.5), POSI (pour chaque degré de liberté concerné afin de mettre à jour la position courante du système articulé considéré) et RAZDEP (qui met à 0 le nombre de pas restant à envoyer à chaque actionneur concerné).

[*Note : Compte tenu des rampes de décélération, l'arrêt peut ne pas être immédiat*].

6.8. Arrêt de l'action d'une pince

Les primitives correspondantes font appel aux commandes Z80 ARRACT, POSI (pour l'actionneur concerné) et RAZACT (qui met à 0 le nombre d'impulsions restant à envoyer à cet actionneur).

6.9. Changement de la vitesse d'un déplacement

Les primitives correspondantes appellent la commande Z80 VI pour chaque actionneur concerné avec la nouvelle vitesse souhaitée comme paramètre. L'effet est de modifier instantanément la fréquence d'envoi des impulsions aux actionneurs.

6.10. Passage en commande manuelle

La primitive de passage en commande manuelle fait appel à la commande Z80 MANUEL sur chaque système Z80. En retour, elle met à jour les positions courantes des systèmes mécaniques.

6.11. Utilisation de la caméra CCD

La primitive correspondante fait appel à un programme[1] implanté sur le SOLAR 16/65 qui "binarise" l'image en provenance de la caméra CCD par comparaison avec un seuil adapté à l'intensité lumineuse moyenne de l'image, puis effectue une analyse de connexité. Elle fournit à la machine LM la surface (nombre de "pixels") des tâches noires (ou blanches) et détermine les coordonnées de leurs barycentres dans le système de coordonnées de la caméra.

6.12. Utilisation du capteur de force

Cette primitive commande la conversion analogique-digitale de la tension sur la voie correspondant à la composante de force ou de moment choisie. Elle convertit ensuite cette tension en NEWTON ou en NEWTON-CM.

6.13. Lecture des variables d'état booléennes BOOLi (i= 1 à 16)

La primitive de lecture de ces variables fait appel à la commande Z80 SIGNAL avec le numéro de la variable en argument.

6.14. Synchronisation avec d'autres processus

La primitive correspondante est l'appel de la commande Z80 CONTROL avec un entier 0 ou 1 et un entier entre 1 et 10 comme arguments. L'effet est de mettre à la valeur logique 0 ou 1 une sortie binaire disponible sur le système Z80 concerné.

---

[1] *Réalisé par A. LUX.*

7. PERSPECTIVES D'EVOLUTION

Deux développements de l'implantation de LM sont en préparation :

- L'un consiste à réaliser un système de programmation LM complet incluant des modules d'analyse syntaxique et de vérification sémantique, ainsi qu'un moniteur d'exécution (cf. § 1).
- L'autre consiste à adapter ce logiciel au manipulateur SCEMI, en ajoutant notamment les divers calculs de trajectoires prévus aux chapitres 2 et 3.

CHAPITRE 6

EXPERIMENTATION

La mise au point du matériel et du logiciel s'est faite par étapes. Nous soulignons dans ce chapitre les difficultés que nous avons rencontrées, les erreurs commises, ainsi que les améliorations possibles du système. Nous présentons deux exemples de manipulation écrits dans le langage LM.A et réalisés par le manipulateur BLEU. De nombreux autres programmes ont été expérimentés sur ce manipulateur et nous ont aidés à mettre au point l'ensemble du système.

1. NOTES SUR LE MATERIEL MIS EN OEUVRE

1.1. Le manipulateur BLEU

La principale difficulté que nous ayons rencontrée sur ce matériel est le réglage des rampes d'accélération pour les moteurs pas à pas. Le réglage de ces rampes se fait en positionnant des interrupteurs situés dans les baies des actionneurs. Elles ne sont donc pas programmables et il est difficile de les régler d'une façon optimale pour tous les types de déplacements.

Plusieurs constructeurs offrent actuellement sur le marché des circuits intégrés de commande pour moteurs pas à pas incluant la possibilité de faire varier les rampes d'accélération par programme. L'utilisation de tels circuits rendrait beaucoup plus souple la commande des déplacements du manipulateur.

Actuellement, des problèmes de parasites et de perte de pas en cours de mouvement subsistent, mais ne nous semblent pas insurmontables.

Par ailleurs, la partie mécanique conçue pour être rigide souffre en fait d'un jeu non négligeable. Ceci, ajouté au manque de souplesse de la commande de moteurs, ne permet pas en général de réaliser de petits mouvements sans faire trembler l'extrémité de la pince.

Enfin, la pince du manipulateur prévue pour supporter une masse de 50N produit un effort de serrage insuffisant pour maintenir une telle masse entre les deux mors (notamment au moment des accélérations). Nous pensons

que ces difficultés ne sont pas inhérentes au manipulateur, mais que la pince à deux mors reste un outil de préhension très élémentaire. Une amélioration pourrait sans doute être apportée en utilisant des surfaces de contact plus rugueuses.

1.2. Le manipulateur NOIR

Bien que le logiciel et l'électronique de commande du manipulateur NOIR aient été entièrement réalisés, un mauvais appariement des moteurs et de l'électronique de puissance ne nous a pas permis de faire fonctionner la partie mécanique en régime normal. Ce matériel fait l'objet d'une étude complémentaire de la part des constructeurs des parties mécanique et électronique.

1.3. Le capteur de force

Le capteur de force ne nous a posé aucun problème de mise en oeuvre. Il apparaît comme entièrement fiable. Le seul problème que nous ayons à signaler sur ce capteur est la présence d'une dérive un peu trop forte de la mesure de la composante FZ. La diminution de cette dérive devrait être possible par un changement des cables de transport des charges.

1.4. La caméra CCD

Une très longue mise au point a été nécessaire pour faire fonctionner cette caméra, tant au niveau de l'acquisition de l'image que de l'adaptation des signaux analogiques. Un aspect important de la mise au point a été l'acquisition d'une optique munie d'un diaphragme. On peut, grâce à ce dispositif, régler facilement les conditions d'éclairage de la matrice CCD et procéder au réglage de l'électronique.

1.5. Les connexions

Dans un système complexe, les connexions ne doivent pas apparaître comme des problèmes à résoudre au moment de la réalisation. Une attitude contraire nous a conduits à passer beaucoup de temps à les mettre au point. Une réflexion approfondie sur le type de liaison et le choix du matériel nous paraît donc indispensable au moment de la conception.

2. NOTES SUR LE LOGICIEL

Le logiciel d'interprétation du langage LM-A a été implanté en quatre mois par J.C. LATOMBE, J.F. MIRIBEL et moi-même. La structure de la machine LM a facilité ce travail d'équipe.

Lors de la mise au point du logiciel, les principales difficultés auxquelles nous nous sommes heurtés ont été les configurations critiques des tâches s'exécutant sur les différents sytèmes (Micro 1-03, Z80). Chacun de ces systèmes exécutant plusieurs tâches de façon asynchrone, il était difficile d'isoler les erreurs et donc de les analyser.

De plus, la faiblesse de la mémoire de masse utilisée (deux disquettes simple face - simple densité) nous a posé des problèmes pour réaliser l'interpréteur au moment de l'édition de lien.

3. EXEMPLES DE PROGRAMME ECRIT EN LM-A

3.1. Edition et lancement d'un programme LM-A

L'édition d'un programme en LM-A se fait grâce à l'éditeur du système RT-11 TECO. L'analyse lexicographique et l'exécution du code numérique peuvent être enchaînées sans création d'un fichier intermédiaire par l'utilisateur. La mise au point de programme est donc assez facile.

Nous donnons un exemple commenté des commandes à faire pour exécuter un programme écrit en LM-A.

R.ANA	c activation de l'analyseur lexicographique
nom de fichier	c le suffixe .LMS pour LM source est pris par défaut
EXEC? oui	c demande l'exécution du module engendré
	c non : création d'un fichier contenant le code numérique
	c les messages suivants sont engendrés par l'interpréteur LM
INIT ROBITRON? oui	c commande l'initialisation du manipulateur
INIT CAMERA? oui	c commande le calibrage de la caméra
INIT NOIR? oui	c commande le calibrage de NOIR
CAPTEUR DE FIN DE COURSE? oui	c le système demande si les sécurités sont
	c actives et lance le programme

3.2. Programme d'utilisation du capteur de force

L'exemple 1 de programme permet à l'utilisateur de déplacer le manipulateur en utilisant le capteur de force. En exerçant une poussée sous la pince, l'utilisateur peut faire monter le manipulateur. De même, en tirant sur la pince, il le fait descendre. Dans les deux cas, le mouvement s'arrête en l'absence de force.

```
 TYPE FORCEZ.LMS
        REPERE MAXS INF
        Ø.3 BLEU CVIT
        BLEU REP VZ 175.Ø TRANSLAT PRTRS =MAXS
        BLEU REP VZ -175.Ø TRANSLAT PRTRS =INF
BOUC1:  FZ 5.Ø <REEL $$BOUC2
        !MAXS BLEU DEPL Ø.2 TEMPO
TOTO1:  FZ 5.Ø >REEL $$TOTO1
        BLEU ARRDEP Ø.5 TEMPO RAZF
BOUC2: FZ -5.Ø >REEL $$BOUC1
        !INF BLEU DEPL Ø.2 TEMPO
TOTO2:  FZ -5.Ø <REEL $$TOTO2
        BLEU ARRDEP Ø.5 TEMPO RAZF
        $BOUC1
        FIN
```

Exemple 1

Nous donnons dans l'exemple 2 un programme équivalent en LM.

```
            REEL TO ;
            REPERE MAXS ;
            REPERE INF ;
            DEBUT
            MAXS = BLEU * TRANSLATION(VZ, 175.0) ;
            INF = BLEU * TRANSLATION(VZ, -175.0) ;
BOUC1 :     SI FZ < 5.0 ALORS ALLERA BOUC2 ;
            DEPLACER BLEU A MAXS ;
            TO = T ;              c correspond à : 0.2 TEMPO
            PAUSE T > TO + 0.2 ;
            PAUSE FZ < 5.0 : DEBUT
                               ARRETER DEPLACEMENT BLEU ;
                               TO = T ;
                               PAUSE T > TO + 0.5 ;
                               RAZF c remise à zéro du capteur de force
                             FIN
            FINPAUSE ;
BOUC2 :     SI FZ > -5.0 ALORS ALLERA BOUC1 ;
            DEPLACER BLEU A INF ;
            TO = T ;
            PAUSE T > T + 0.2 ;
            PAUSE FZ < -5.0 ALORS ALLERA BOUC1 ;
            TO = T ;
            PAUSE T > T + 0.2 ;
            PAUSE FZ > -5.0 : DEBUT
                                ARRETER DEPLACEMENT BLEU ;
                                TO = T ;
                                PAUSE T > TO + 0.5 ;
                                RALF
                              FIN
            FINPAUSE
            FIN
```

Exemple 2

3.3. Programme de fermeture d'un robinet

L'objectif de ce programme (exemple 3) est de fermer un robinet avec un couple supérieur à une valeur donnée. On connaît la position approximative du robinet et la position de l'axe du robinet, mais on ne connaît pas la position de la poignée autour de cet axe.

Dans une première phase, on déplace la pince de BLEU au-dessus du robinet, puis, après l'avoir ouverte, on cherche à tatons la poignée. Pour cela, on effectue des déplacements de haut en bas. Chaque déplacement est arrêté lorsqu'on mesure une force FZ supérieure à 30N ou lorsqu'on a parcouru une certaine distance (les mors de la pince se situent alors de part et d'autre de la poignée). Entre deux déplacements, la pince tourne de $\pi/10$ autour de son axe.

La deuxième phase consiste à tourner la poignée par 1/2 tours successifs (la pince ne peut pas faire plus d'un tour dans le même sens) jusqu'à l'obtention du couple MZ désiré (80 N/cm).

Le contrôle du moment exercé autour de l'axe Z pendant le serrage est réalisé par le programme parallèle étiqueté par TEST. Lorsque le couple est suffisant, le manipulateur regagne la position initiale.

3.4. Montage d'un carburateur et essais complémentaires

Nous avons réalisé un programme de montage pour deux éléments d'un carburateur. Le programme utilise le capteur de force pour les positionner de façon approximative.

L'ajustement fin se fait en bougeant la partie supérieure sans la tenir. La gravité entraîne naturellement celle-ci vers sa position exacte.

D'autres programmes tels que la manipulation de cubes ou l'insertion d'un axe dans un trou avec une faible tolérance ont été testés. On peut améliorer la fiabilité de ce dernier programme en utilisant des informations provenant des composantes X et Y du capteur de force.

Par ailleurs, des essais ont été effectués pour saisir des objets à partir des informations reçues du système de vision I. Lors de ces essais, la localisation des objets était réalisée avec une erreur de l'ordre de 2 cm (pour une résolution de l'ordre de 4 mm). Nous pensons améliorer cette précision, notamment par un meilleur calibrage.

```
 TYPE CALROB.LMS
        REPERE DEP DROB
        REEL VALMZ VALFZ
        BOOLEEN ARRET DRAP
        VRAI =DRAP
        BLEU REP =DEP
        BLEU REP VX -230. TRANSLAT PRTRS
                VY 500. TRANSLAT PRTRS
                VZ 43. TRANSLAT PRTRS =DROB
        !DROB BLEU DEPL
        BLEU ATTDEP
        40.0 BLEU OUVR BLEU ATTOUV
        FAUX =ARRET
        BLEU REP VZ -80. TRANSLAT PRTRS BLEU DEPL
        BLEU ATTDEP
        BLEU 0.2 CVIT
ITER:   BLEU REP VZ -40. TRANSLAT PRTRS BLEU RAZF  DEPL
        BLEU 0 //DESC
        BLEU ATTDEP
        !DRAP $$SUITE
        VRAI =DRAP
        BLEU REP VZ -VEC1 PI 10. /REEL ROTATION PRTRS
                VZ 40. TRANSLAT PRTRS BLEU DEPL BLEU ATTDEP
        $ITER
SUITE:  !DROB VZ -VEC1 PI 4. /REEL ROTATION PRTRS
                VZ -110. TRANSLAT PRTRS BLEU DEPL
        35. BLEU OUVR BLEU ATTOUV
BOUC:   BLEU ATTDEP BLEU 1. CVIT !ARRET $$TERM
        40. BLEU OUVR BLEU ATTOUV
        BLEU REP VZ 40. TRANSLAT PRTRS
        BLEU DEPL BLEU ATTDEP
        BLEU REP VZ PI ROTATION PRTRS
        BLEU DEPL BLEU ATTDEP
        BLEU REP VZ -40. TRANSLAT PRTRS
        BLEU DEPL BLEU ATTDEP
        35. BLEU OUVR BLEU ATTOUV
        BLEU 0.15 CVIT RAZF
        BLEU REP VZ -VEC1 PI ROTATION PRTRS
        BLEU DEPL
        BLEU 0 //TEST
        $BOUC
TERM:   BLEU REP VZ PI 10. /REEL ROTATION PRTRS BLEU DEPL BLEU ATTDEP
        BLEU REP VZ 100. TRANSLAT PRTRS BLEU DEPL
        BLEU ATTDEP
        !DEP BLEU DEPL
        0.0 BLEU OUVR
        !VALFZ ECRREE !VALMZ ECRREE
        RETOUR
DESC:
        FZ VALARE 30. <REEL $$AGAIN FZ =VALFZ BLEU ARRDEP FAUX =DRAP AGAIN: FIN//
TEST:   MZ VALARE 80. <REEL $$ENCORE MZ =VALMZ BLEU ARRDEP VRAI =ARRET ENCORE: FIN//
        FIN
```

Exemple 3

CONCLUSION

Le projet PANDORE est un projet d'Intelligence Artificielle qui s'intéresse à la commande et à la perception des robots.

Notre objectif était la définition et la réalisation d'un support expérimental pour ce projet.

Le résultat de notre travail est un ensemble matériel opérationnel comprenant deux manipulateurs, un capteur de force, un système de vision et l'interpréteur d'un langage de commande.

Dans la mesure du possible, nous avons cherché à assembler des modules commerciaux pour réaliser le matériel. Les problèmes auxquels nous avons été confrontés se sont donc principalement situés au niveau de l'intégration de ces modules. Cette stratégie nous a permis de disposer d'un matériel d'une bonne fiabilité bien qu'il s'agisse d'un prototype. Celui-ci permet d'expérimenter des programmes d'assemblages et de manipulations complexes utilisant un capteur de force et un système de vision.

Toutefois, le choix des modules a donné lieu à quelques erreurs de jugement comme nous l'avons vu au chapitre 6. Cette première approche "artisanale" dans la construction d'un manipulateur nous a sensibilisés aux problèmes inhérents au matériel de manipulation.

Par ailleurs, un certain nombre d'extensions sont prévues, la plus importante étant l'acquisition du manipulateur SCEMI. Ce manipulateur présente de nombreux avantages sur les manipulateurs actuels du projet PANDORE, notamment pour sa vitesse d'exécution et sa précision.

La technologie actuelle autorise la réalisation de machines mécaniques programmables et adaptatives à partir d'éléments commerciaux. Cette qualité ne se retrouve pas au niveau de la commande. Pour cette raison, nous avons défini un langage de programmation, LM, qui exprime dans un formalisme simple les opérations que les manipulateurs doivent exécuter. Ce langage permet de décrire une tâche en terme des mouvements, des actions, de l'outil terminal des manipulateurs, et des informations provenant des capteurs. Des extensions sont prévues à ce langage telles que les mouvements à compliance et trajectoires incomplètement spécifiées.

Le logiciel actuellement implanté est l'interpréteur d'un langage LMA syntaxiquement plus simple que LM mais permettant d'exprimer sa sémantique. Une implantation complète de LM est en cours. D'autres implantations sont prévues à titre industriel (SCEMI) ou universitaire (Projet ARA, Besançon).

Les expérimentations faites à l'aide du matériel mis en oeuvre et du langage LMA montrent clairement que si ce niveau de programmation est indispensable pour la commande des manipulateurs, la programmation des tâches d'assemblage reste complexe à définir et à mettre au point.

Ces difficultés justifient certaines des recherches plus fondamentales entreprises dans le cadre du projet PANDORE telles que la synthèse automatique de mouvements fins en assemblage ou la préhension automatique des objets.

ANNEXE A

EXEMPLES DE PROGRAMMES EN LM

Dans cette annexe, nous donnons trois exemples simples de programmes écrits dans le langage LM. Ces exemples sont essentiellement destinés à illustrer quelques possibilités offertes par ce langage. On pourrait imaginer de nombreuses variantes à chacun d'eux, capables de s'adapter à différents types d'incidents.

Nous avons numéroté chaque ligne des programmes, pour faciliter l'exposé des commentaires. Les unités de mesure choisies dans ces commentaires correspondent à celles imposées par une implantation fictive du langage car la définition de celui-ci ne spécifie pas les unités à utiliser.

Exemple 1 :

L'exemple 1 de programme permet de déplacer un cube d'une position POS1 à une position POS2 (cf. Figure 1).

Les lignes 01 et 02 déclarent les variables utilisées dans le programme. Nous supposons que l'outil du manipulateur commandé est une pince à mors parallèles ; le repère PINCE désigne un repère attaché à cette pince comme le montre la figure 2. Les repères CUBE et PRISE.CUBE sont attachés au cube à manipuler comme le montre la figure 3. Les repères POS1 et POS2 désignent les positions initiale et finale du cube. Le repère REPOS désigne la position de repos du manipulateur.

La ligne 04 définit la position du repère PINCE relativement au repère d'état ROBOT1 et la ligne 05 lie ces deux repères. La translation choisie dans l'instruction de la ligne 04 est essentiellement illustrative car, dans cette annexe, nous n'avons défini ni le repère ROBOT1 relativement à l'extrémité utile du manipulateur, ni le mode de fixation de la pince sur cette extrémité, ni les caractéristiques géométriques de la pince.

Les lignes 06 à 10 définissent la position de repos du manipulateur et les positions initiale et finale du cube à déplacer. La ligne 11 définit la position du repère PRISE.CUBE relativement au repère CUBE et la ligne 13 lie ces deux repères.

```
01  TRANSFORMATION ACCES ;
02  REPERE PINCE, CUBE, PRISE.CUBE, POS1, POS2, REPOS ;
03  DEBUT
04     PINCE := ROBOT1 * TRANSLATION(-VZ,90) ;
05     LIER PINCE ROBOT1 ;
06     REPOS := STATION * TRANSLATION(VZ,300) ;
07     POS1 := STATION * TRANSLATION(VY,100) * TRANSLATION(VX,100) ;
08     POS2 := STATION * TRANSLATION(VY,400) * TRANSLATION(VX,500)
09                                          * ROTATION(VZ,3.14159/4) ;
10     CUBE := POS1 ;
11     PRISE.CUBE := CUBE * TRANSLATION(VY,10) * TRANSLATION(VX,10)
12                                            * TRANSLATION(VZ,50) ;
13     LIER PRISE.CUBE CUBE ;
14     ACCES := TRANSLATION(VZ,150) ;
15     ACTIONNER PINCE ROBOT1 30 / ;
16     DEPLACER PINCE A PRISE.CUBE VIA PRISE.CUBE * ACCES / ;
17  *  ACTIONNER PINCE ROBOT1 0 JUSQUA PRESSION1 > 150 / ;
18     SI ECART1 > 15
19        ALORS
20             DEBUT
21                LIER PRISE.CUBE PINCE ;
22                DEPLACER CUBE A POS2 VIA POS1 * ACCES, POS2 * ACCES / ;
23                ACTIONNER PINCE ROBOT1 30 / ;
24                DELIER PINCE PRISE.CUBE
25             FIN
26        SINON ECRIRE 'LE CUBE N'EST PAS ICI' ;
27     DEPLACER ROBOT1 A REPOS VIA ROBOT1 * ACCES ;
28     RETOUR
29  FIN
```

Exemple 1

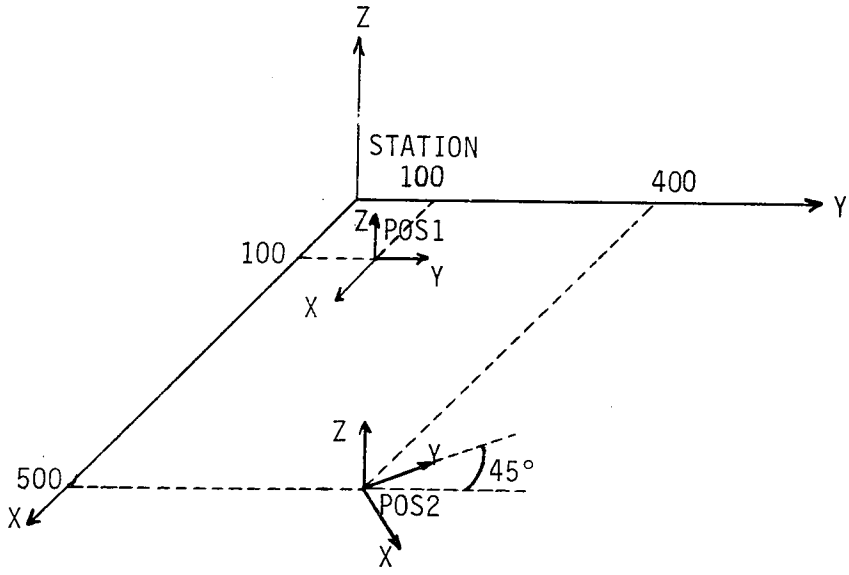


Figure 1

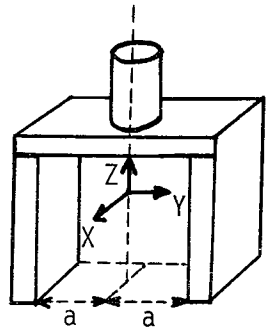


Figure 2

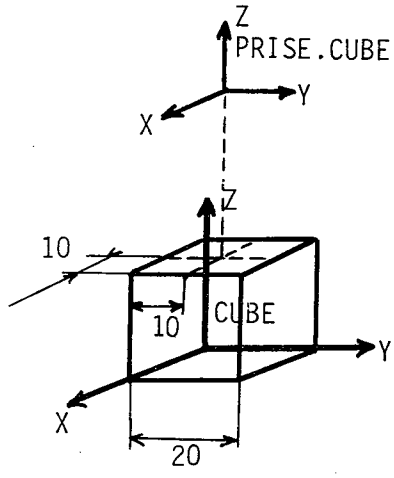


Figure 3

La ligne 15 commande une ouverture de la pince de 30 mm (le cube a 20 mm de côté). La ligne 16 commande un déplacement du manipulateur de telle sorte que le repère PINCE vienne en PRISE.CUBE. Le passage par la position intermédiaire PRISE.CUBE * ACCES impose à la pince d'approcher le cube verticalement.

Le ligne 17 commande la fermeture complète de la pince. La condition d'arrêt utilise la variable d'état PRESSION. Nous supposons que cette variable mesure la pression moyenne exercée par les deux mors de la pince. Elle commande d'arrêter la fermeture de la pince lorsque cette pression dépasse 150 g/cm^2.

La ligne 18 teste l'écart entre les mors de la pince lorsque l'action précédente est terminée. On considère qu'un écart inférieur à 15 mm signifie qu'on n'a pas saisi le cube. Cela entraîne l'impression d'un message sur le terminal (ligne 26).

Si l'écart entre les pinces est supérieur à 15 mm, on considère que le cube a été saisi et on lie le repère PRISE.CUBE à PINCE (ligne 21). On note que le repère CUBE est alors lié indirectement à ROBOT1. La ligne 22 commande son déplacement en POS2, en passant par deux positions intermédiaires situées au dessus de POS1 et de POS2 pour éviter des collisions avec d'autres objets éventuels posés sur la table de travail. La ligne 23 commande l'ouverture des pinces, ce qui conduit à délier PRISE.CUBE de PINCE (ligne 24).

A la ligne 27, on déplace le manipulateur à sa position de repos avant de terminer l'exécution du programme.

Exemple 2 :

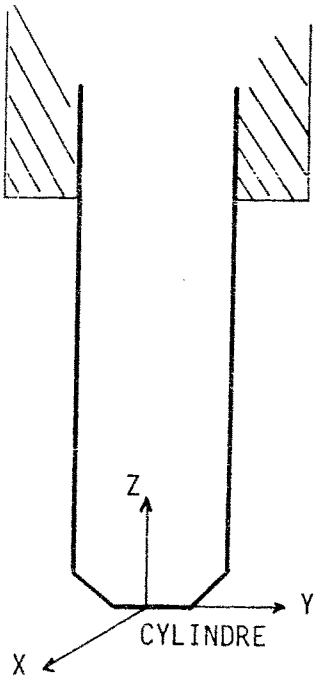
L'exemple 2 est une procédure pour insérer un cylindre déterminé par le repère CYLINDRE dans un trou localisé par le repère TROU. Les repères CYLINDRE et TROU font partie des paramètres de la procédure ; CYLINDRE doit être un repère déplaçable. Celle-ci retourne la valeur VRAI si l'insertion réussit et la valeur FAUX dans le cas contraire.

```
01  PROCEDURE INSERER (CYLINDRE REPERE, TROU REPERE, FZMAX REEL,
02                                  EPS REEL, NMAX ENTIER) BOOLEEN ;
03  ENTIER N ; VECTEUR V ; REPERE TROU1 ;
04  DEBUT
05      N := 0 ; TROU1 := TROU * TRANSLATION(-VZ,10) ;
06      TANTQUE N < NMAX FAIRE
07          DEBUT
08              DEPLACER CYLINDRE DE TRANSF (CYLINDRE,TROU1)
09                                      JUSQUA FZ1 > FZMAX ;
10              SI DISTANCE (CYLINDRE, TROU) < 1
11                  ALORS RETOUR (VRAI)
12                  SINON
13                      DEBUT
14                          V := VECTEUR(FX1,FY1,0) ;
15                          DEPLACER CYLINDRE DE TRANSLATION(V,EPS) ;
16                          EPS := EPS/2 ; N := N+1
17                      FIN
18          FIN ;
19      ECRIRE 'L'INSERTION A ECHOUE' ;
20      RETOUR (FAUX)
21  FIN
```

Exemple 2

```
01  PROCEDURE VISSER (MZMAX REEL, FZMIN REEL, EPS REEL) ;
02  REPERE GLOBAL TOURNEVIS ;
03  DEBUT
04      ACTIONNER TOURNEVIS ROBOT1 JUSQUA MZ1 > MZMAX ;
05      BOUCLE : PAUSE
06                  FZ1 < FZMIN : DEPLACER TOURNEVIS
07                                      DE TRANSLATION (-VZ1,EPS) ;
08                  MZ1 > MZMAX : RETOUR
09              FINPAUSE ;
10      ALLERA BOUCLE
11  FIN
```

Exemple 3



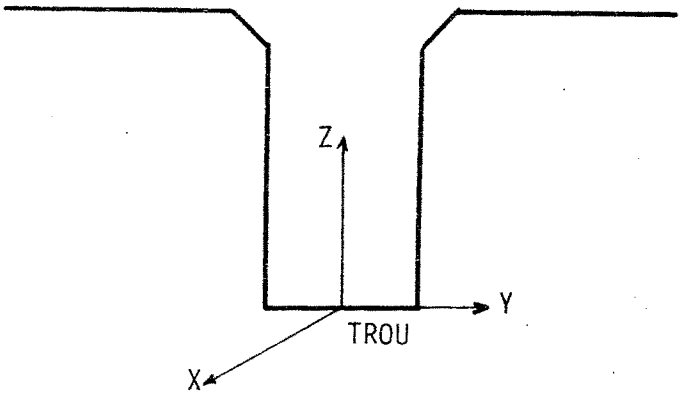


Figure 4

Au moment de l'appel de la procédure, nous supposons que la situation est celle de la figure 4 : le cylindre est tenu verticalement au dessus du trou. De plus, nous supposons que l'erreur de positionnement de l'axe du cylindre relativement à celui du trou est inférieure à la largeur des chanfreins.

La ligne 08 commande le déplacement du cylindre dans le trou suivant un chemin rectiligne plus long de 10 mm que la distance de CYLINDRE à TROU. La condition d'arrêt interrompt le déplacement lorsque la composante verticale de la force (FZ1) exercée par la pince dépasse le seuil FZMAX.

La ligne 10 compare la distance de CYLINDRE à TROU à l'arrêt. Si cette distance est inférieure au mm, on considère que l'insertion a réussi. Dans le cas contraire, on procède à une correction. Celle-ci consiste à déplacer légèrement (translation horizontale de longueur EPS) le cylindre de façon à réduire la composante horizontale de la force exercée par la pince (ligne 15). Une fois cette correction effectuée, la procédure tente à nouveau l'insertion.

Après chaque correction, EPS est divisé par 2 (ligne 16) pour favoriser la convergence de l'itération. Le nombre d'itérations est limité à NMAX (ligne 06).

Exemple 3 :

L'exemple 3 est une procédure qui illustre la coordination d'une action de l'outil (ici un tournevis électrique d'un manipulateur) et de déplacements de ce manipulateur. Il s'agit de réaliser une opération de vissage en faisant en sorte que la force exercée par le tournevis sur la vis reste supérieure à un seuil.

La ligne 04 commande d'actionner le tournevis jusqu'à l'obtention d'un couple MZMAX. La ligne 05 met l'exécution de la procédure en attente (mais l'action du tournevis se poursuit) jusqu'à ce que le couple MZ1 dépasse la valeur MZMAX - auquel cas on considère que le vissage est terminé -, ou que la composante de force FZ1 devienne inférieure à FZMIN. Dans le second cas, on commande un déplacement du manipulateur destiné à augmenter FZ1 (lignes 06 et 07).

ANNEXE B

DESCRIPTION DES SYSTEMES Z80

Deux systèmes Z80 identiques servent d'interfaces entre le calculateur Micro 1/03 et les actionneurs de BLEU et NOIR (cf. chapitre 4). Dans cette annexe, nous donnons une description plus détaillée de chacun de ces systèmes.

1. CONFIGURATION MATERIELLE

Les schémas 1, 2, 3, 4 représentent respectivement :

(1) la partie processeur d'un système Z80 avec l'interface d'entrées-sorties pour le calculateur Micro 1/03 et les circuits d'horloges,
(2) la partie mémoire,
(3) la partie entrées-sorties,
(4) la partie sécurité.

1.1. La partie processeur

a) *Le processeur*

Il s'agit d'un microprocesseur 8 bits Z80. Celui-ci nous a paru mieux adapté à la gestion des interruptions que les autres microprocesseurs de la même classe. De plus, un système TEKTRONIX 8002 de développement et d'émulation pour ce type de microprocesseur existe à l'atelier de Micro-informatique.

b) *Interface d'entrées-sorties calculateur*

Les communications entre le système Z80 et le Micro 1/03 se font sur une ligne parallèle de 8 bits à la vitesse de 30 K bauds. Plus précisément, le Micro 1/03 est relié aux systèmes Z80 par un interface de 16 bits parallèle DRV 11 unique et partagé par les deux systèmes. La figure 5 représente l'interface entre le DRV 11 et les deux PIO des systèmes Z80.

c) *Circuits d'horloges*

Les trains d'impulsions sont envoyés aux actionneurs par programme. Nous utilisons 2 CTC (Control Timer Circuit) pour produire les interruptions nécessaires à l'envoi des pas au moteur. Chacun de ces circuits possède 4 décompteurs programmables qui interrompent le microprocesseur lors du

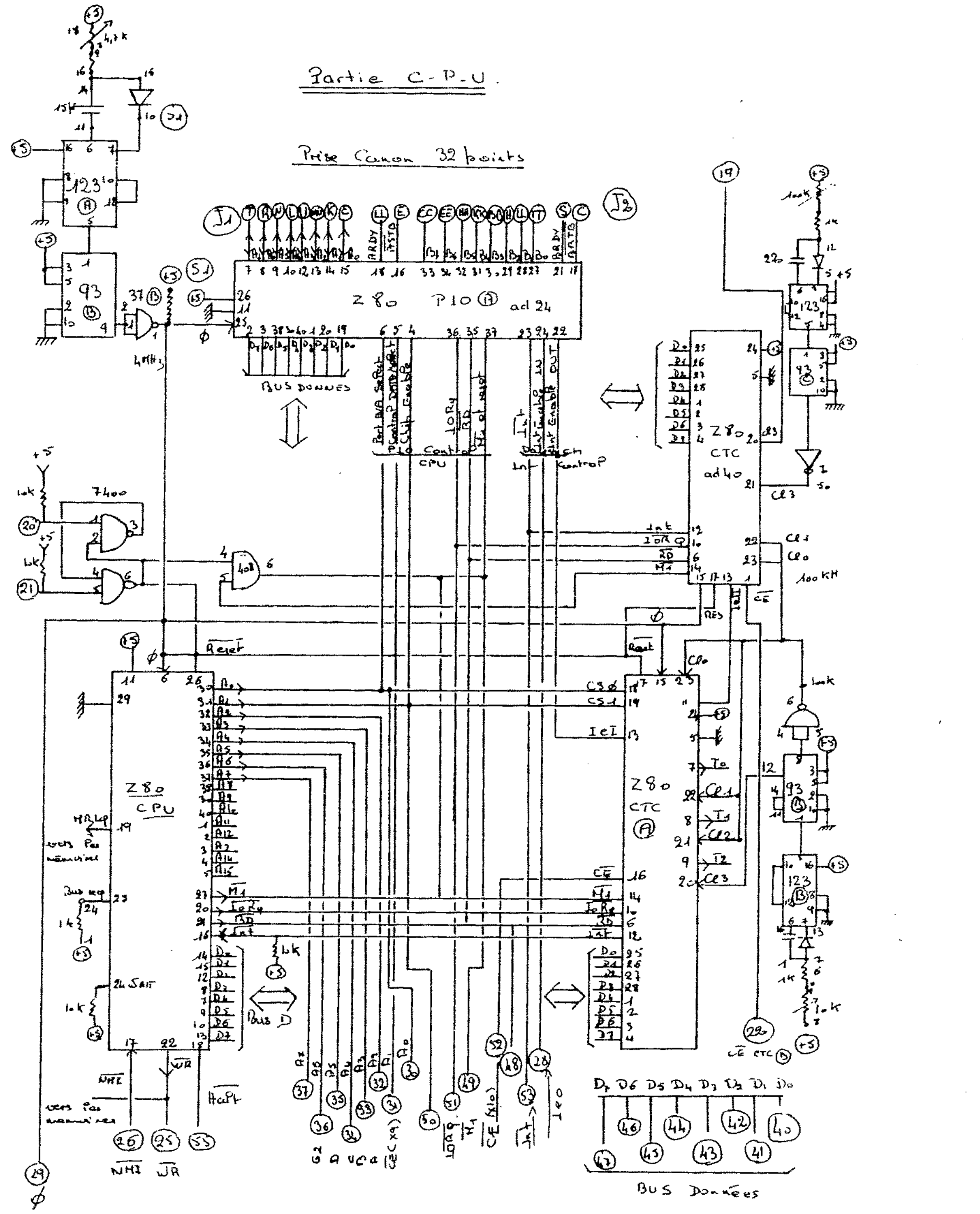

Partie C-P-U.
Prise Canon 32 points
Z80 P10 ad 24
BUS DONNES
Z80 CTC ad 40
Z80 CPU
Z80 CTC
BUS Données

Figure 1

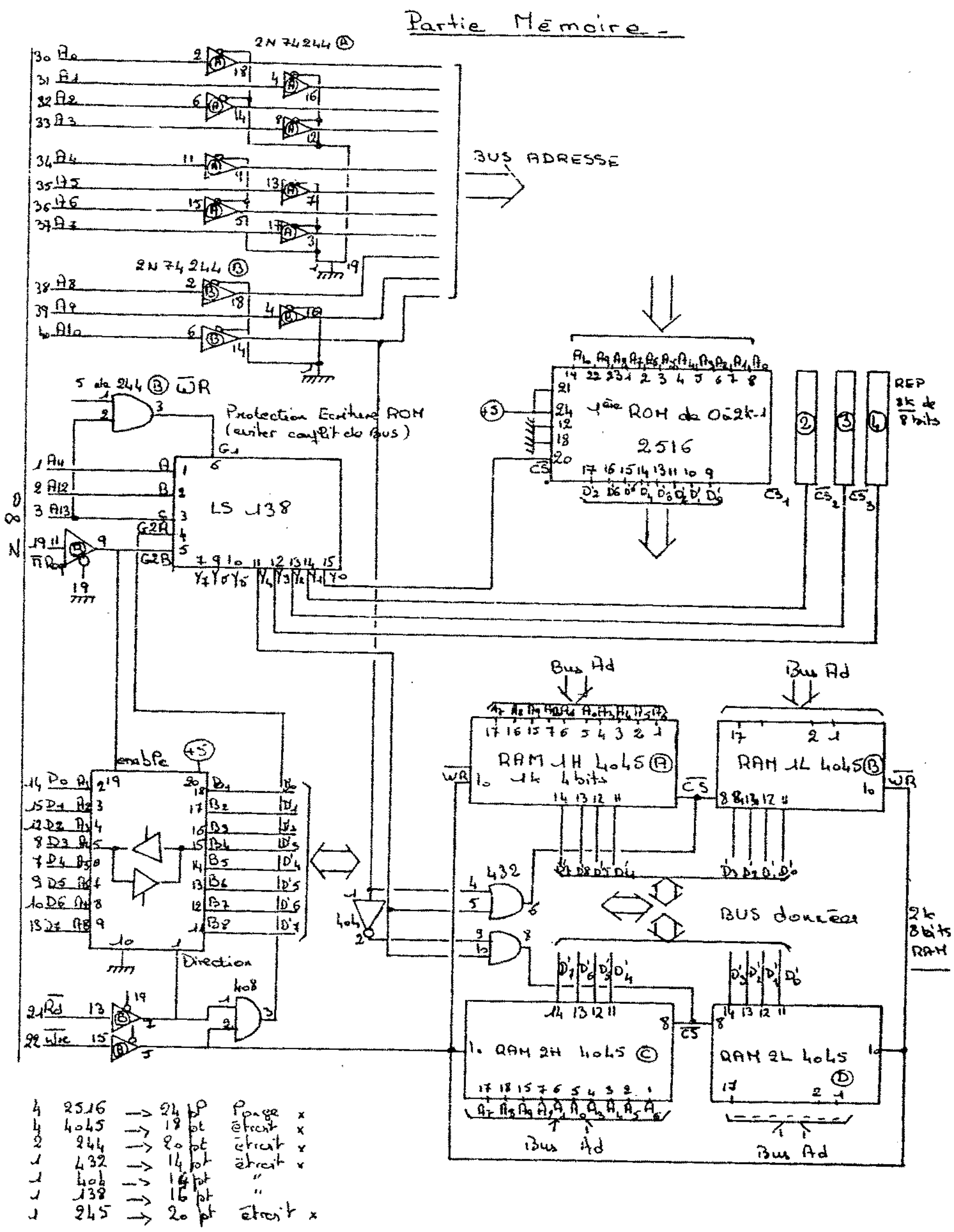

Partie Mémoire -
2N 74244 Ⓐ
2N 74244 Ⓑ
BUS ADRESSE
Protection Ecriture ROM (eviter conflit de bus)
LS 138
ROM de 0à2k-1
2516
REP 8k de 8 bits
RAM 1H 4045 Ⓐ
1k 4 bits
RAM 1L 4045 Ⓑ
RAM 2H 4045 Ⓒ
RAM 2L 4045 Ⓓ
Bus Ad
BUS données
2k 8 bits RAM
enable
Direction
4 2516 -> 24 pt
4 4045 -> 18 pt
2 244 -> 20 pt
1 432 -> 14 pt
1 408 -> 14 pt
1 138 -> 16 pt
1 245 -> 20 pt

Figure 2

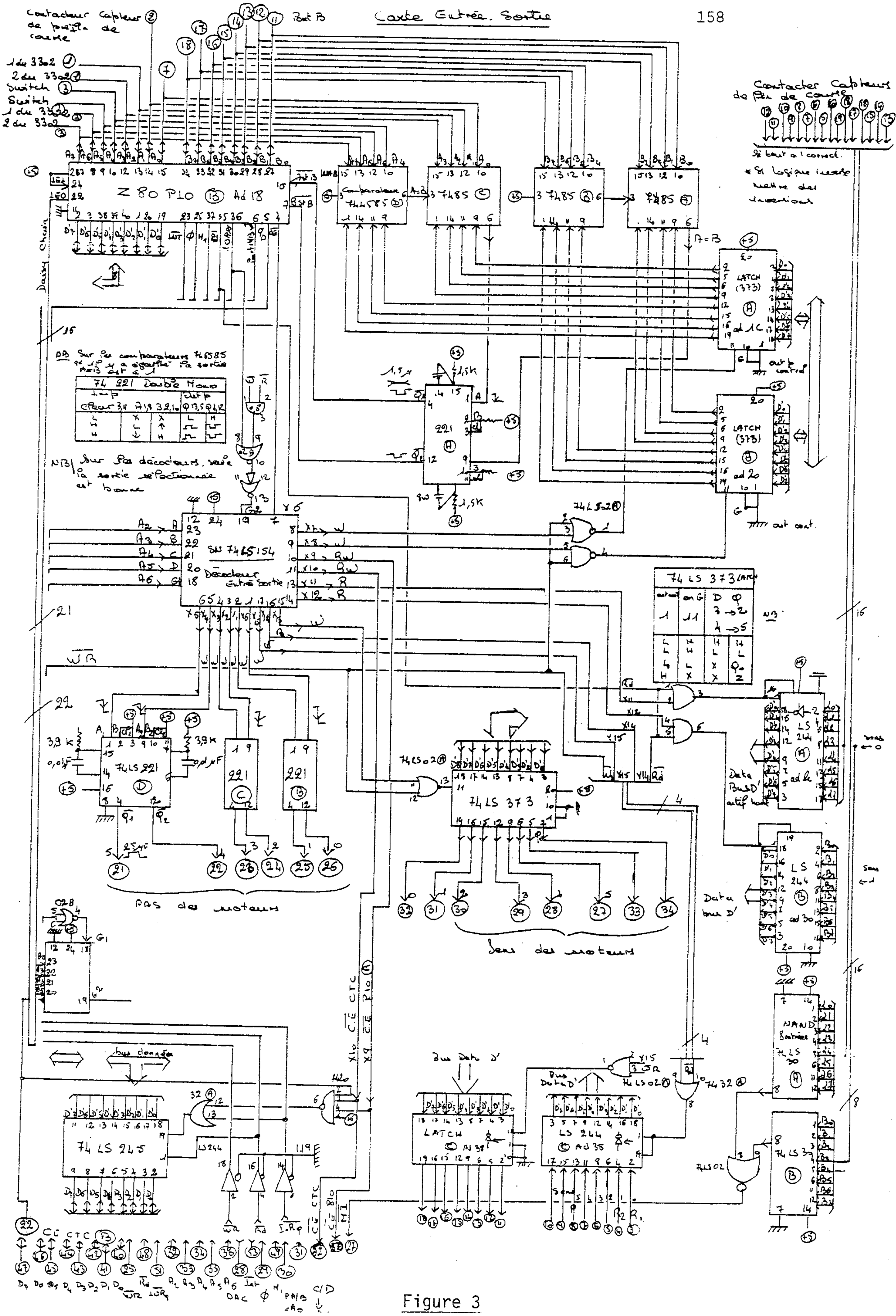

Carte Entrée Sortie
Z 80 PIO
74LS85
7485
LATCH (373)
74 221 Double Mono
SN 74LS154
Décodeur Entrée Sortie
74 LS 373
74LS221
221
74 LS 244
PAS des moteurs
Sens des moteurs
NAND 8 entrées 74 LS 30
74 LS 245
bus données
Bus Data D'
LATCH
LS 244
74 LS 30

Figure 3

CABLAGE CAPTEUR FIN DE COURSE

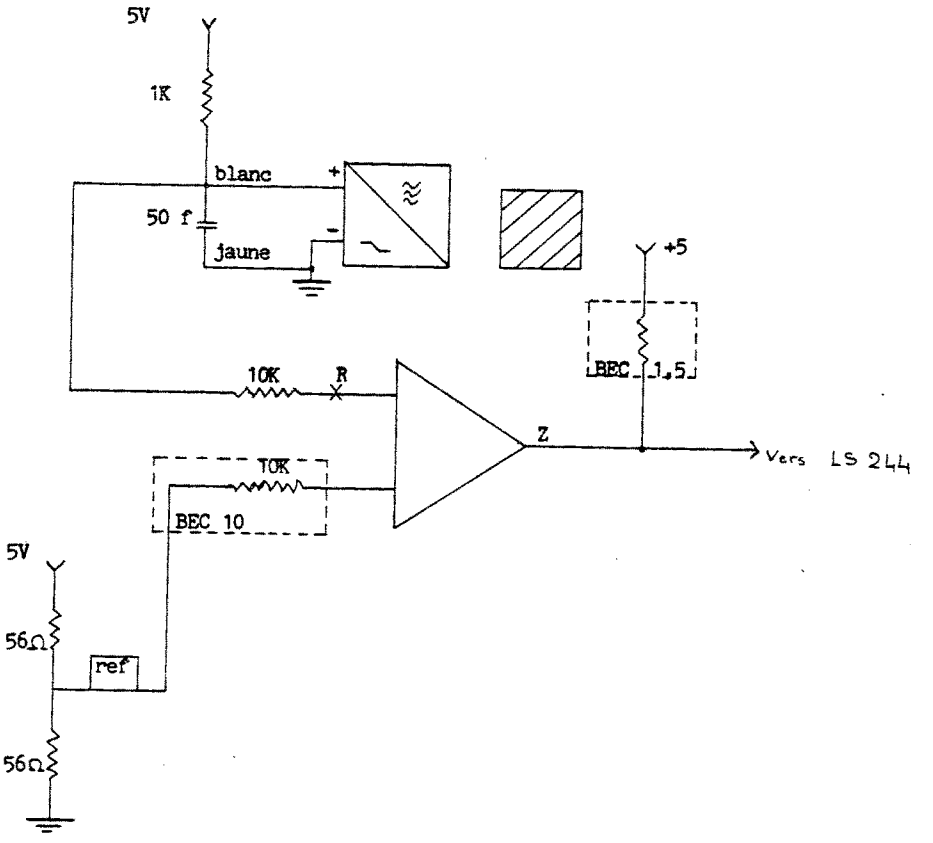


Figure 4

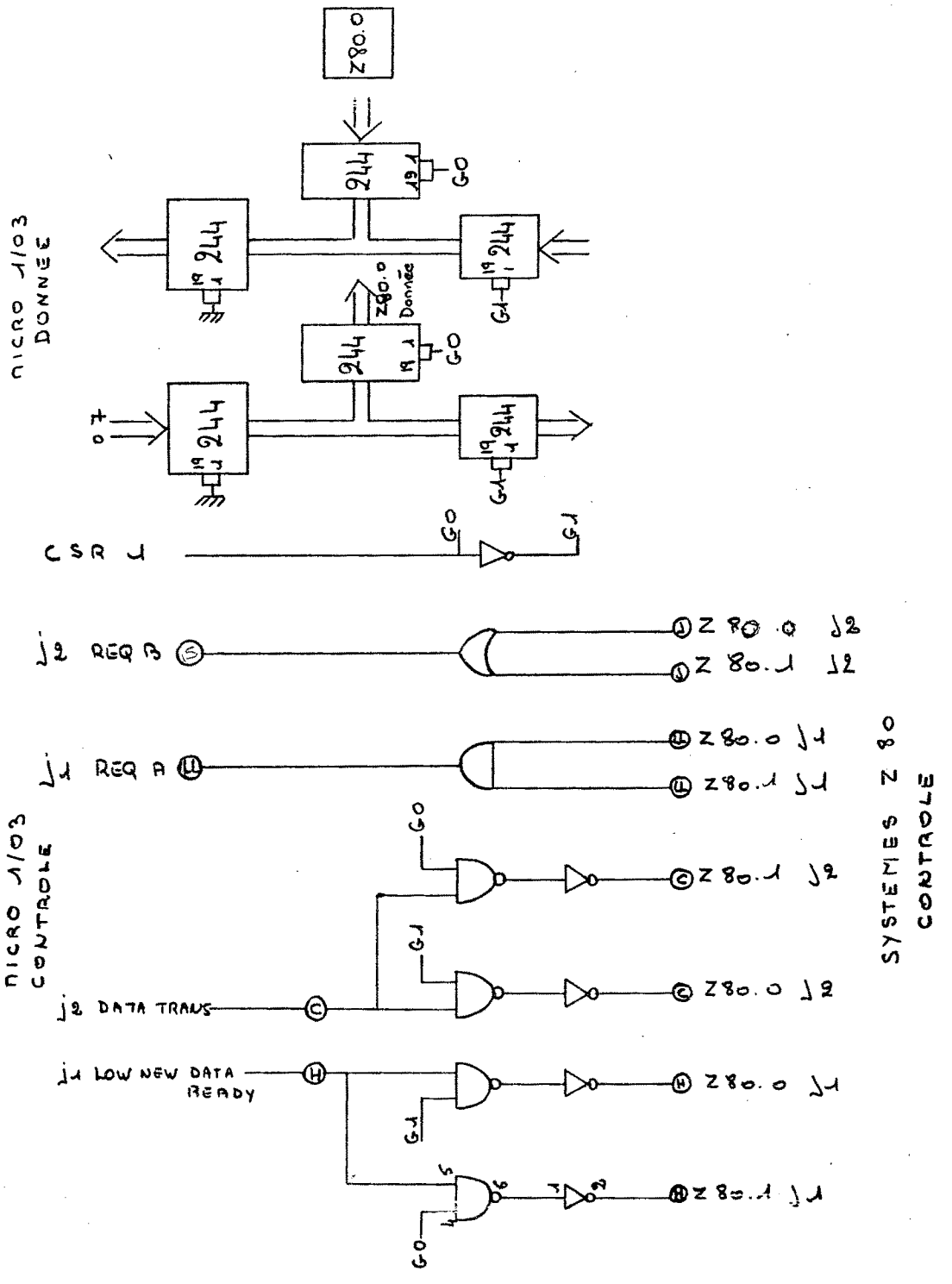

Z80.0
244
G0
G1
Z80.0 Donnée
MICRO 1/03 DONNEE
CSR 1
J2 REQ B
Z 80.0 J2
Z 80.1 J2
J1 REQ A
Z80.0 J1
Z80.1 J1
MICRO 1/03 CONTROLE
J2 DATA TRANS
J1 LOW NEW DATA READY
SYSTEMES Z 80 CONTROLE

Figure 5

passage par zéro d'un décompteur. On utilise chaque compteur comme un diviseur de fréquence programmable en connectant ceux-ci à une horloge de référence. On peut ainsi choisir par programme la durée de l'intervalle séparant deux interruptions. Jusqu'à six décompteurs sont utilisés pour les actionneurs. Les deux autres décompteurs servent respectivement à l'activation d'une tâche système et à la scrutation du boîtier de commande manuelle.

1.2. La partie mémoire

Le système possède 8K mots de mémoire morte reprogrammable et 2K mots de mémoire vive. La capacité mémoire peut être doublée en rajoutant les circuits nécessaires.

1.3. La partie entrées-sorties

a) *Commandes des actionneurs*

Le sens de rotation est commandé à l'aide d'un port de sortie de 8 bits (1 bit par actionneur).

Par ailleurs, pour envoyer les impulsions, nous avons identifié chaque actionneur à un port d'entrée-sortie. L'envoi d'une impulsion à un actionneur est réalisé par le décodage sur le bus de l'adresse du port d'entrée-sortie. L'instruction "OUT (MOTO), A" provoque ainsi l'envoi d'une impulsion. La donnée contenue dans A apparaît sur le bus de données, mais n'est pas utilisée. Seule l'adresse "MOTO" est décodée pendant le temps de l'instruction. Un monostable transforme cette impulsion en une impulsion plus longue transmise à l'actionneur adressé. Cette technique permet de réaliser un gain de temps appréciable dans les programmes d'interruption gérant l'envoi des pas, aucun registre n'ayant besoin d'être modifié.

b) *Port de contrôle*

Un port de sortie de 8 bits donne accès aux commandes suivantes :

- la mise à 0 du bit 7 provoque l'ouverture du contacteur d'alimentation des moteurs,
- la mise à 1 du bit 6 inhibe l'action des capteurs de fin de course et déclanche l'alarme sonore,
- le bit 0 commande le témoin de contrôle manuel des actionneurs.

Il est prévu d'utiliser les bits 1 à 5 pour commander l'ouverture d'une pince pneumatique et pour envoyer des signaux de synchronisation à d'autres systèmes.

Un port d'entrée de 8 bits est disponible. Il peut être utilisé pour connaître l'état de capteurs tout ou rien.

c) *Interface boîtier de commande manuelle*

Un port d'entrée de 8 bits permet de connaître l'état des boutons du boîtier de commande manuelle. Un potentiomètre commande la fréquence de référence du décompteur associé à la tâche de scrutation du boîtier.

d) *Capteurs de fin de course*

Chaque capteur de fin de course est relié à un port d'entrée où l'on peut lire son état à tout moment.

2. TRAITEMENT DES CAPTEURS DE FIN DE COURSE

2.1. Sécurité

La sécurité du manipulateur peut être mise en cause :
- soit par une commande impossible dont l'exécution entraînerait un incident,
- soit par une panne indépendante de l'utilisateur (pertes anormales de pas, erreur d'un actionneur).

Sous réserve que le logiciel de plus haut niveau réalise les vérifications nécessaires d'une manière parfaite, tout incident du premier type est impossible. Pour les autres pannes, un changement d'état sur les capteurs de fin de course provoque l'ouverture du contacteur d'alimentation des moteurs.

2.2. Initialisation du manipulateur

Les capteurs de fin de course sont aussi utilisés pour initialiser le manipulateur sur une position de référence. Dans cette phase, ils ne doivent pas faire disjoncter le contacteur.

D'autre part, après un incident, il est nécessaire de pouvoir dégager le manipulateur de la position qui a provoqué l'arrêt du fonctionnement.

Une commande manuelle et une commande programmable permettent de masquer les sécurités. Une alarme sonore est active lorsque les sécurités sont masquées.

3. COMMANDES DU SYSTEME

Chaque commande est envoyée par le Micro 1/03 au système Z80 sur une chaîne d'octets de la forme :

nombre d'octets de la commande	numéro de la commande	[paramètre]*
1 octet	1 octet	1 octet/ paramètre

Le compte-rendu du système Z80 a une structure analogue :

nombre d'octets du compte-rendu	[paramètre]*
1 octet	1 octet/ paramètre

3.1. Interface Logiciel

a) *Entrées-sorties*

Les entrées-sorties sont faites depuis le Micro 1/03 par un sous-programme écrit en assembleur (IOZ80). Chacun des octets de la commande doit se trouver dans un mot d'un tableau d'entiers IBUFF. On a :

- avant l'appel du programme IOZ80 :

  IBUFF(1) = nombre d'octets de la commande

  IBUFF(2) = numéro de la commande

  IBUFF(3) = paramètre

  ...

- après l'appel du programme d'entrées-sorties :

  IBUFF(1) = 1er paramètre du compte-rendu

  IBUFF(2) = 2ème paramètre du compte-rendu

  ...

b) *Primitives de commande*

A chaque commande du système Z80, excepté les commandes moniteur (cf. § 3.2), est associé un sous-programme FORTRAN paramétré.

Le rôle de ce sous-programme est de :
- déterminer les deux premiers octets de la commande,
- faire les changements d'unités nécessaires, tels que convertir une distance exprimée en mm en un nombre de pas pour un moteur donné,
- mettre les paramètres dans IBUFF sous le format correct,
- appeler le programme IOZ80 et convertir les paramètres du compte-rendu en format FORTRAN.

Ces sous-programmes FORTRAN sont utilisés par les primitives actionneur-capteur de l'interpréteur du langage LM-A (cf. § 6, chapitre 5).

3.2. Commandes "moniteur"

Un moniteur minimal destiné à la mise au point a été implanté sur le système Z80.

a) *Examen de la mémoire*

- Commande :

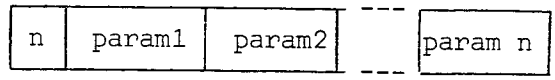
4	1	param1	param2	param3

param1 : poids fort de l'adresse du début d'examen

param2 : poids faible de l'adresse du début d'examen

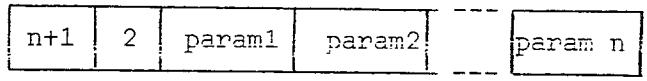
param3 : nombre d'octets à lister (n).

- Compte-rendu :



b) *Modification de la mémoire*

- Commande :

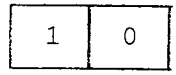


param1 : poids fort de l'adresse du début de la modification

param2 : poids faible de l'adresse du début de la modification

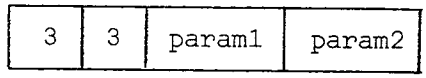
param3, param n : nouvelles valeurs à mettre en mémoire à partir de l'adresse précisée.

- Compte-rendu :



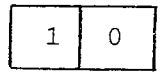
c) *Lancement d'un programme*

- Commande :



param1, param2 : param1 et param2 sont respectivement les poids fort et faible de l'adresse de lancement.

- Compte-rendu :



3.3. Commandes des déplacements et des actions de la pince

A chaque actionneur de BLEU et de NOIR est attaché un numéro.

- Pour BLEU, ce sont :

  0 (translation verticale T1), 1 (rotation R1), 2 (translation horizontale T2), 3 (rotation R2), 5 (pince),

- Pour NOIR, ce sont :

  0 (rotation R'1), 1 (rotation R'2) et 5 (pince).

a) Description des déplacements

La commande suivante permet de décrire au système Z80 le prochain mouvement de l'actionneur spécifié. La description d'un déplacement concernant n degrés de liberté est faite par n commandes, chacune adressée à l'actionneur correspondant. Une autre commande permet d'exécuter le déplacement (cf. § b).

- Commande :

5	6	param1	param2	param3	param4

param1 : numéro de l'actionneur (0, 1, 2 ou 3 pour BLEU ; 0 ou 1 pour NOIR)
param2 : (bit 7) signe du déplacement
(bits 6 à 0) poids fort du déplacement en nombre de pas
param3 : poids faible du nombre de pas
param4 : diviseur de la fréquence de référence.

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

CALL DEP (N, D, V)

N (entier) : numéro de l'actionneur
D (réel) : déplacement en mm ou radian
V (réel) : vitesse en mm/s ou radian/s

b) Exécution du déplacement

Cette commande lance l'exécution d'un déplacement conformément à la description qui en a été faite.

- Commande :

1	10

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

CALL LANDEP.

c) *Description de l'action*

La commande suivante permet de décrire au système Z80 la prochaine action (ouverture-fermeture) de la pince. Une autre commande permet d'exécuter cette action (cf. § d).

- Commande :

5	6	5	param1	param2	param3

param1 : (bit 7) signe du déplacement
(bits 6 à 0) poids fort du déplacement en nombre de pas
param2 : poids faible du nombre de pas
param3 : diviseur de la fréquence de référence.

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

CALL ACT (D, V)

D (réel) : déplacement en mm ou radian
V (réel) : vitesse de déplacement.

d) *Exécution de l'action*

Cette commande lance l'exécution de l'action de la pince conformément à la description qui en a été faite.

- Commande :

1	11

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

CALL LANACT.

3.4. Commandes d'arrêt

a) *Arrêt d'un déplacement*

Cette commande arrête un déplacement en cours. Le descripteur (nombre de pas, vitesse) du mouvement de chaque actionneur concerné est conservé. Le nombre de pas est celui restant à faire.

- Commande : | 1 | 12 |

- Compte-rendu : | 1 | 0 |

- Primitive de commande FORTRAN :

```
CALL ARRDEP.
```

b) *Arrêt de l'action de la pince*

Cette commande arrête l'action de la pince. Le descripteur du mouvement de l'actionneur 5 est conservé. Le nombre de pas est celui restant à faire.

- Commande : | 1 | 13 |

- Compte-rendu : | 1 | 0 |

- Primitive de commande FORTRAN :

```
CALL ARRACT.
```

c) *Remise à zéro du nombre de pas dans le descripteur des actionneurs de déplacement*

Cette commande indique que le déplacement se termine à la position courante.

- Commande : | 1 | 14 |

- Compte-rendu : | 1 | 0 |

- Primitive de commande FORTRAN :

  CALL RAZDEP.

d) Remise à zéro du nombre de pas dans le descripteur de l'actionneur de la pince

Cette commande indique que l'action de la pince se termine avec l'écart courant.

- Commande :

1	15

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

  CALL RAZACT.

3.5. Modification des descripteurs de mouvements

L'utilisateur peut au cours d'un déplacement ou d'une action modifier les descripteurs des mouvements des actionneurs correspondants.

Les commandes correspondantes doivent obligatoirement être précédées par une commande d'arrêt (ARRDEP ou ARRACT) et être suivies par une commande d'exécution (LANDEP ou LANACT).

a) Modification de la vitesse seule

- Commande :

2	7	param1	param2

  param1 : numéro de l'actionneur

  param2 : nouveau diviseur de la fréquence de référence

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

  CALL VI (N,V)

  N (entier) : numéro de l'actionneur

  V (réel) : nouvelle vitesse en mm/s ou radian/s.

b) *Modification des descripteurs de mouvement ou d'action*

On emploie les commandes des § 3.3 a et c.

3.6. Position

La commande suivante permet de connaître la position d'un degré de liberté d'un système mécanique ou de sa pince.

- Commande :

2	5	param1

param1 : numéro de l'actionneur.

- Compte-rendu :

3	param1	param2

param1 : poids fort du nombre de pas restant à faire
param2 : poids faible du nombre de pas restant à faire.

- Primitive de commande FORTRAN :

CALL POSI (N,D)

N (entier) : numéro de l'actionneur
D (réel) : distance restant à parcourir (en mm ou radian).

[Remarque : Pour connaître au même instant la position de plusieurs degrés de liberté d'un système mécanique et l'écart de sa pince, il faut faire précéder les commandes POSI par des commandes d'arrêt (ARRDEP, ARRACT) et les faire suivre par des commandes d'exécution (LANDEP, LANACT)].

3.7. Commandes diverses

a) *Passage en commande manuelle*

- Commande :

1	16

- Compte-rendu :

12	param1	param2	----	param11	param12

param 2n-1 : poids fort du nombre de pas effectués par l'actionneur n-1
param 2n : poids faible du nombre de pas effectués par l'actionneur n-1

- Primitive de commande FORTRAN :

  CALL MANUEL (D0, D1, ..., D5)

  D0, D1, ..., D5 (réels) : distances parcourues par chaque actionneur pendant la commande manuelle

[Remarque : Pour BLEU, comme pour NOIR, seules certaines de ces distances sont significatives].

b) *Initialisation d'un degré de liberté ou d'une pince*

- Commande :

11	17	param1	0	0	0	5	4	param1	param2	param3	param4

  param1 : numéro de l'actionneur correspondant

  param2 : bit 7 obligatoirement à 1
  bits 6 à 0 poids fort de la position initiale repérée par rapport à la butée

  param3 : poids faible de la position initiale

  param4 : diviseur de la fréquence de référence

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

  CALL INIT (N, 0, V)

  N (entier) : numéro de l'actionneur à initialiser

  D (réel) : distance en mm ou radian où il faut positionner l'actionneur par rapport à la butée

  V (réel) : vitesse de retour au point spécifié en mm/s ou radian/s.

c) *Ecriture d'un mot sur le port de sortie*

- Commande :

2	21	param1

  param1 : valeur du mot à écrire

- Compte-rendu :

1	0

- Primitive de commande FORTRAN :

  CALL CONTROL (I)

  I (entier) : valeur du mot à écrire

d) Lecture du mot présent sur le port d'entrée

- Commande : | 1 | 22 |

- Compte-rendu : | 1 | param1 |

  param1 : valeur du mot présent sur le port d'entrée

- Primitive de commande FORTRAN :

  CALL SIGNAL (I)

  I (entier) : valeur du mot présent sur l'entrée

e) Simulation du temps de freinage des actionneurs

Un programme système est activé sur une interruption provoquée par une horloge (cf. § 1.1.c). Le temps séparant deux exécutions de ce programme simule la durée du freinage des actionneurs. On peut le faire varier (pour l'adapter aux vitesses choisies) en utilisant la commande suivante :

- Commande : | 2 | 20 | param1 |

  param1 : nouveau diviseur de la fréquence de référence

- Compte-rendu : | 1 | 0 |

- Primitive de commande FORTRAN :

  CALL RETARD (T)

  T (réel) : temps simulant la durée du freinage des actionneurs.

f) Etat du robot

Cette commande permet à l'utilisateur de savoir si un déplacement est en cours d'exécution.

- Commande : | 1 | 18 |

- Compte-rendu : | 1 | param1 |

  param1 : nombre d'actionneurs de déplacement en fonctionnement

- Primitive de commande FORTRAN :

  CALL ENDEP (I)

  I (entier) : nombre d'actionneurs de déplacement en fonctionnement

g) Etat de la pince

Cette commande permet à l'utilisateur de savoir si la pince est en action :

- Commande : | 1 | 19 |

- Compte-rendu : | 1 | param1 |

  param1 = 0 pince à l'arrêt

  param1 = 1 pince en mouvement

- Primitive de commande :

  CALL ENACT (I)

  I (entier) : prend pour valeur 0 si la pince est à l'arrêt, 1 si elle est en fonctionnement.

4. STRUCTURE DU LOGICIEL

Le logiciel implanté sur le système Z80 réalise les fonctions suivantes :
- interprétation des commandes du Micro 1/03,
- envoi des impulsions aux actionneurs,
- simulation du retard entre l'arrêt logique et l'arrêt physique des actionneurs (cf. chapitre 4 figure 10).

A chacune de ces fonctions correspond une ou plusieurs tâches qui sont activées sur interruption. Lorsqu'aucune tâche n'est active, le système exécute l'instruction HALT ; ceci a pour effet d'allumer le voyant jaune en face avant du système Z80 correspondant.

4.1. Interprétation des commandes du Micro 1/03

Une interruption est provoquée par l'interface de communication lorsque le Micro 1/03 affiche le premier mot de la commande. Cette interruption active la tâche d'interprétation qui se déroule en trois phases :

(1) L'ensemble des mots constituant la commande est rangé dans un tableau. Les échanges avec le Micro 1/03 se faisant par interruption, les interruptions du port d'entrée du Micro 1/03 sont alors interdites.

(2) Un analyseur rudimentaire appelle les procédures correspondant à la commande ; par exemple : activation des tâches d'envoi de pas aux actionneurs.

(3) Le compte-rendu est transmis au Micro 1/03, puis on autorise de nouveau les interruptions du Micro 1/03.
[Note : Dans le cas d'une commande d'exécution, LANDEP ou LANACT, ce compte-rendu suit immédiatement la réception de la commande. Il n'y a donc pas attente de la fin du déplacement ou de l'action].

4.2. Envoi des pas aux moteurs

Lors de l'interprétation d'une commande de lancement du mouvement du robot ou de l'action de la pince, l'interpréteur démasque les interruptions engendrées par les décompteurs. Une interruption provoquée par le décompteur associé à l'actionneur fait exécuter le programme suivant :

Exemple pour l'actionneur n° 0

```
IMOTO    EG $            ; point d'entrée                    nombre de cycles
         OUT (MOTO), A   ; sortie d'un pas                    19
         DEC (IX+1)      ; décrémente les poids faibles       4
         JR Z,DECFORT    ;                                    21
         EI              ; retour dans le                     7 ou 11
         RETI            ; programme interrompu               4
DECFORT  DEC (IX)        ; décrémente poids fort              14
         JR M,FIN        ; supprimer cette tâche              24
         EI              ; retour dans le                     7 ou 11
         RETI            ; programme interrompu               4
FIN      EQ $
```

Le registre IX pointe sur la table contenant le nombre de pas qu'il reste à faire à chaque moteur. Ce registre ne doit pas être utilisé par l'ensemble des autres programmes. Compte tenu de cette convention et du mode de génération des pas (cf. § 1.3.a), cette procédure s'exécute sans sauvegarder, ni restituer aucun registre. Elle occupe en moyenne 70 cycles d'horloge. L'horloge du système Z80 battant à 4 MHZ, le système peut donc engendrer en les comptant des impulsions à la fréquence de 50 KHZ. En commandant tous les moteurs à pleine vitesse, ces tâches occupent de l'ordre de 30 % du temps d'occupation de la machine.

4.3. Simulation du temps de freinage d'un actionneur

Lorsqu'une tâche d'envoi d'impulsions se désactive (envoi du dernier pas), le moteur est logiquement arrêté. Pour tenir compte des rampes de décélération (chapitre 4, figure 10), le système intercale une temporisation entre cet instant et l'instant où le moteur sera considéré comme arrêté physiquement.

ANNEXE C

CONNEXION DE LA CAMERA CCD AU SOLAR 16/65

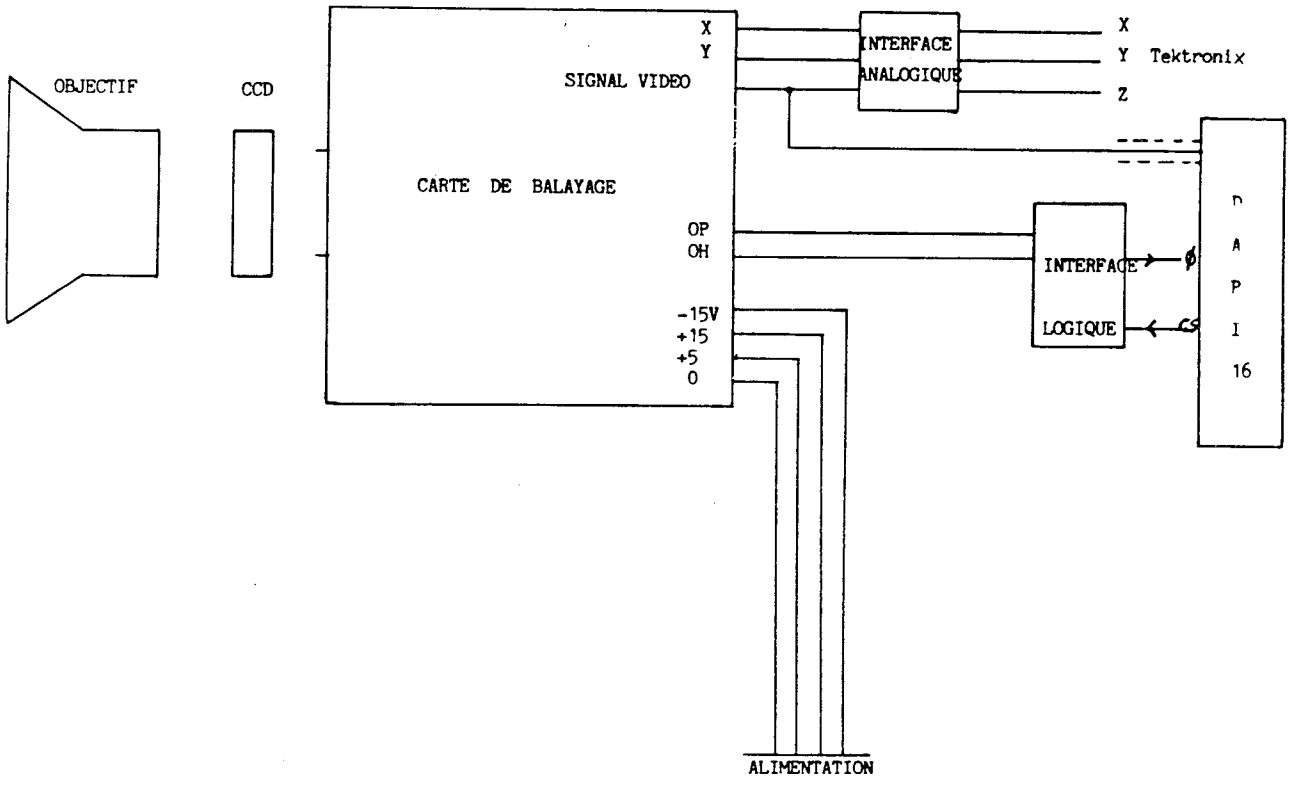


Figure 1

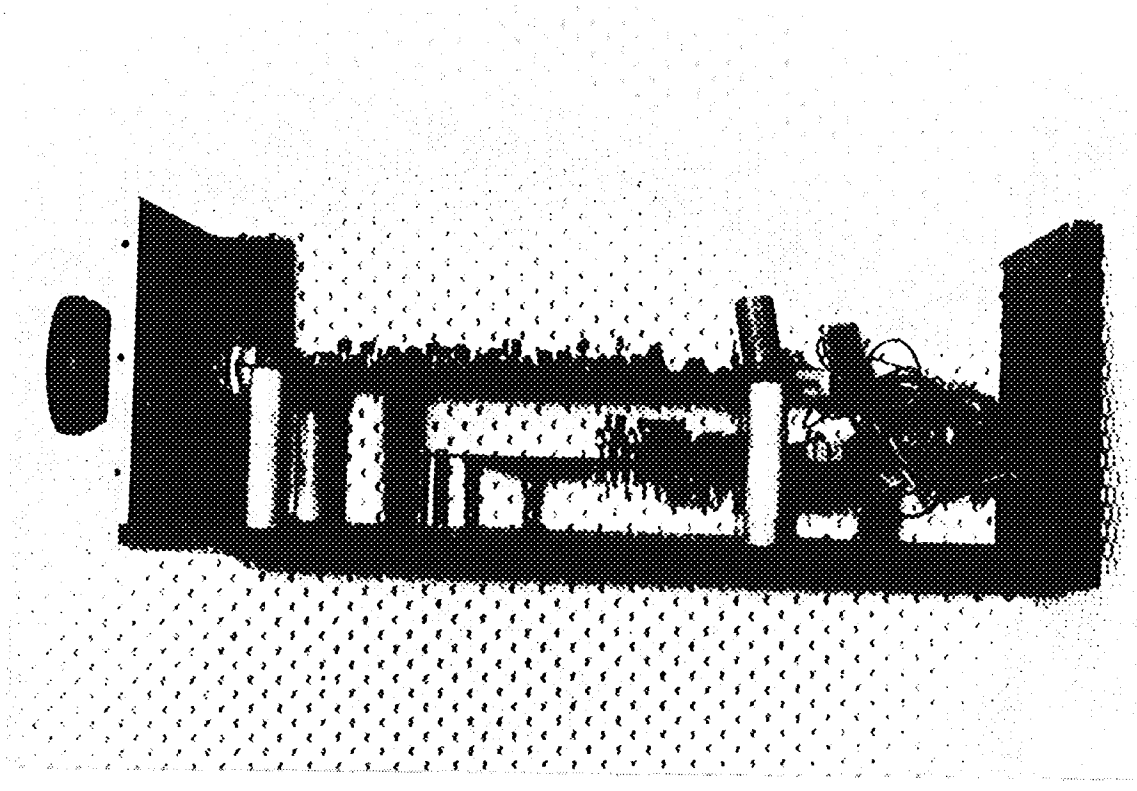


Figure 2

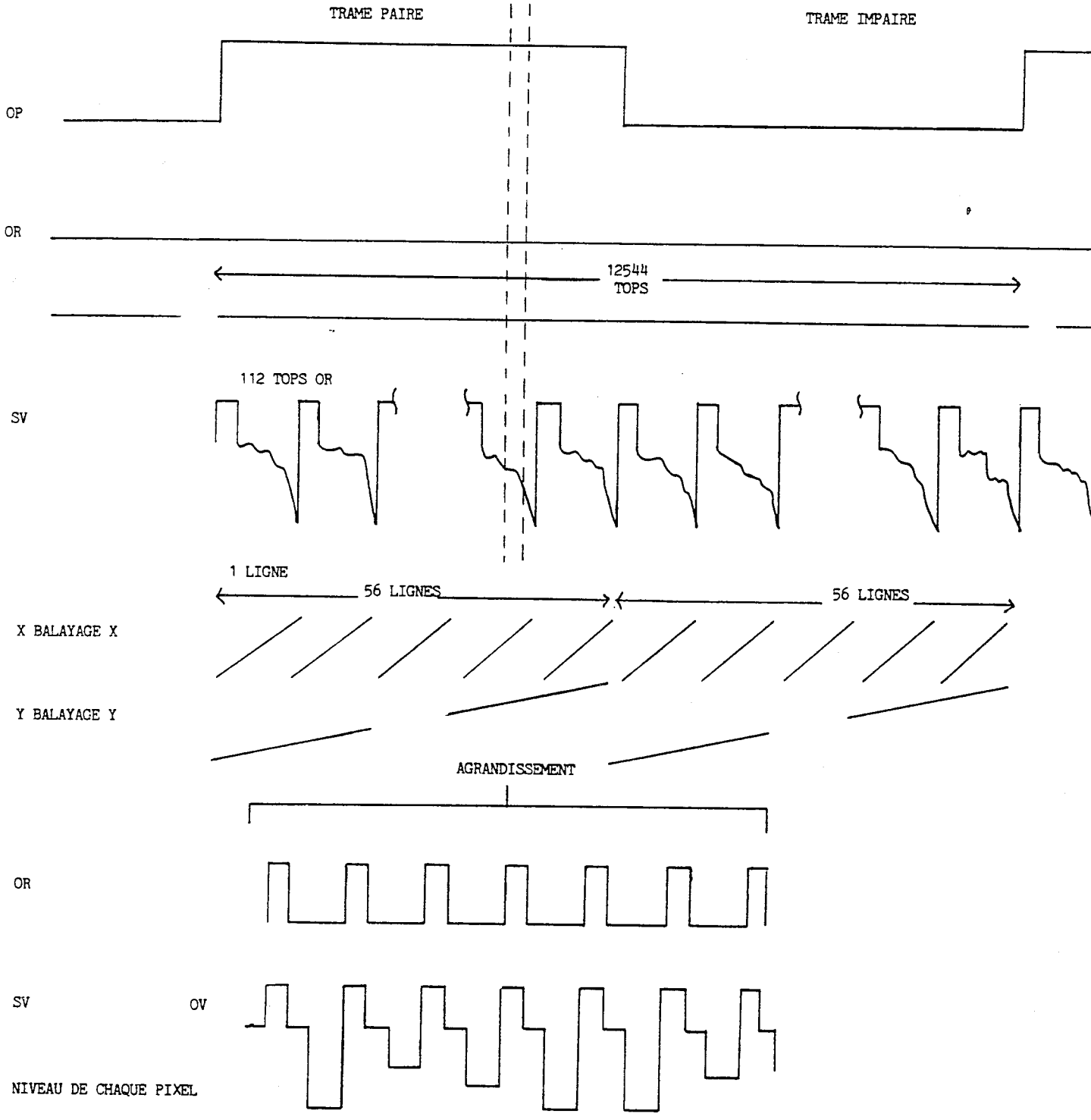

TRAME PAIRE
TRAME IMPAIRE
OP
OR
12544
TOPS
112 TOPS OR
SV
1 LIGNE
56 LIGNES
56 LIGNES
X BALAYAGE X
Y BALAYACE Y
AGRANDISSEMENT
OR
SV
OV
NIVEAU DE CHAQUE PIXEL

Figure 3

1. PRESENTATION DU MATERIEL

La figure 1 représente l'ensemble du matériel constituant le système de vision I.

La carte de balayage construite par la Société FAIRCHILD engendre les horloges et les différentes tensions utiles au fonctionnement du CCD. Le boîtier (figure 2) a été construit par l'atelier de Mécanique du Laboratoire de Spectroscopie de l'USMG. Nous avons utilisé un objectif d'une caméra vidéo de focale 6.5 mm muni d'un diaphragme. Un oscilloscope à trois entrées X, Y, Z (Tektronix 620) sert d'écran de visualisation. Nous utilisons le convertisseur analogique-digital inclus dans le coupleur d'entrée-sortie DAPI 16 du SOLAR 16/65 pour acquérir l'image. Deux interfaces permettent de relier la carte de balayage au moniteur et au calculateur.

2. LES SIGNAUX DE SORTIE DE LA CARTE DE BALAYAGE

La figure 3 représente les signaux que nous avons utilisés.

- Le balayage complet de la matrice CCD se fait en deux trames qui correspondent aux lignes paires et impaires de la matrice. Le niveau de l'horloge OP indique la trame en cours de scrutation.
- L'horloge OR est l'horloge de base du système.A chaque battement de cette horloge correspond la sortie d'un nouveau point (pixel) de l'image. Un balayage complet de la matrice produit 12544 points (112x112). La fréquence de cette horloge est de 1,2 MHZ.
- A chaque pixel correspond la sortie d'un signal représenté sur la figure 4.

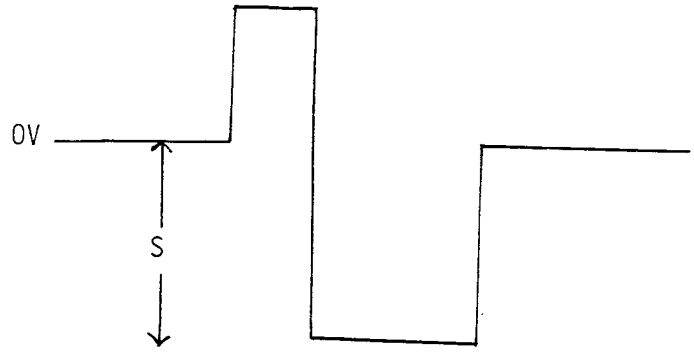


Figure 4

En régime normal (non saturé), l'intensité lumineuse reçue par le pixel entre deux balayages est proportionnelle à la différence de potentiel S.

- Les signaux X et Y servent à engendrer le balayage sur l'écran du moniteur. On remarque que chaque trame est composée de 56 lignes et que chaque ligne comporte 112 points.

3. LE CONVERTISSEUR ANALOGIQUE-DIGITAL

Ce convertisseur est muni d'un dispositif permettant de régler par programme :

- la résolution,
- le gain de l'amplificateur d'entrée,
- le mode d'échantillonnage.

a) Résolution

Dans notre application, nous utilisons la plus faible résolution, soit 8 bits (7 bits + 1 bit de signe).

b) Gain de l'amplificateur

Ce gain fait partie des paramètres du programme d'acquisition d'une image. L'utilisateur peut ainsi adapter le gain à la lumière ambiante.

c) Mode d'échantillonnage

L'échantillonnage et la conversion du signal peuvent être commandés :

- soit par une horloge interne du SOLAR 16/65,
- soit par une horloge externe.

L'acquisition de la donnée se fait en mode canal. La fréquence maximum de conversion dépend du convertisseur et du coupleur ; elle est actuellement de 100 KHZ.

Dans notre application, les "tops" d'échantillonnage sont engendrés par l'interface logique à la fréquence de 60 KHZ et servent d'horloge externe pour l'acquisition.

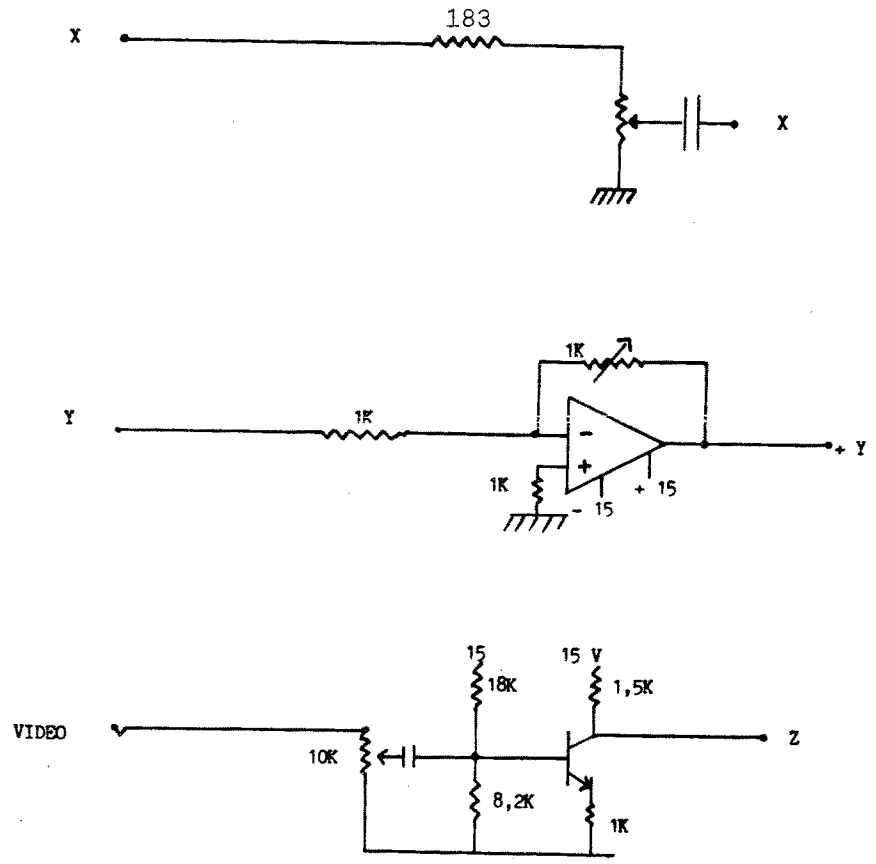


Figure 5

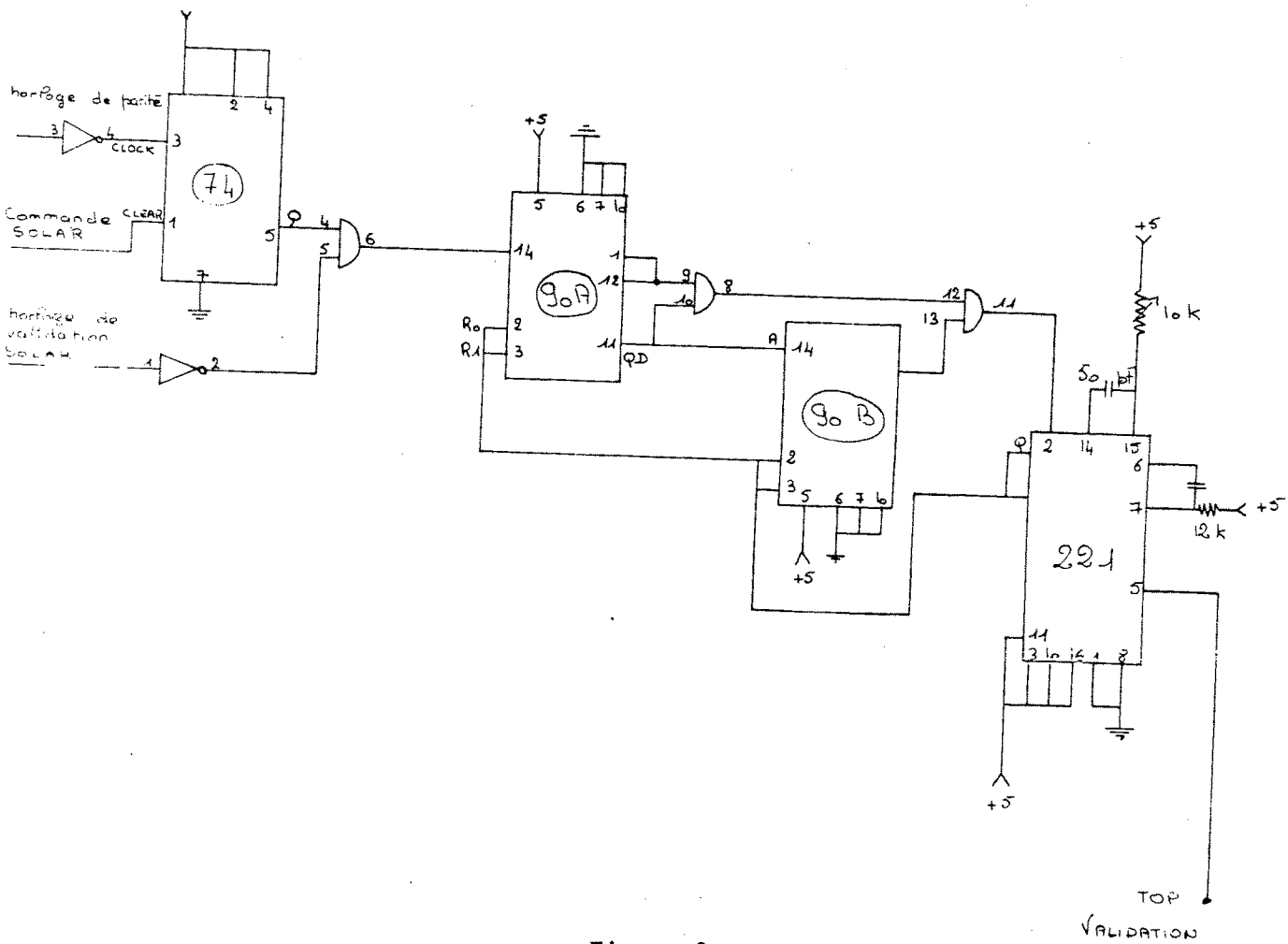


Figure 6

4. INTERFACES

Pour adapter les sorties de la carte de développement au moniteur Tektronix et au SOLAR, l'Atelier de Micro-informatique a construit deux interfaces que nous décrivons ici.

a) Interface analogique

La figure 5 donne le schéma de principe de cet interface. Le but est d'obtenir une image positive sur le moniteur et d'adapter le balayage de l'écran aux dimensions de celui-ci.

b) Interface logique

La figure 6 donne le schéma de principe de cet interface. Il se compose d'un décompteur par 19. Le décomptage des tops de l'horloge OR ne se fait que si la ligne CS est à 1 et ne commence que sur un front montant de l'horloge OP. Le monostable sert à placer le front montant de OV en coïncidence avec la partie valide du signal vidéo (cf. figure 7).

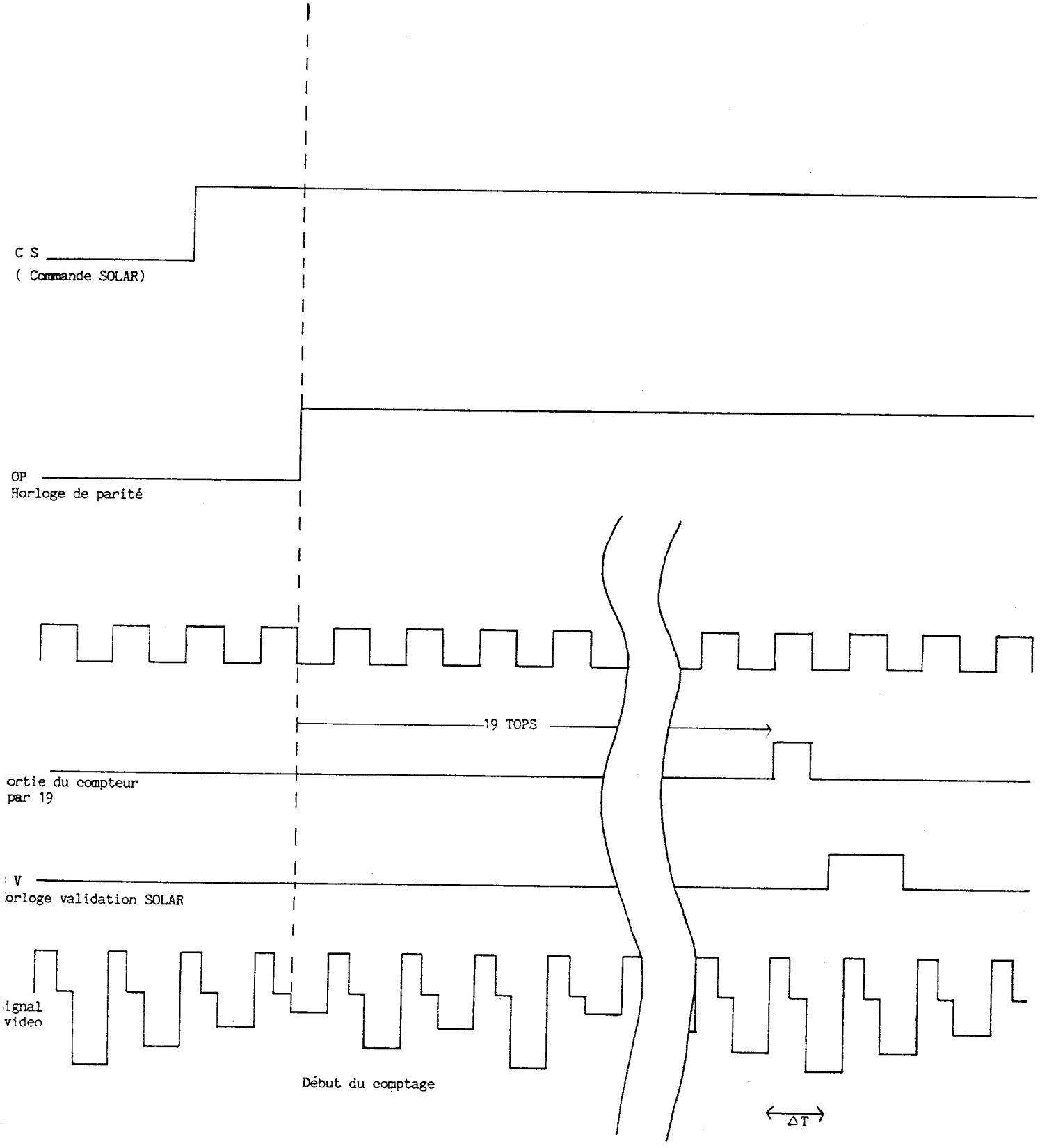
5. ACQUISITION DE L'IMAGE

Nous utilisons une ligne de la sortie logique du DAPI 16 pour indiquer que le coupleur est prêt à réaliser la conversion analogique-digitale. Lorsque l'interface logique détecte un niveau 1 sur cette ligne, elle attend le début d'une trame paire (front montant de OP) pour commencer à échantillonner l'horloge de référence OR à raison de 1 top sur 19.

Le SOLAR 16-65 fait donc l'acquisition à la cadence de 60 KHZ en 19 balayages complets de l'image ; 19 étant premier avec 12544, aucun point n'est pris deux fois durant les 12544 premières acquisitions.

A la suite de cette acquisition, trois traitements successifs sont appliqués aux données avant d'exécuter des programmes d'analyse d'image :

- réordonnancement des points de l'image dans l'ordre où ils sont fournis par la carte de balayage,
- suppression des points n'apparaïssant pas dans la matrice 100x100,
- entrelacement des deux trames.



ΔT Retard introduit par le monostable

Figure 7

RÉFÉRENCES

[1] P. ANDRE : "Méthode et structure de coordination des actionneurs d'une prothèse de bras", 2ème Congrès AFCET-IRIA Reconnaissance de formes et Intelligence Artificielle, Toulouse, Septembre 1979.

[2] P. COIFFET : "Sur la structure et la commande par calculateur numérique des robots-manipulateurs", Carrefour sur la Robotique Industrielle, INSA, Lyon, Juin 1980.

[3] S. FAHLMAN : "A planning system for robot construction tasks", Artificial Intelligence, n° 1, 1974.

[4] D. FALEK, M. PARENT : "LAMA-S : an evolutive Language for an intelligent robot", Séminaire International sur les méthodes et langages de programmation des robots industriels, IRIA, 27-29 Juin 1979.

[5] R.A. FINKEL : "Constructing and debugging manipulator programs", Ph.D. Dissertation, Stanford AI Laboratory Memo AIM-284, Stanford, USA, Août 1976.

[6] A. FOURNIER : "Génération de mouvements en robotique. Applications des inverses généralisées et des pseudo-inverses", Thèse d'Etat, Université des Sciences et Techniques du Languedoc, Montpellier, 2 Avril 1980.

[7] R. GINI, D. GIUSE : "MAL : linguaggio di programmazione per robot - Descrizione generale", Rapport interne du Laboratoire de Calcul du Polytecnico de Milan, 1978.

[8] J.C. LATOMBE, A. LUX : "Intelligence Artificielle et Robotique Industrielle", Le Nouvel Automatisme, Mai (1ère partie) et Juin-Juillet (2ème partie) 1979.

[9] 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, 27-29 Juin 1979.

[10] J.C. LATOMBE, E. MAZER : "Définition d'un langage de programmation pour la robotique (LM) - Analyse fonctionnelle de l'interpréteur". Rapport de Recherche IMAG n° 197, Mars 1980.

[11] J.C. LATOMBE et al. : "Intelligence Artificielle et Robotique : le projet PANDORE - 1er rapport", Rapport de Recherche IMAG, 1981 (à paraître).

[12] A. LUX : "Etude d'un modèle abstrait pour une machine LISP et de son implantation", Thèse de 3ème Cycle, Grenoble, Mars 1975.

[13] M.T. MASON : "Compliance and force control for computer controlled manipulators", MIT AI Lab, Technical Report 515, Avril 1979.

[14] P. MOLINIER : "Poursuite de trajectoires par le manipulateur MA 23 commandé par calculateur. Mise en évidence des problèmes temporels posés par la commande des robots par calculateur numérique", Thèse de Spécialité, Montpellier, Novembre 1977.

[15] N.J. NILSSON : "Artificial Intelligence", IFIP Congress, Stockholm, Août 1974.

[16] R. PAUL : "WAVE : a model based language for manipulator control", The Industrial Robot, Mars 1977.

[17] R. PAUL : "Manipulator path control", Proceedings of the IEEE International Conference on Cybernetics and Society, San Francisco, Septembre 1975.

[18] V. QUINT : "Application bureautique sur un réseau local", Bureautique 80, Paris, Avril 1980.

[19] B. RAPHAEL : "The relevance of robot research to Artificial Intelligence", SRI, AI Center, Technical Note 13, 1969.

[20] M. RENAUD : "Calcul de la matrice jacobienne nécessaire à la commande coordonnée d'un manipulateur", Mechanism and Machine Theory (à paraître).

[21] L.G. ROBERTS : Machine perception of three-dimensional solids", dans "Optical and Electro-optical information processing", J.T. Tipett et al., MIT Press 1965.

[22] C. ROSEN et al. : "Exploratory research in advanced automation", SRI Report, Août 1974.

[23] B.E. SHIMANO : "The kinematic design and force control of computer controlled manipulators", Stanford AI Lab, Memo AIM-313, Mars 1978.

[24] R.H. TAYLOR : "Synthesis of manipulator control programs from task-level specifications", Stanford AI Lab., Memo AIM-228, Juillet 1976.

[25] R.H. TAYLOR : "Planning and execution of straight line manipulator trajectories", IBM Journal of Research and Development, n° 4, Juillet 1979.

[26] UNIMATION, Inc. : "User's guide to VAL a robot programming and control system", Version 11, Unimation Inc., Shelter Rock Lane, Danbury, Conn., USA, Février 1979.

[27] D.E. WHITNEY : "The mathematics of coordinated control of prosthetics arms and manipulators", Transactions of ASME, Série F, Décembre 1972.

[28] P.H. WINSTON : "The psychology of computer vision", Mc Graw-Hill Book Co., New York, 1975.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

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

VU le rapport de présentation de Monsieur :

- J.C. LATOMBE, Maître-Assistant à l'Institut National Polytechnique de GRENOBLE

Monsieur Emmanuel M A Z E R

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Informatique".

Grenoble, le 6 Janvier 1981

Le Président de l'I.N.P.G.

Ph. TRAYNARD
Président
de l'Institut National Polytechnique