LM-OBJ : un langage de programmation des robots de niveau objet

Patricia Dubois

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE GRENOBLE

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Patricia DUBOIS

LM-OBJ : UN LANGAGE DE PROGRAMMATION DES ROBOTS DE NIVEAU OBJET

Les travaux relatifs au présent mémoire ont été effectués à *l'Institut National Polytechnique de Grenoble* (I.N.P.G.) au *Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle* (L.I.F.I.A.) sous la direction de Christian LAUGIER.

Résumé

La programmation automatique des robots est un des grands axes de la recherche en robotique. Dans ce cadre, les membres du LIFIA ont développé SHARP. Ce logiciel fournit à partir de la description de la tâche à réaliser et de la géométrie des objets à manipuler, le plan d'actions permettant de réaliser le montage. Ce plan d'actions est écrit en LM-OBJ (Langage de Manipulation de niveau OBJet).

Le travail présenté dans ce mémoire est la conception et la réalisation de l'interpréteur de LM-OBJ, permettant de faire exécuter par le robot les plans d'actions fournis par SHARP. Après avoir exposé comment se situe notre approche par rapport à celle plus classique des langages de manipulation (position relative entre deux objets plutôt que position absolue dans l'espace de travail en termes de repères cartésiens), nous décrivons les différentes instructions du langage : déplacements, saisies et lâchers d'objets, réalisation de contacts .... Afin de bien situer la place de LM-OBJ dans un logiciel de programmation automatique comme SHARP, nous exposons les réalisations antérieures dans le cadre de SHARP (planificateur, LM, fonctions compliantes).

La structure de la base de données, comprenant notamment la description géométrique des objets à manipuler, est exposée en détail. Nous présentons les diverses solutions envisagées permettant de traduire une instruction LM-OBJ en une commande exécutable par le robot (en terme de repère déplaçable et de transformation géométrique à lui appliquer). Nous exposons comment les contacts sont réalisés à l'aide d'un capteur de forces afin d'exécuter des mouvements fins d'assemblage.

Mots-clés : robotique - modélisation géométrique - interpréteur géométrique - langage de manipulation - compliance.

Keywords : robotic - geometrical modelling - geometrical interpretor - manipulating language - compliance.

Remerciements

Je tiens à remercier

* Monsieur *Philippe JORRAND*, directeur de recherche au CNRS, directeur du Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle, pour m'avoir accueillie dans son laboratoire.

* Monsieur *Christian LAUGIER*, chargé de recherche à l'INRIA, responsable de l'équipe robotique du LIFIA, pour avoir suivi mon travail tout au long de cette année.

* Monsieur *Claude KAISER* professeur au Conservatoire National des Arts et Métiers, président du jury.

* Monsieur *Louis BOLLIET* professeur émérite à l'Univerité des Sciences Sociales de Grenoble, Monsieur *Jacques COURTIN* professeur à l'Université des Sciences Sociales de Grenoble et Monsieur *Pierre PUGET* ingénieur à I.T.M.I., d'avoir bien voulu jugé ce travail.

J'aimerais aussi remercier

* les membres de l'équipe robotique du LIFIA qui ont contribué par leurs travaux à la réalisation du projet SHARP, et notamment :
Pascal DI GIACOMO,
Christian GANDON,
Emmanuel MAZER,
Michel PASQUIER,
Pascal THEVENEAU
pour leurs précieux conseils.

* les membres du LIFIA qui par leur bonne humeur ont su créer une ambiance de travail agréable.

Sommaire

Chapitre 1

INTRODUCTION

1.1 Le cadre de travail

Le travail présenté dans ce mémoire a été réalisé au sein de l'équipe robotique du Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle (LIFIA) de l'Institut National Polytechnique de Grenoble (INPG). Il se place dans le cadre de recherches menées au LIFIA, dont l'objectif à long terme est la réalisation d'un "robot intelligent" doué d'autonomie, de dextérité et de capacités évoluées de perception et de communication.

C'est ainsi qu'est né SHARP (High level System for Automatic Robot Programming), le projet de programmation automatique des robots sur lequel travaillent les membres de l'équipe robotique du LIFIA. Ce système est orienté vers la programmation des robots à postes fixes dans le cadre de la robotique d'assemblage.

1.2 Le contexte

1.2.1 Les langages de programmation des robots

Il est devenu classique de distinguer trois classes de langages de programmation des robots : la programmation par l'exemple, les langages de manipulation et la programmation automatique.

La programmation par l'exemple :

C'est la méthode de programmation des robots la plus ancienne. Elle consiste à faire exécuter au robot la tâche sous contrôle "manuel", par exemple à l'aide d'un boitier de commande. Les mouvements de tous les éléments du manipulateur

(déplacements des axes, ouverture de pince, ...) sont d'abord enregistrés ; ils sont ensuite reproduits automatiquement par le robot afin d'exécuter à volonté la même tâche.

Cette méthode demande peu de moyens informatiques. Elle est de plus très simple et facile à mettre en œuvre. C'est pourquoi elle est encore souvent employée dans l'industrie, chaque fois que la tâche exécutée le permet. Elle présente cependant d'importants inconvénients qui en limitent son domaine d'application :

* Il est impossible d'utiliser des données en provenance de capteurs, et donc d'adapter le comportement du robot aux variations de son environnement. Le programme est alors écrit pour une tâche bien précise, ne supportant aucune variation (mêmes objets, même position des pièces).
* Il est très difficile de modifier les opérations enregistrées du fait que celles-ci sont codées de manière numérique.
* On ne dispose pas de structures itératives ou conditionnelles.
* Le robot est mobilisé durant toute la phase de programmation.

Les langages de manipulation :

Les langages de manipulation tels que LM [Maz 81,Mir 84] ou VAL [SGS 84], sont en général construit autour de langages algorithmiques. Ils combinent des instructions classiques de ces langages (structures itératives, conditionnelles, calculs, affectations, ...) avec des constructions spécifiques à la robotique dont le rôle est de remplir les fonctions suivantes :

* La modélisation de l'univers en termes de repères cartésiens, indépendants de l'architecture mécanique du robot.
* La communication avec l'environnement du robot (lien avec les capteurs en particulier).
* La description des mouvements du robot et des actions de l'outil terminal.
* La coordination de plusieurs robots, ou du robot avec des systèmes mécaniques extérieurs (alimentation de pièces en particulier).

Le pouvoir d'expression de ce type de langage est sans commune mesure avec celui de la programmation "par l'exemple". Toutefois il faut noter que :

* Ces programmes ne peuvent être écrits que par des personnes connaissant l'informatique et la programmation.

* L'intégration et le traitement de données en provenance de capteurs contribue à rendre encore plus délicate la programmation d'une tâche de manipulation.

* La robustesse du programme écrit ne peut être obtenue qu'après de très nombreux tests et amendements du programme.

Ces inconvénients sont dus principalement à la pauvreté des modèles utilisés (repères cartésiens).

La programmation automatique :

Elle permet de s'affranchir des problèmes cités précédemment. Elle utilise pour cela des techniques de l'intelligence artificielle et de la modélisation géométrique, afin de simuler le raisonnement humain en vue de la réalisation de tâches d'assemblages. Le but du système est de permettre une programmation dite de niveau "tâche". L'utilisateur ne donne alors aucune description explicite des actions que le système doit exécuter, mais il spécifie la tâche en termes d'une séquence d'opérations à réaliser (par exemple : assembler objet-1 et objet-2). Seuls les objectifs à atteindre sont alors décrits au système qui se charge de trouver la "meilleure" façon d'y parvenir. Le système utilise dans ce but une description des composants à assembler par l'intermédiaire d'une base de données CAO.

La programmation automatique des robots pour les tâches d'assemblage soulève cependant de nombreux problèmes partiellement résolus à l'heure actuelle [Lau 87] : calcul de prises stables, détermination de trajectoires sûres pour le transport des objets dans un univers encombré et, planification des petits mouvements permettant le montage final.

Les principaux projets traitant de ces problèmes sont AUTOPASS [LW 77], LAMA [Loz 76], TWAIN [LB 85], HANDEY [Maz 87] et SHARP [LP 85] [Lau 87]. Ils mettent en jeu des modules de perception et de planification d'actions. Bien que des résultats significatifs aient été obtenus, la programmation automatique des robots reste encore au stade de la recherche.

1.2.2 Le système SHARP

SHARP (High level System for Automatic Robot Programming) est le projet de programmation automatique des robots développé au LIFIA. Il est orienté vers la programmation des robots à postes fixes dans le cadre de la robotique d'assemblage.

Objectif visé :

Le rôle du système est de produire automatiquement un programme de manipulation exécutable par le robot, à partir d'une description de l'environnement de travail (robot, objets, ...) et de la tâche à accomplir.

Entrées et sorties du système :

Afin que le système soit en mesure de produire un programme d'assemblage, il faut lui fournir les informations suivantes :

– *Description de l'espace de travail* en terme de la géométrie des objets à manipuler, des positions initiales des objets, des incertitudes de position, de la géométrie du robot et de ses propriétés cinématiques.

– *Description de l'assemblage à réaliser* en terme des relations d'assemblage qui relient les objets entre eux.

En sortie, le système produit un plan d'actions écrit dans un langage cible permettant de faire réaliser l'assemblage demandé par le robot. L'annexe A contient un exemple de plan d'assemblage produit par le système.

L'objectif du travail présenté dans ce mémoire est de développer ce langage cible.

Démarche suivie :

Il faut distinguer trois étapes dans la réalisation d'une tâche d'assemblage. Le premier problème qui se pose est la façon dont l'outil doit saisir l'objet à manipuler. Il faut ensuite le transporter jusqu'au lieu du montage et ce, tout en évitant d'éventuels obstacles. La phase finale est le montage proprement dit des deux objets.

SHARP commence par résoudre le problème de la saisie de l'objet par l'outil terminal du robot [LP 83,Per 86]. Comment l'approcher ? Comment le saisir en ayant une prise stable permettant ensuite de réaliser le montage souhaité ?

Pour produire les ordres d'assemblages proprement dits (description détaillée de toutes les étapes du montage final) le système étudie le plan de démontage des pièces afin d'en déduire par inversion le plan d'assemblage. Ce module de planification des mouvements fins [The 88] étudie la stratégie d'assemblage d'une position initiale au voisinage de la position finale jusqu'à cette dernière.

La recherche de trajectoires sans collision [Pas 89] permet de planifier le transport de l'objet depuis la position de saisie jusqu'à la position initiale d'assemblage et ce, tout en évitant les obstacles (autres objets) éventuellement présents sur le parcours.

Après planification des actions, une preuve de programme [Pug 85] s'assure de la validité globale du plan au regard des incertitudes de position dues aux mécanismes d'alimentation en pièces, aux résultats de la perception et aux erreurs de commande du robot.

1.3 Le problème traité

Le sujet traité dans ce mémoire est le développement de l'interpréteur du langage cible produit par le planificateur de SHARP : LM-OBJ.

Cet interpréteur est un module de SHARP (figure 1.1). Il permet l'exécution par le robot d'un plan d'assemblage produit par le système. Les plans d'actions sont écrits à l'aide d'un langage utilisant des constructions symboliques et un modèle de nature essentiellement géométrique de l'espace de travail. LM-OBJ a pour objet d'effectuer la liaison entre les ordres "géométriques" du plan d'actions et l'univers réel où évolue le robot. En d'autres termes, il doit produire des ordres numériques exécutables par le robot à partir d'une instruction du langage cible.

Les modèles géométriques décrivent les objets et leur position dans l'espace de travail. Ces modèles comportent des incertitudes vis à vis de l'univers réel : erreurs sur la localisation des objets ainsi que sur leurs dimensions. Pour pallier à ces insuffisances du modèle, certaines instructions du langage (celles qui mettent en jeu des contacts) utilisent des données en provenance de capteurs, elles permettent de "réduire" ainsi les effets des incertitudes.

LM-OBJ décrit les opérations à exécuter à l'aide de relations entre composants géométriques des objets (faces, arêtes, points). Elles décrivent la position relative entre les deux objets une fois le déplacement accompli. *Un déplacement est ainsi toujours décrit en terme de l'objectif visé.* Cet objectif est alors la réalisation d'une relation géométrique ou d'un contact.

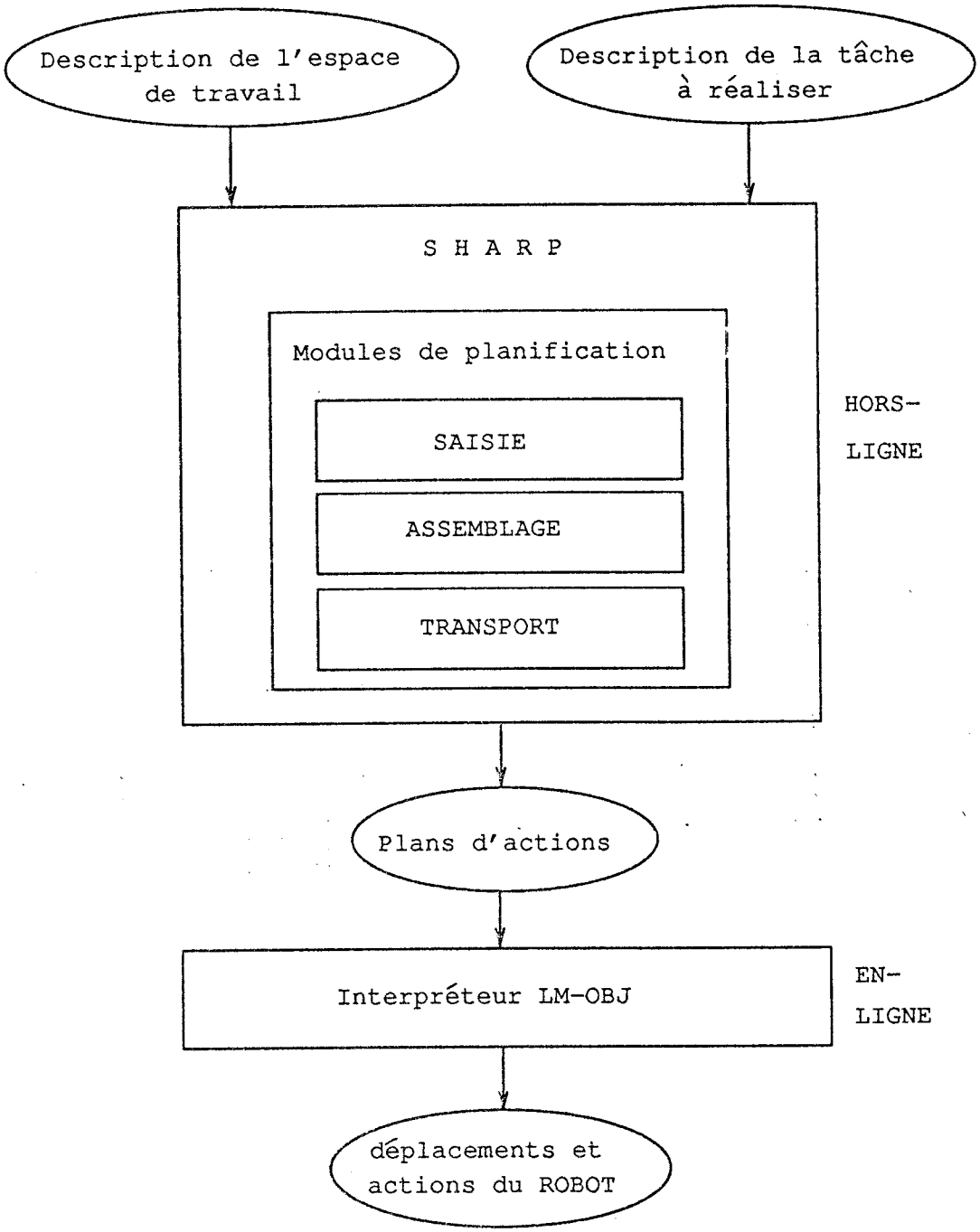


Figure 1.1: Structure fonctionnelle du système SHARP

Cette façon de procéder n'est pas réellement utilisée dans les langages classiques de programmation des robots, qui conduisent tous à spécifier les déplacements en termes de repères cartésiens et de transformations géométriques.

A partir de cette remarque nous avons pensé, que l'utilisation de modèles géométriques des objets et du langage cible de SHARP permettaient de définir un nouveau langage. Ce langage peut être qualifié de "géométrique" puisqu'il fait essentiellement usage de la géométrie des objets pour décrire un déplacement. Les différences avec les langages classiques de manipulation (Cf. paragraphe précédent) sont les suivantes :

* La position d'un objet dans l'espace s'exprime aussi (et surtout) à l'aide de *relations géométriques* (parallélisme, alignement) vis à vis d'un autre objet, et non plus uniquement en termes de position absolue dans l'univers.
* Les données en provenance de *capteurs sont gérées et analysées par le système*, l'utilisateur n'a pas à programmer leur traitement.
* Le modèle contient des informations beaucoup *plus riches* que celles fournies par la connaissance d'un repère (connaissances géométrique des objets et du robot en particulier). Le travail au niveau objet permet une interprétation plus claire des données sensorielles.
* La présence de modèles d'origine CAO peut éventuellement permettre une programmation graphique.

1.4 L'approche utilisée

Notre démarche consiste, à l'aide des modèles d'origine CAO, à traduire les déplacements exprimés en terme de relations géométriques, en instruction de mouvement LM dans laquelle est précisé le repère cartésien à déplacer et la transformation géométrique à lui appliquer. Dans le cadre de déplacements engendrant des contacts, le système met en œuvre des données en provenance d'un capteur de forces à six composantes.

LM-GEO [Maz 82b] est aussi un langage de type "géométrique", il interprète les relations afin de les traduire en instructions LM. Toutefois il n'utilise pas de capteur.

Un langage de même nature est actuellement en cours de développement au L.A.A.S. (Laboratoire d'Automatique et d'Analyse des Systèmes) [GS 88], la tra-

duction des relations se fait alors directement en commandes de vitesse et de position sur les axes du robot.

1.5 Plan du document

Le chapitre deux décrit le "cahier des charges" du langage en termes des objectifs visés. Il expose ce qui a déjà été réalisé dans le cadre du projet SHARP et qui servira de base à ce travail.

Les différentes solutions envisagées pour mener à bien ce projet sont exposées au chapitre trois, dans lequel sont aussi justifiés les choix parmi ces solutions conduisant à la réalisation finale.

Le chapitre quatre présente de quelle façon le travail a été réalisé techniquement.

Le chapitre cinq conclu ce rapport en évaluant le logiciel écrit et en présentant les points qui mériteraient une réflexion plus approfondie en vue de son amélioration.

Chapitre 2

OBJECTIFS ET CONTRAINTES

2.1 Le problème traité

Le but du travail est le développement d'un langage de programmation des robots de niveau objet. Ce langage doit à la fois pouvoir être utilisé pour exécuter les plans d'actions produits par SHARP. Il peut également être employé pour programmer le robot, lorsque l'on dispose des modèles géométriques nécessaires.

Utilisation dans l'environnement SHARP :

La fonction première de ce langage est d'interpréter le plan d'actions produit par le planificateur sous l'environnement SHARP, afin de le traduire en commandes exécutables par le robot.

Ce langage comporte donc des instructions permettant la réalisation de mouvements de transfert d'objets, d'opérations de prise d'objets dans la pince du robot (ainsi que l'opération inverse qui consiste à lâcher des objets tenus dans la pince) et de petits mouvements contrôlés par un capteur de force afin de réaliser l'assemblage de deux pièces (mouvements fins).

Utilisé sous SHARP, ce langage travaille à partir de modèles d'origine CAO décrivant l'espace de travail du robot (objets, position, ...) et des instructions produites automatiquement par le planificateur définissant une tâche d'assemblage à réaliser. Le rôle de l'interpréteur du langage (dit langage cible de SHARP) est de faire exécuter les instructions décrites dans un environnement symbolique, dans l'espace physique et réel où évolue le robot.

La cohérence du modèle géométrique de l'univers doit être maintenue par l'interpréteur du langage après exécution de commandes de déplacement sur le robot, afin de disposer en permanence de la position "réelle" des objets.

Utilisation comme un langage de commande de robots :

Le langage développé peut être utilisé hors de l'environnement SHARP à condition de disposer des modèles CAO nécessaires. Il posséde en effet toutes les caractéristiques d'un langage de commande des robots, avec les principales différences suivantes :

- La présence de modèles d'origine CAO permet une spécification des mouvements non plus uniquement en entités abstraites comme les repères cartésiens, mais aussi en terme de relations géométriques et de contacts à réaliser.

- Un langage de ce type est de niveau plus élevé que les langages de manipulation traditionnels, car il comporte des modèles plus riches : description de la forme des objets, de leur composition, de leur position dans l'univers, ... Le travail au niveau objet permet une interprétation plus claire des données sensorielles.

- L'utilisation de modèles CAO peut permettre de réaliser une simulation graphique, afin de vérifier la cohérence du programme (collisions en particulier), avant de l'exécuter sur le robot.

2.2 Principe de spécification des opérations du robot

2.2.1 L'approche classique

Dans les langages de manipulation, une instruction de déplacement s'exprime à l'aide d'une des deux instructions suivantes :

DEPLACER *repère-1* A *repère-2*
DEPLACER *repère-1* DE *trsf*

où :

repère-1 est un repère attaché à un objet mobile, c'est à dire un élément du robot ou une pièce tenue dans l'outil terminal du manipulateur ;

repère-2 est soit un repère attaché à un objet non susceptible de bouger au cours du déplacement, soit un repère dans l'espace de travail spécifiant la position à atteindre ;

trsf est une transformation géométrique précisant le déplacement relatif par rapport à *repère-1*.

Exemple :

Soient un objet A mobile sur lequel est attaché un repère *repère-1* (figure 2.1) et un objet B fixe sur lequel on indique le repère *repère-2* (figure 2.2).

L'exécution de l'instruction

DEPLACER *repère-1* A *repère-2*

conduit à amener le *repère-1* en coïncidence avec le *repère-2*. La situation obtenue est alors celle décrite dans la figure 2.3.

Notons ici les positions particulières des repères *repère-1* et *repère-2*. Si nous avions souhaité un montage avec calage sur les points *P-2* et *P-1*, il aurait fallu décrire deux autres repères différents de *repère-1* et *repère-2*.

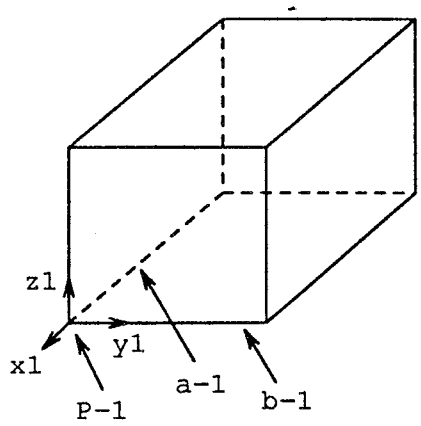


Figure 2.1: Représentation de l'objet A

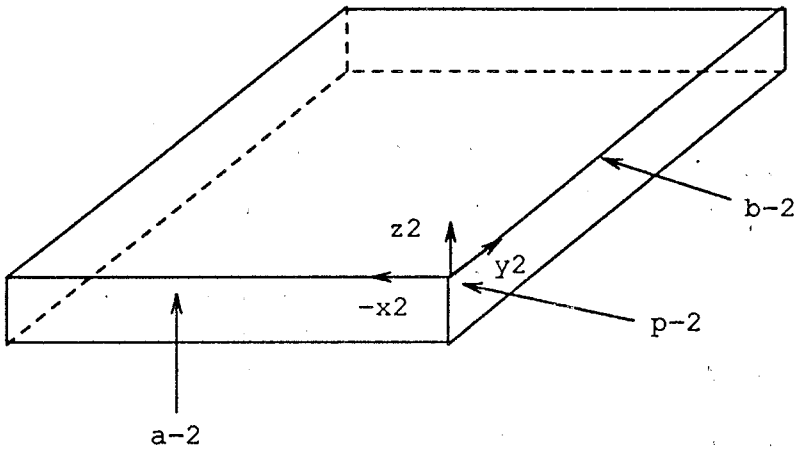


Figure 2.2: Représentation de l'objet B

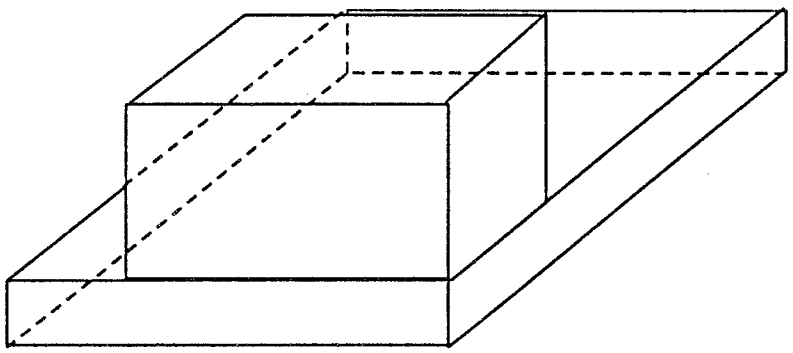


Figure 2.3: Position des deux objets après exécution du mouvement

Afin de positionner deux objets l'un par rapport à l'autre, il faut donc placer un repère à un endroit précis sur chaque objet, ce qui n'est pas une opération facile. En effet, la description numérique d'un repère nécessite la connaissance :

– *Des coordonnées de son origine* exprimées en valeur absolue (coordonnées dans un repère de référence fixe) ou en valeur relative (translation par rapport à un autre repère).

– *De l'orientation des axes du repère* représentés par des angles ou des vecteurs exprimés dans le repère absolu ou dans un autre repère.

L'origine du nouveau repère pourrait être confondue avec un point de l'objet (par exemple dans la figure 2.1 le point *P-1*), mais *nous ne disposons pas*, dans les langages de manipulation classiques, *de la connaissance géométrique des objets* ; le point *P-1* de l'exemple n'est donc pas connu à priori par le système.

Les coordonnées de ce point doivent alors être calculées soit en valeur absolue, soit relativement à un autre repère attaché à l'objet. Ceci ne peut se faire que par des mesures sur l'objet et dans l'environnement de travail, ce qui est d'autant plus difficile que *les repères sont des entités non matérielles.*

Les mêmes problèmes se posent (avec une difficulté supplémentaire due à la "mesure" des angles) au moment de la détermination de l'orientation du repère.

2.2.2 Notre approche

Principe :

Au regard de l'exemple précédent, et sachant que sous l'environnement SHARP la géométrie des objets est connue, l'opération de manipulation aurait pu être décrite de la façon suivante :

arête a-1 de l'*objet-A* sur *arête a-2* de l'*objet-B*
et

arête b-1 de l'*objet-A* sur *arête b-2* de l'*objet-B.*

L'idée essentielle de notre approche est de pouvoir demander une action du robot exprimant symboliquement une position relative entre deux objets. Nous avons donc la possibilité *d'exprimer un déplacement en terme de ses effets sur les objets manipulés.* C'est la raison pour laquelle nous parlons de programmation de niveau

objet.

Définitions :

Nous appelons *situation géométrique* la position relative entre deux objets. Elle est décrite par un ensemble de relations géométriques qui expriment une position relative entre entités géométriques (faces, arêtes, sommets).

Ces entités sont de deux natures :

– *physiques* : elles correspondent à des éléments de la surface des objets manipulés, et elles sont représentées par des entités géométriques finies telles que faces, arêtes ou sommets ;

– *virtuelles* : ce sont des entités infinies telles que points, droites, plans. Ces entités n'ont pas d'existence matérielle. Elles correspondent à des éléments caractéristiques de la géométrie des objets (par exemple axe de symétrie).

De cette façon nous définissons aussi bien une situation de contact entre objets, à l'aide de relations géométriques entre entités physiques, qu'une situation de positionnement dans l'espace libre lorsque les relations mêlent entités physiques et entités virtuelles.

Le principe de description des situations géométriques est la mise en correspondance d'entités géométriques, par exemple, *entité-1* sur *entité-2* et *entité-3* sur *entité-4*. Chaque correspondance entre deux entités géométriques est une relation géométrique. Une situation est décrite à l'aide d'un ensemble de relations géométriques.

Afin de passer à une commande de mouvement exécutable par le robot, l'interpréteur de langage calcule à partir d'une situation géométrique décrivant l'objectif d'un déplacement, le repère à déplacer et la transformation à lui appliquer. Dans l'implantation actuelle du système, les paramètres sont fournis à un interpréteur de commandes LM pour la réalisation physique du déplacement par une instruction du type :

DEPLACER *repère* DE *transformation.*

Syntaxe d'une situation géométrique :

<situation-géométrique> ::= (<relation-géométrique>$^{+}$)
<relation-géométrique> ::= (<entité-géo-virtuelle> SUR <entité-géo-virtuelle>) |
(<entité-géo-virtuelle> SUR <entité-géo-réelle>) |
(<entité-géo-réelle> SUR <entité-géo-virtuelle>)
<situation-de-contact> ::= <relation-géo.-contact> $^{+}$ { <relation-géométrique> $^{+}$ }
<relation-géo.-contact> ::= (<entité-géo-réelle> SUR <entité-géo-réelle>)
<entité-géo-réelle> ::= nom-de-face | nom-d'arête | nom-de-sommet
<entité-géo-virtuelle> ::= plan | droite | point

Remarque 1 :

Lorsque le langage est utilisé en tant que langage de commande de robots hors de l'environnement SHARP, l'utilisateur doit avoir facilement accès au nom des entités géométriques. Il faut donc prévoir qu'il puisse les nommer lui-même afin d'éviter qu' elles soient accessibles par des noms, non significatifs au regard de l'objet (par exemple *entité-56*), engendrés automatiquement par le système à la création du modèle géométrique.

Remarque 2 :

Dans le cas d'objets ayant des symétries (par exemple cube, rectangle, cylindre) il serait agréable de disposer d'une decription fonctionnelle de l'objectif à atteindre. Par exemple, mettre un cube sur un plan pourrait se traduire par une relation du type (Cube SUR Plan). Le système se chargeant alors de poser le cube sur le plan par sa face la plus proche du plan. Or, une telle relation n'est actuellement pas permise dans LM-OBJ, et les situations géométriques permettant de décrire une telle opération peuvent sur-contraindre la position finale.
Exemple : Soit la description fonctionnelle : poser le cube *Cube* sur le plan *Plan* (figure 2.4), l'instruction LM-OBJ décrivant cela utilise une des situations géométriques suivantes :

(*Cube-f-i* SUR *Plan*) avec i $\in$ [1,6] où *Cube-f-i* est la ième face du cube.

Ces relations conduisent toutes au même résultat final du cube sur le plan : quelque soit la face sur le plan nous ne voyons aucune différence, mais elles imposent

des déplacements différents. Et ce peut être gênant si on n'y prend garde (par exemple : retournement du cube pour le poser).

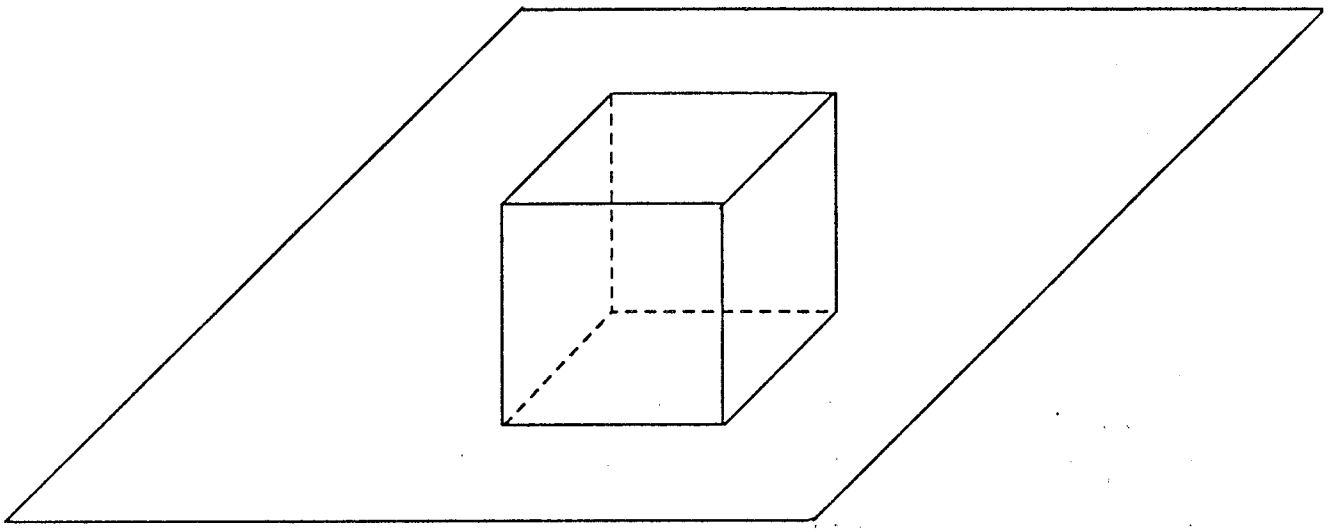


Figure 2.4: Cube posé sur le plan

2.3 Les instructions du langage

Les instructions nécessaires dans ce langage peuvent être classées de la façon suivante :

Les mouvements de transfert : Ils permettent d'atteindre une position dans l'espace en se déplaçant d'un point à un autre d'un environnement encombré, et ce tout en évitant d'éventuels obstacles.

Les opérations de saisie et de lâcher : Ce sont celles qui permettent de prendre un objet dans la pince du robot pour ensuite pouvoir le déplacer, ou de lâcher un objet tenu dans la pince afin de le déposer, par exemple après un montage.

les mouvements fins au contact : Ce sont de petits mouvements utilisant les données d'un capteur et permettant de réaliser le montage final de deux objets.

Les mouvements fins correctifs : Ce sont de petits mouvements permettant de corriger une situation géométrique mal réalisée à cause d'incertitudes diverses, ce qui permet ensuite de mener à bien l'assemblage.

2.3.1 Les mouvements de transfert

Les mouvements de transfert sont des déplacements permettant d'atteindre une position dans l'espace. Ils sont effectués à grande vitesse car ils ne présentent (en principe) pas de risque de collision, et ils sont insensibles aux incertitudes de position.

Ces mouvements appelés aussi grands mouvements, donnent la possibilité de déplacer des objets, de positionner le robot dans une configuration précise. Dans ces déplacements, une situation s'exprime par un ensemble de relations géométriques ou une mise en coïncidence de repères (aucun contact n'étant mis en jeu).

Une configuration du manipulateur est donnée par la position de chacun des éléments de la chaîne articulée que constitue le robot. Les mouvements de transfert permettent l'utilisation de coordonnées articulaires, ce qui offre la possibilité de positionner le robot dans des configurations précises. Ceci est important pour la programmation automatique, car la planification s'effectue en raisonnant en terme de configurations du manipulateur plutôt qu'en terme de positions à atteindre.

Un paramètre "chemin" permet alors de préciser des contraintes de trajectoire

à respecter (situations intermédiaires) lors de l'exécution du mouvement. Cette méthode permet de suivre un parcours donné qui évite les obstacles.

Syntaxe de l'instruction :

Une instruction de transfert du langage a la syntaxe suivante :

(REALISER-S <situation-géométrique> { <chemin> })

<chemin> ::= (VIA <situation-intermediaire>$^+$)
<situation-intermediaire> ::= <situation-géométrique> | <coordonnées-articulaires>
<coordonnées-articulaires> ::= (θ_1 θ_2 θ_3 θ_4 θ_5 θ_6) avec θ_i angle sur l'axe i du robot à six degrés de liberté.

la syntaxe d'une situation-géométrique est décrite dans le paragraphe précédent (voir 2.2.2).

Les restrictions actuelles :

Dans ce travail nous ne considérons que des objets dont les *faces sont planes* et les *arêtes droites* ; ceci conduit à des équations géométriques plus simples à résoudre.

Le rôle de l'instruction REALISER-S est de réaliser une situation géométrique dans l'univers, mise en coïncidence d'entités géométriques physiques ou virtuelles. Dans cette optique les faces et arêtes des objets ne sont considérés que par leur plan ou droite porteur, donc infinis, pas de délimitation des arêtes ou des faces.

Quelques extensions à la syntaxe :

Afin de prendre en compte des instructions de plus "bas niveau ", nous avons ajouté trois possibilités supplémentaires pour la description d'une situation :

* L'utilisation de la mise en correspondance de repères, (repère-1 SUR repère-2).
* L'expression d'une position en coordonnées articulaires. Cette possibilité était déjà permise pour décrire un chemin.
* La possibilité de décrire un déplacement par un repère déplaçable et la transformation à lui apliquer.

Nous obtenons alors la syntaxe suivante pour une situation géométrique "étendue" utilisée dans le cas d'un mouvement de transfert :

<situation> ::= <situation-géométrique> | (repère SUR repère) |
<coordonnées-articulaires> | (repère transformation)

2.3.2 Les opérations de saisie et lâcher

Elles permettent la prise d'un objet par l'outil terminal du robot ou le lâcher de cet objet afin de le déposer.

• **La saisie**

Une instruction de saisie permet de prendre un objet dans la pince du robot. Cependant, avant de saisir l'objet, il faut positionner correctement la pince par rapport à cet objet, aussi l'instruction de saisie exécute-t'elle d'abord un déplacement avant de prendre l'objet.

– *Positionnement pince-objet :* Ce mouvement, ayant pour but l'approche d'un objet, sera alors réalisé à vitesse réduite. L'objectif n'étant pas le contact entre la pince et l'objet, mais uniquement l'approche de l'un par l'autre, c'est l'instruction de positionnement dans l'espace qui est utilisée. Ainsi, avant toute action sur la pince, l'instruction de saisie fait appel à la commande (REALISER-S <situation-géométrique>) où <situation-géométrique> représente la position de saisie, sa syntaxe est la même que celle donnée en 2.2.2.

– *L'action de saisie* : Elle peut se faire par ouverture de la pince (saisie d'un objet par un trou), ou par fermeture de la pince. Cette dernière possibilité est la valeur par défaut car elle correspond au cas le plus courant.

D'autres paramètres pourraient être ajoutés en fonction du type d'outil de préhension utilisé : mors à écartement variable, mors possédant des capteurs de pression, ...

– *Le nom de l'objet à saisir :* Sa connaissance est nécessaire pour maintenir la liste des objets "déplaçables", en effet tout objet tenu dans la pince du robot est soumis aux mêmes déplacements que ceux du manipulateur. Ceci est nécessaire pour mettre à jour le modèle. Ce nom est connu de façon implicite. En effet, le paramètre <situation-géométrique> désigne l'objet à saisir car il exprime une relation entre un élément du robot et l'objet à saisir.

Exemple : (*axe-pince SUR axe-objet-A*), si *axe-objet-A* appartient au solide *objet-A* alors ce dernier est le nom recherché.

Syntaxe de l'instruction :

Une instruction de saisie du langage à la syntaxe suivante :

(SAISIR <situation-géométrique> { <action> })

<action> ::= OUVRIR | FERMER
La syntaxe du paramètre <situation-géométrique> est celle donnée au paragraphe 2.2.2.

Extension à l'instruction de saisie :

Dans l'instruction de transfert REALISER-S, nous avons étendu la syntaxe des situations géométriques de façon à accepter des instructions de "plus bas niveau". Cette extension est la même pour l'instruction de saisie, elle nécessite l'ajout d'un paramètre dans le cas où le déplacement est spécifié par les coordonnées articulaires ou par une transformation. En effet, ces deux types de situations ne permettent pas de connaître le nom de l'objet qui va être saisi, dans ce cas il doit être donné explicitement dans l'instruction. Nous obtenons alors la syntaxe suivante pour l'instruction de saisie :
(SAISIR <situation> { nom-d'objet } { <action> })
Le paramètre <situation> est défini comme dans le cas d'une instruction REALISER-S (voir paragraphe précédent : quelques extensions à la syntaxe).

- **Le lâcher**

L'opération de lâcher consiste à déposer l'objet une fois l'opération de manipulation réalisée. La situation géométrique à atteindre après le lâcher est précisée afin d'éloigner la pince du robot de l'objet déposé.

Le nom de l'objet lâché n'est pas nécessaire puisqu'il correspond à celui de l'objet saisi précédemment.

L'action de lâcher se fait soit par ouverture des mors (cas le plus courant), soit par fermeture selon le type de saisie réalisée auparavant.

Le déplacement après la pose de l'objet, est exécuté à l'aide de l'instruction REALISER-S, puisqu'il s'agit simplement de positionner la pince du robot dans l'espace.

Comme dans le cas de la saisie, d'autres paramètres pourraient être ajoutés en fonction du type d'outil de préhension utilisé : mors à écartement variable, mors possédant des capteurs de pression, ...

Syntaxe de l'instruction :

La syntaxe d'une instruction de lâcher d'objet est la suivante :

(LACHER <situation-géométrique> { <action> })

<action> ::= OUVRIR | FERMER.
Le paramètre situation géométrique est décrit au paragraphe 2.2.2.

Extension :

Comme pour l'instruction de déplacement, la situation géométrique peut être étendue à une situation générale. La possibilité est donnée de préciser le mouvement à effectuer après lâcher par coordonnées articulaires, par transformation géométrique ou par mise en correspondance de deux repères. La syntaxe de l'instruction est alors : (LACHER <situation> { <action> }), où <situation> est la même que celle de l'instruction REALISER-S décrite au paragraphe 2.3.1.

2.3.3 Les mouvements fins au contact

Un des objectifs essentiel de ce langage est la réalisation de montages et d'assemblages d'objets. Pour cela il faut mettre en contact des pièces et pouvoir maintenir certains contacts lors d'un déplacement.

Ces mouvements sont appelés "mouvements fins au contact" car ils se produisent dans un espace très contraint et mettent en jeu des petits mouvements guidés par des données sensorielles. L'incertitude de position des objets est très importante vis à vis des jeux de montage, aussi ces mouvements utilisent des capteurs afin d'éviter des chocs ou des coincements. Ils ont donc pour objet de réaliser un assemblage malgré les incertitudes portant sur la position et les dimensions des objets manipulés. Pour cela deux types de constructions utilisant des capteurs de force sont employés [Gan 86] :

– *Les mouvements gardés :* Ils sont composés d'une instruction de déplacement et d'une expression de la condition d'arrêt du mouvement.
Exemple : déplacer *robot* selon *Vz* jusqu'à $Fz > 40$; où *Vz* est la direction de l'axe Z du repère robot et *Fz* est la force mesurée selon *Vz*.

– *Les mouvements compliants :* Ils sont composés d'une instruction de déplacement et d'expressions (intervalles de valeurs pour les capteurs de forces) indiquant les contacts à maintenir durant le déplacement. Par exemple, pour suivre une surface avec maintien du contact, on applique la stratégie suivante : tant que le contact existe entre l'objet et la surface (valeur lue sur le capteur de force supérieur à un seuil donné), le déplacement se fait dans la direction indiquée ; dés que le contact est rompu (valeur lue sur le capteur de forces inférieure au seuil précisé), des mouvements correctifs sont exécutés pour retrouver le contact.
Exemple : déplacer *robot* selon *Vz* en maintenant $(10 < Fx < 25)$; où *Fx* est la force mesurée selon l'axe X du repère robot.

La réalisation d'un assemblage met en jeu ces deux types de mouvements, souvent combinés : réalisation d'un contact (mouvement gardé) tout en maintenant un autre (mouvement compliant). L'instruction REALISER-C de notre langage est de ce type.

Remarque : d'autres capteurs pourraient être utilisés tels que les capteurs de vision, mais à l'heure actuelle une telle mise en œuvre serait coûteuse et délicate.

Syntaxe :

La syntaxe d'une instruction de mouvement fin au contact est la suivante :

(REALISER-C <situation-de-contact> { <direction> } { <maintenir-c> }

<maintenir-c> ::= (MAINTENIR <situation-de-contact>$^+$)
<direction> ::= (SUIVANT <d>)
<d> ::= (Vx Vy Vz) vecteur orienté indiquant une direction de translation |
((Px Py Pz) (Vz Vy Vz)) axe de rotation composé d'un point et d'un vecteur.

La syntaxe du paramètre <situation-de-contact> est celle donnée au paragraphe 2.2.2.

Remarques :

* Les déplacements se faisant dans un espace très contraint et sensible aux incertitudes, la vitesse de mouvement est minimale.

* Lorsqu'un paramètre <direction> est précisé, le déplacement se fait uniquement de la façon indiquée. Aucune vérification n'est cependant faite par le système pour s'assurer que la direction conduit bien au contact demandé.

  Quant aucune direction n'est fournie, le système se charge lui même de trouver le déplacement par calculs sur les entités géométriques (Cf. 4.3).

* Le paramètre <maintenir-c> précise la liste des contacts déjà établis par de précédentes instructions REALISER-C. Les contacts ne doivent pas être rompus par la création du nouveau contact. Bien entendu les contacts spécifiés par <maintenir-c> et <situation-de-contact> doivent être compatibles mais aucune vérification n'est faite à ce niveau par l'interpréteur.

* Il existe une instruction permettant de rompre un contact établi précédamment ; sa syntaxe est la suivante :
  (SUPPRIMER-C <situation-de-contact> { <direction> } { <maintenir-c> }).
  La syntaxe de <situation-de-contact>, <direction> et <maintenir-c> est la même que celle donnée dans le paragraphe précédent.

2.3.4 Les mouvements fins correctifs

• **Instruction de correction d'une situation**

En raison de l'incertitude, il n'est pas toujours possible de réaliser une situation géométrique de manière exacte. De plus, une situation géométrique peut ne pas avoir été atteinte à l'issue d'un mouvement du robot, à cause d'un arrêt du mouvement dû à un contact ou à un coincement. La réalisation d'une telle situation peut alors se faire en effectuant un processus itératif, permettant de réduire l'incertitude à chaque itération. Cette incertitude n'est cependant pas explicitement représentée dans les instructions du langage.

L'instruction CORRIGER est utilisée dans le corps de boucle pour décrire les mouvements permettant de converger vers la situation géométrique recherchée. Afin de maintenir la cohérence du modèle, la relation géométrique concernée par la correction est explicitement citée dans les paramètres de l'instruction.

Syntaxe de l'instruction :

(CORRIGER <situation-géométrique> (SUIVANT <d>) (AMPLITUDE <a>))

Le paramètre <situation-géométrique> est identique à celui utilisé dans les autres instructions, sa syntaxe est donnée au paragraphe 2.2.2.

<d> ::= (Vx Vy Vz) vecteur orienté indiquant une direction de translation | ((Px Py Pz) (Vz Vy Vz)) axe de rotation composé d'un point et d'un vecteur.
<a> ::= valeur réelle indiquant le déplacement en millimètres (translation) ou en radians (rotation).

- **Instruction de vérification d'une situation**

Une telle instruction permet de vérifier si une situation géométrique a été réalisée, et donc que chacune des relations géométriques de la situation est établie. Pour cela le système utilise le modèle qui reflète à chaque instant l'état de l'environnement de travail aux diverses incertitudes prés. La vérification se fait sur les valeurs numériques des entités, par exemple, si deux plans doivent être en correspondance alors le système vérifie que leurs normales sont parallèles et qu'un point de l'un appartient à l'autre, si c'est le cas alors la relation est vérifiée. Chaque relation de la situation géométrique est testée, dés qu'une relation n'est pas vérifiée alors la situation géométrique n'est pas réalisée.

Cette instruction peut être utilisée comme test dans une structure itérative ou conditionnelle de mouvements fins correctifs. Un exemple d'une telle structure est donné dans le paragraphe 2.3.4.

Syntaxe de l'instruction :

(VRF-SITUATION <situation-générale>)

<situation-générale> = <situation-géométrique> | <situation-de-contact>
Avec le paramètre <situation-géométrique> identique à celui défini en 2.2.2 et <situation-de-contact> défini en 2.3.3.

• **Clause de GARDE**

Les conditions de garde permettent de stopper le mouvement lorsque au moins l'une des conditions est évaluée à vrai. La présence d'une clause de garde vient du fait que les incertitudes ne permettent pas de garantir que le mouvement demandé se terminera sur l'objectif prévu. Elle nécessite par contre de prévoir auparavant quelles seront les différentes situations d'échec que le robot peut rencontrer lors de l'exécution du mouvement. Dans l'implantation actuelle du système seules des conditions portant sur les contacts sont évaluées. Une telle clause peut être utilisée dans les instructions REALISER-S et REALISER-C.

Syntaxe de la clause :

(GARDE <liste-contacts>)

<liste-contacts> ::= (<relation-géo.-contact> $^{+}$)
<relation-géo.-contact> ::= (<entité-géo-réelle> SUR <entité-géo-réelle>)
<entité-géo-réelle> ::= nom-de-face | nom-d'arête | nom-de-sommet

• **Clause EFFORT**

Pour réaliser certains montages, il est nécessaire de disposer d'une clause de garde avec des conditions de force. Parfois il est indispensable de forcer pour réaliser une insertion, dans ce cas, la clause EFFORT impose une certaine pression. Cette clause peut être utilisée dans une instruction REALISER-C.

Syntaxe de la clause :

(EFFORT <condition>)

<condition> ::= (< cond-i > $^{+}$)
<cond-i> ::= (X > a)
avec X = Fx | Fy | Fz | Mx | My | Mz
où Fx, Fy et Fz sont les forces repectivement mesurées selon les axes X, Y et Z du repère robot ; Mx, My et Mz sont les moments détectés par rapport aux axes X, Y et Z du robot;
et a un réel exprimant une force.

- **Exemple d'utilisation**

```
(realiser-s <insertion> (GARDE <chanfrein>))
(tantque (non (vrf-situation <insertion>)) faire
        (cond ((vrf-situation <chanfrein>)
               (corriger <aligne-axe> (SUIVANT (Fx Fy 0)) (AMPLITUDE I/2))
               (I=I/2)
               (realiser-s <insertion> (GARDE <chanfrein>))))
```

avec :
<insertion> ::= relation à réaliser,
<chanfrein> ::= contact avec le chanfrein,
<aligne-axe> ::= alignement des axes verticaux (celui du goujon et celui du chanfrein).

La direction (Fx Fy 0) est définie à partir des composantes horizontales du vecteur de force.
I représente l'erreur maximum associée à un alignement d'axes (non géré automatiquement par le système a' l'heure actuelle).

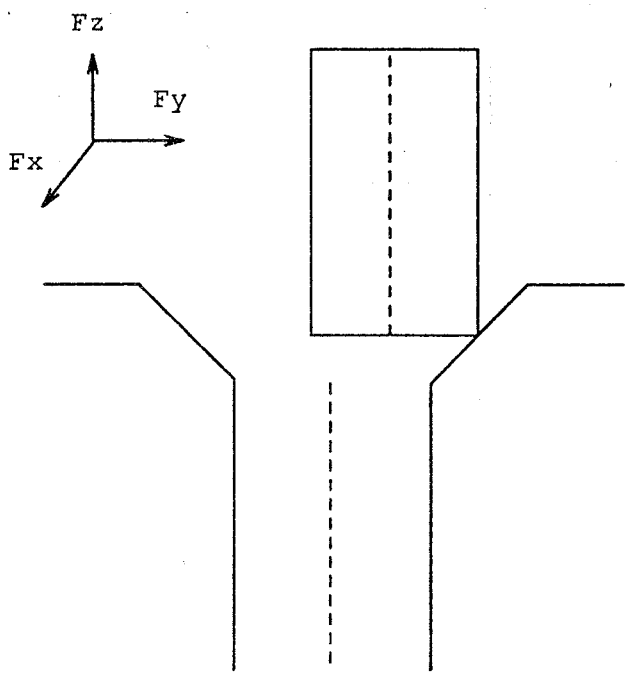


Figure 2.5: Insertion d'un goujon dans un chanfrein

2.4 Les réalisations antérieures dans le cadre de SHARP

Dans le cadre du projet SHARP de programmation automatique des robots, des travaux ont déjà permis un certain nombre de réalisations. Celles-ci vont servir de support, de bases ou d'outils au développement du langage de niveau objet :

* le langage de manipulation LM,
* le système de modélisation géométrique de l'univers et des objets,
* les plans d'action produits automatiquement par le planificateur.

2.4.1 Le système LM

Réalisé au LIFIA dans les années 80, LM [Maz 81,Mir 84,MM 85] est maintenant commercialisé par la société ITMI et il fonctionne sur de nombreux calculateurs et robots.

Le langage LM est un langage de type PASCAL, qui comporte des structures de données et des constructions adaptées à la programmation des robots.

Ce langage dit de niveau "effecteur" permet de travailler en termes des déplacements et des actions de l'outil terminal du robot. Outre les instructions classiques des langages algorithmiques, il comporte :

– des instructions permettant de commander les déplacements du robot,

– la possibilité de définir des entités géométriques telles que des repères cartésiens, des vecteurs, des transformations,

– la possibilité d'accéder par des "variables d'état" à des valeurs fonction de paramètres externes au système de commande (en particulier le capteur de forces).

Position des objets et du robot :

Un repère particulier appelé STATION sert de référentiel commun à l'utilisateur et au système LM. La position de STATION peut varier selon l'implantation, mais elle est fixe pour un site donné.

La position absolue des repères attachés aux objets est définie par rapport à ce référentiel.

La position de l'extrémité terminale du manipulateur (pince) est modélisée par le repère ROBOT. Si l'utilisateur le désire, il peut utiliser momentanément un repère autre que ROBOT pour modéliser une position située à l'extrémité terminale du manipulateur (objet tenu dans la pince, nouvel outil terminal). Il suffit pour cela de lier un repère au repère ROBOT. Les instructions utilisées sont alors les suivantes :

LIER *repère-1* A *repère-2*

ou

LIER *repère-1* A *repère-2* PAR *transformation.*

L'instruction LIER permet de rendre solidaire le solide auquel appartient *repère-1* et celui auquel appartient *repère-2.*

De la même façon on dispose de l'instruction permettant de rompre une telle liaison :

DELIER *repère-1*

ou

DELIER *repère-1* DE *repère-2*

Les déplacements sous LM

Une instruction de déplacement sous LM peut s'exprimer par :

DEPLACER *repère-1* A *repère-2*

ou

DEPLACER *repère-1* DE *transformation*

ou

DEPLACER *repère-1* VIA *repère-2 repère-3 ... repère-n*

Le principe est d'amener un repère sur un autre ou de lui faire subir une transformation géométrique :

* Le repère *repère-1* doit être déplaçable, c'est à dire lié au repère ROBOT ou le repère ROBOT lui même. On écrira par exemple :

  DEPLACER robot DE TRANSLAT(Vz,-10) ;
  DEPLACER robot A *repère-montage* ;

* L'option VIA permet de définir un déplacement passant au voisinage de positions intermédiaires qui sont évaluées avant le déplacement. On écrira par exemple :

  DEPLACER ROBOT A *repère-objet* VIA *repère-1 repère-2.*
  La position d'arrivée est alors spécifiée par *repère-objet*, mais le déplacement comprend trois composantes allant de ROBOT à *repère-1*, de *repère-1* à *repère-2* et de *repère-2* à *repère-objet.*

* Un déplacement peut être exécuté en mode "cartésien" ou en mode "libre" :
  En mode cartésien, la trajectoire suivie par l'origine du repère est rectiligne, et la rotation est effectuée à chaque instant autour d'un axe passant par l'origine de ce repère et parallèle à une direction fixe.
  En mode libre, la trajectoire est telle que les degrés de liberté du manipulateur partent et arrivent en même temps. Sa "forme" dépend alors de la structure cinématique du manipulateur et de la configuration dans laquelle il se trouve au moment de l'exécution du déplacement.

Remarque :

Il existe aussi une instruction permettant d'exprimer un déplacement en coordonnées articulaires :

DEPLACER-AXE num-axe position-axe ;

2.4.2 Les instructions compliantes

Ce sont des instructions intervenant dans la phase finale de manipulation, et notamment dans la partie assemblage. Elles permettent d'adapter les mouvements en fonction de l'environnement du robot en vue de l'objectif à atteindre. Les capteurs de force permettent une perception dynamique de cet environnement.

Un mouvement compliant peut être défini comme un mouvement adapté en fonction de valeurs lues sur un capteur (de forces en l'occurrence) et ce, en vue d'un but précis. Il s'agit d'un mouvement asservi aux données du capteur.

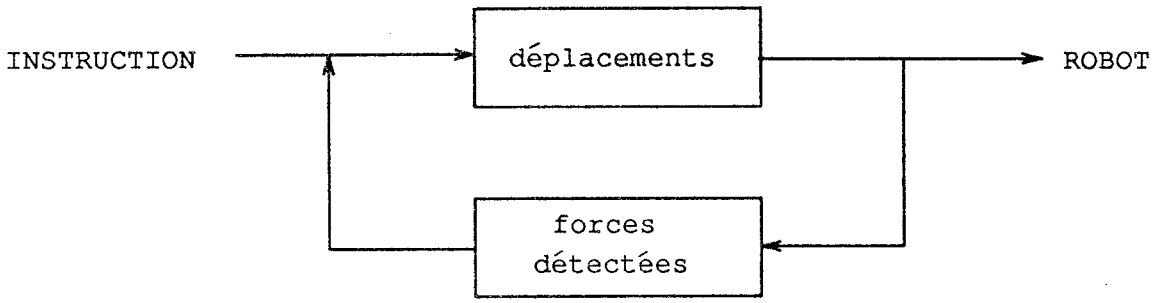


Figure 2.6: asservissement du mouvement au capteur de force

Les travaux de C.Gandon [Gan 86] ont conduit à des fonctions de bases écrites en LM et permettant de réaliser des mouvements compliants ou gardés (Cf. 2.3.3).

Réalisation de contacts ponctuels ou contacts point-plan

Ces contacts induisent uniquement des forces "de translation" (aucun moment n'est détecté) suivant différents axes du repère dont l'origine est au point de contact. Pour réaliser un contact ponctuel, il faut réaliser un mouvement gardé en indiquant la direction de translation et les seuils de contact. En effet la force d'arrêt n'est pas toujours dans la direction de déplacement.

Exemple : Déplacer R selon -Vz jusqu'a Fz > 15.

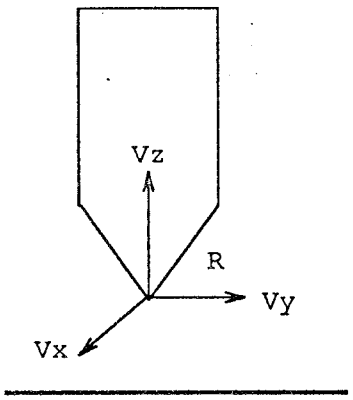
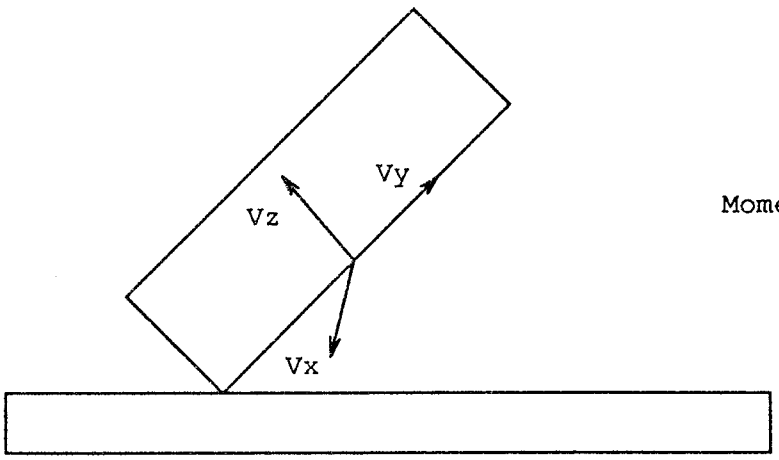


Figure 2.7: Paramètres d'un contact point-plan

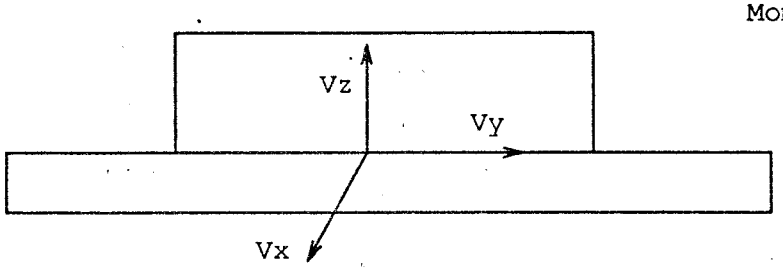
Réalisation de contacts plan-plan

La caractéristique de ce type de contact permet la détection de moments dont l'évaluation peut être utilisée pour adapter l'orientation du repère déplaçable. Le

contact est obtenu lorsque les moments ont pu, par des rotations adéquates, être annulés ou minimisés.



Moments Mx et My
non nuls



Moments Mx et My
nuls

Figure 2.8: moments au cours d'établissement d'un contact plan-plan

Pour cela une fonction de base permet d'engendrer les rotations nécessaires autour d'un axe donné pour annuler le moment correspondant. La réalisation d'un contact plan-plan nécessite en principe trois phases :

1. L'établissement d'un contact ponctuel, par exemple
   déplacer R selon -Vz jusqu'à Fz > 15

2. Rotations autour d'un des deux autres axes pour l'annulation du moment correspondant, par exemple : déplacer R de rotation(Vx,1) jusqu'à abs(Mx) < 2.0

3. Des rotations autour de l'autre axe pour annulation du moment correspondant, par exemple : déplacer R de rotation(Vy,1) jusqu'à abs(My) < 2.0

Les déplacements à compliance

Il s'agit de se déplacer tout en conservant certains contacts (point-plan, plan-plan) établis précédemment. Dans ce cas il faut préciser la direction de déplacement ainsi que les contacts à conserver. Par exemple, déplacer un point sur un plan (dans ce cas on précise $10 < Fz < 25$) ou un objet sur un plan (dans ce cas les moments Mx et My doivent être nuls).

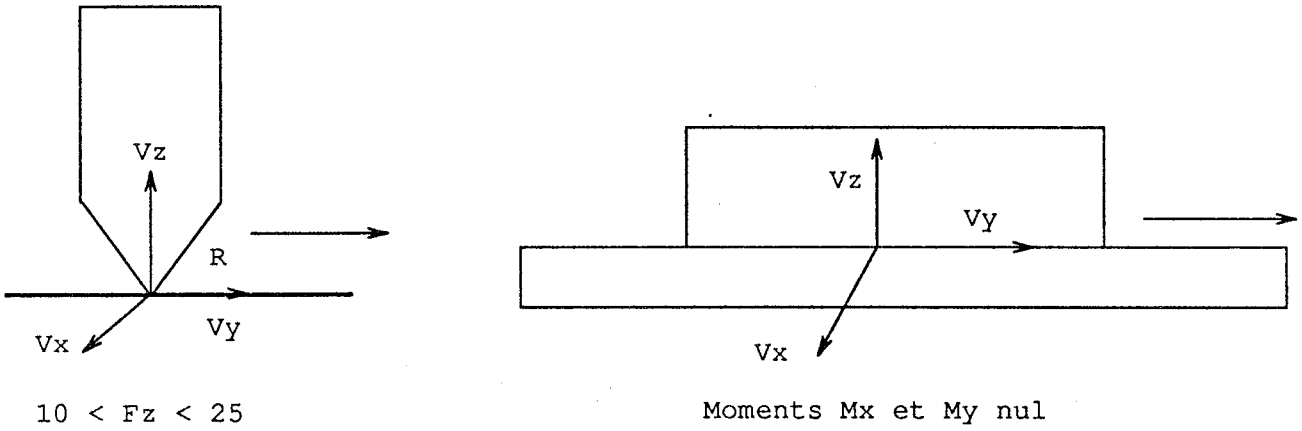


Figure 2.9: Suivi de surface par un point ou par un plan

2.4.3 Le modèle géométrique des objets

La modélisation consiste à représenter, en utilisant un codage, un ensemble de données manipulables par l'intermédiaire d'un calculateur. Dans SHARP, le planificateur, et notamment la partie analysant la phase d'assemblage, utilise le raisonnement géométrique pour produire des plans d'actions en terme de relations entre des entités géométriques (points, arêtes, faces).

2.4.3.1 Les différents éléments du modèle :

– *Représentation des objets de l'espace de travail :* Leur modèle permet d'effectuer facilement des calculs géométriques sur les entités constituant les objets. Chaque entité géométrique (face, arête, sommet) est représentée par son(ses) équation(s). Une structure hiérarchique permet de retrouver les constituants d'un objet à partir du nom de celui-ci et inversement.

– *Position des objets :* Elle est décrite de manière classique à l'aide de repères

cartésiens. Chaque objet comporte un repère de référence propre. La transformation géométrique entre le repère de l'univers (repère absolu) et celui de l'objet détermine la position de ce dernier dans l'environnement de travail. Cette position est représentée par un vecteur de translation et un quaternion de rotation [PW 82]. La position d'un composant est donnée par rapport au repère de l'objet. Les paramètres numériques des entités géométriques sont définis dans le repère de l'objet.

– *Le robot :* Les composants du robot sont représentés comme des objets solides, reliés entre eux par des liaisons articulaires. Le $n^{ième}$ composant de la chaîne articulée qui constitue le robot, est localisé par la donnée de la position du composant n-1 et des valeurs des degrés de liberté existant entre eux. Pour un robot à six degrés de liberté, une configuration du robot sera décrite par six paramètres. Le passage de ce six-uplet à une position cartésienne de l'outil terminal et l'opération inverse constituent la partie cinématique du modèle.

– *Les incertitudes :* Deux sortes d'incertitudes peuvent être identifiées :
· L'incertitude sur la dimension des objets : elle est en général représentée par des intervalles de variation ou une loi de probabilité. Elle n'est pas prise en compte dans notre modèle.
· L'incertitude sur la position des objets : elle est plus importante que la précédente car les ordres de grandeurs ne sont pas comparables. Elle est représentée par une probabilité de distribution. Mais cette information n'est pas réellement utilisée dans la version actuelle du système.

2.4.3.2 Le modèle utilisé dans SHARP

La solution qui a été retenue pour SHARP combine deux types de représentations :

* *"boudary representation"* (BR) qui consiste à modéliser la surface extérieure des objets. Un objet est décrit par ses faces, bornées par des arêtes, elles mêmes délimitées par des sommets. Les entités géométriques sont caractérisées par des paramètres numériques. Ce type de modélisation est bien adapté aux calculs géométriques et à la description des relations entre objets.
* *"constructive solid geometry"* (CSG) : chaque objet est défini à partir de volumes élémentaires (ou composants), combinés entre eux au moyen d'opérateurs ensemblistes. Dans l'implantation actuelle du système, seuls les opérateurs de

collage sont pris en compte.

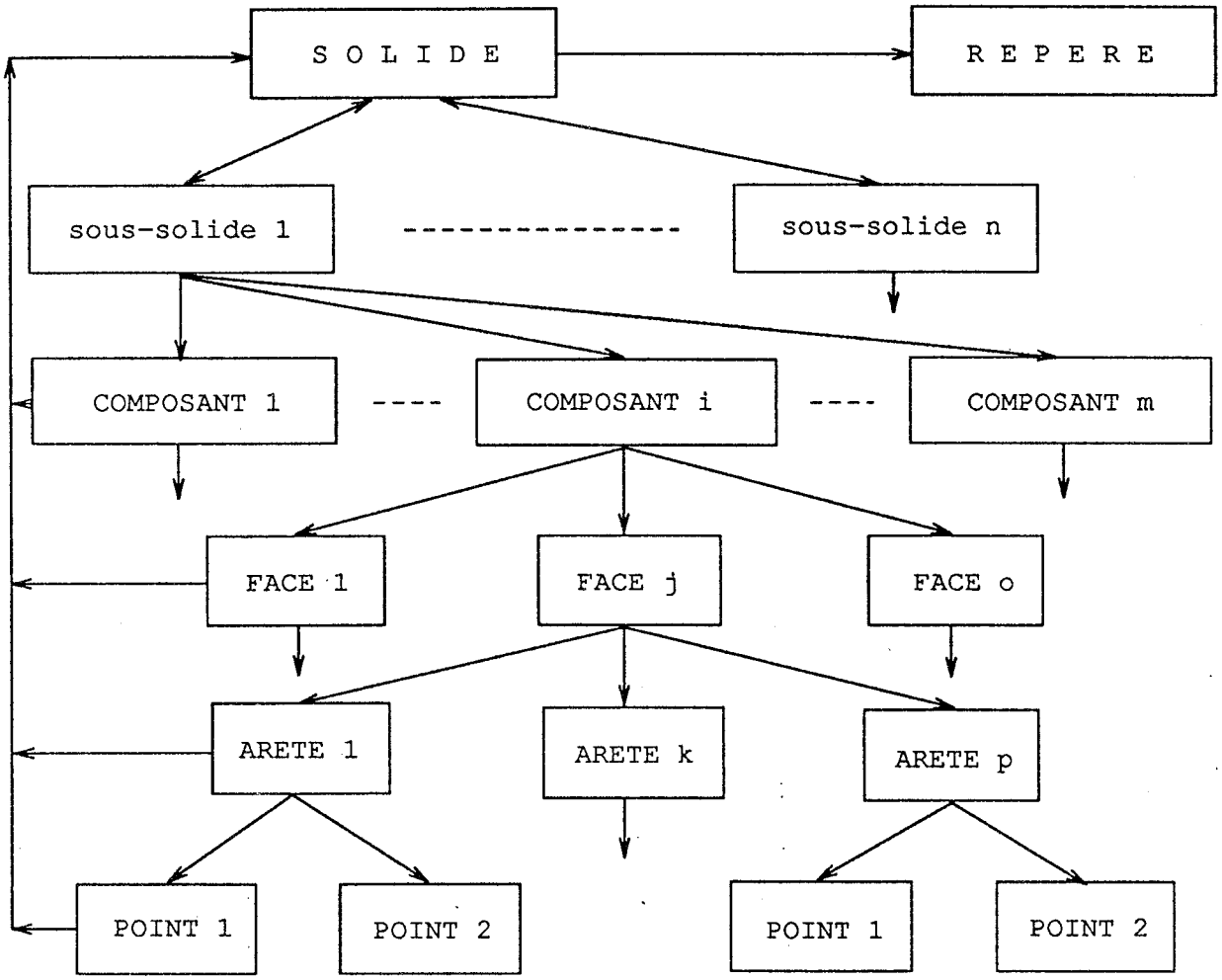


Figure 2.10: Représentation hiérarchique d'un objet

Les informations disponibles dans le modèle pour chaque type d'entités sont les suivantes :

Solide : son type, le nom de son repère de référence, son centre de gravité, la liste de ses composants, une liste permettant l'affichage graphique du solide, la description de son volume englobant.

Composant : son type, sa matière (aluminium, acier, ...), s'il est vide ou plein, la liste de ses faces, son rayon et sa hauteur (dans le cas d'un cylindre), ou ses rayons maxi et mini et sa hauteur (dans le cas d'un cône), sa transformation par rapport au repère de l'objet, le nom de son solide d'appartenance.

Face : son type, la liste de ses facettes, la liste de ses arêtes délimitant son contour, le nom du solide d'appartenance, son équation : un point lui appartenant et une normale extérieure au plan (dans le cas d'un plan), ou un axe de révolution et son rayon (pour un cylindre), ou son sommet et son axe (pour un cône).

Arête : son type, la liste des faces ayant cette arête dans leur frontière, la liste des arêtes partageant un de ses sommets, sa décomposition en arêtes droites dans le cas d'une arête circulaire, ses deux points extrêmes dans le cas d'une droite, son centre, son rayon et un de ses points dans le cas d'un cercle, le nom de son solide d'appartenance.

Point : ses coordonnées exprimées dans le repère de l'objet, le nom de son solide d'appartenance.

Repère : le nom du référentiel absolu, la liste des repères auxquels il est lié, sa position dans l'espace, ses fils.

Un mécanisme de démons [The 88] permet de mettre à jour automatiquement le modèle après chaque opération (simulée ou réelle) du robot :

* la position des objets liés à celui déplacé par le robot ;
* les liens rigides créés ou détruits entre certains objets, par certaines opérations du robot (saisie, lâcher, assemblage) ;
* un certain nombre de propriétés géométriques telles que la liste des faces en contact ou le parallélisme entre les faces ....

2.4.4 Le planificateur

Il détermine les séquences d'actions qui permettront, compte tenu des contraintes liées à la tâche, de réaliser les différents objectifs intermédiaires représentés par les relations d'assemblages. Ces actions incluent des déplacements du robot, des opérations d'outils (saisie, lâcher). La difficulté provient du fait que toutes ces actions doivent être exécutées dans un univers réel complexe. La planification d'un assemblage est alors découpée en trois phases :

* les opérations de saisie,
* les mouvements de transfert,
* les opérations de montage.

2.4.4.1 Planification des opérations de saisie

Quatre aspects différents sont étudiés lors de la planification des opérations de saisie :

– *Le choix de prise* : dans le cas de SHARP ce choix s'appuie sur un modèle géométrique complet de l'univers [LP 83,Lau 83]. La méthode est fondée sur l'analyse morphologique de l'objet à saisir.

– *La stabilité et l'équilibre d'une prise* : elle ne peut pas être directement résolue par des méthodes mathématiques rigoureuses à cause de la présence d'incertitudes (incertitudes de position sur l'objet et l'outil de préhension, mauvaise connaissance des forces appliquées et des phénomènes de friction). Dans SHARP c'est une méthode heuristique qui tente de résoudre ce problème.

– *L'étude d'accessibilité*: permet de déterminer si une prise est réalisable compte-tenu de l'encombrement spatial environnant. Le raisonnement permettant de trouver une trajectoire pour l'outil de préhension, une fois contraint l'orientation et la direction d'approche, s'appuie sur une modélisation de l'espace des configurations de l'outil [LP 83].

– *La compatibilité contextuelle* : provient du fait que l'opération de saisie est exécutée dans un but précis (assemblage d'objets) et qu'elle doit donc permettre de réaliser correctement cette opération finale.

En sortie nous obtenons une instruction du type :
(REALISER-S '((repère-pince SUR repère-saisie)))

avec repère-pince, un repère cartésien placé (par exemple) au centre de la pince du robot, et repère-saisie, un repère positionné sur l'objet à saisir.

2.4.4.2 Planification des trajectoires de transfert

Il s'agit de calculer une trajectoire permettant au robot de transporter des objets dans un espace parsemé d'obstacles. Ces trajectoires doivent satisfaire des contraintes telles que l'absence de collision, tout en présentant un critère d'optimalité satisfaisant.

La méthode repose sur une étude de l'espace des configurations du robot [Pas 89].

– *Espace des configurations :* c'est le graphe de l'espace libre. La position exacte de chacun des composants de la chaîne articulée est complètement déterminée par la donnée des valeurs des p degrés de liberté (pour un robot à p degrés de liberté). L'ensemble des p-uplets que peut vérifier la structure constitue l'espace des configurations. Cet espace est limité par les butées mécaniques du bras (chaîne cinématique), les obstacles de l'espace de travail du robot (taux d'encombrement) et les objets à manipuler. L'établissement de ce graphe se fait à l'aide d'une discrétisation de l'espace articulaire et de grossissements des obstacles.

– *Recherche d'une trajectoire :* Elle consiste au parcours du graphe de l'espace libre dont les nœuds représentent des cellules de l'espace libre, et un arc indique un chemin libre entre deux cellules. Si le pas de discrétisation est trop important, le modèle de l'espace libre est sur-contraint et on risque de ne pas trouver de solution ; dans le cas contraire le temps de calcul est élevé.

– *Rotations du poignet :* La méthode travaille sur les trois premiers degrés de liberté du robot pour le positionnement dans l'espace. Les trois autres degrés (rotations du poignet) soient traités par des méthodes locales. Les heuristiques employées imposent que les mouvements au voisinage de la position de départ et d'arrivée sont exécutés à orientation constante, et que les réorientations éventuelles sont réparties le long de la trajectoire calculée antérieurement.

En sortie nous obtenons une instruction du type :
(REALISER-S situation (VIA chemin))
où "situation" est le paramètre défini en 2.3.1 et où le paramètre "chemin" ne comporte que des positions intermédiaires données en coordonnées articulaires.

2.4.4.3 Planification des opérations de montage

L'objectif visé est de générer des mouvements aptes à réaliser un assemblage donné malgré les contraintes d'incertitudes engendrées par la tâche. Les stratégies implantées combinent des opérations sensorielles avec des petits mouvements du robot.

La démarche consiste à rechercher une séquence ordonnée de relations de contact permettant de guider le robot vers l'objectif à atteindre. Chaque relation réalisée permet de s'approcher du but en réduisant certaines composantes d'incertitude, elle met en jeu des mouvements destinés à rechercher un contact et/ou à en maintenir d'autres créés par des opérations antérieures.

Méthode : Le principe repose sur l'hypothèse de réversibilité selon laquelle une trajectoire de montage peut être obtenue en inversant les mouvements effectués pour le démontage. Le système travaille à partir d'un modèle de pièces assemblées qu'il étudie afin de produire un graphe de démontage, celui-ci sera ensuite parcouru pour obtenir le plan d'assemblage.

– *Graphe de démontage :* Les nœuds contiennent les situations géométriques ou de contacts rencontrées lors du démontage fictif ; les arcs représentent les mouvements qu'il faut effectuer pour passer d'une situation géométrique à une autre.

– *Construction du graphe :* Elle consiste à conserver une trace des différentes opérations fictives de démontage envisagées. Le démontage est réalisé en effectuant un raisonnement qui consiste à décontraindre progressivement l'objet assemblé, ce raisonnement se fait à l'aide de "règles" permettant de passer d'une situation géométrique à une autre.

– *Production du plan de montage :* Le graphe de démontage est parcouru systématiquement et, chaque chemin conduisant d'une position libre insensible aux incertitudes à la situation finale recherchée, est évalué heuristiquement (attribution d'une note indiquant la qualité de la solution correspondante). Ces notes tiennent compte de la stabilité des situations rencontrées et de la nature des mouvements effectués. Une fois le chemin optimal trouvé, le plan de montage est produit. Il comporte alors autant d'instructions qu'il y a de situations intermédiaires le long du chemin retenu. Chaque instruction traduit de manière géométrique la transition à réaliser.

Un exemple de plan produit pour un assemblage est donné en annexe A.

Chapitre 3

ETUDE ET CHOIX DES SOLUTIONS

Deux problèmes principaux se posent pour la réalisation de ce projet :

* *Le choix d'une base de données :*
  L'interpréteur de langage géométrique devant être utilisable à la fois sous SHARP et de façon autonome, il convient de définir quelle structure de données va être utilisée et ce qu'elle doit contenir.

* *L'interprétation des situations géométriques :*
  Trois couches logicielles sont à développer pour passer de la description géométrique de la tâche à la commande du robot :
  – Au plus bas niveau, la commande du robot se fait à l'aide d'une armoire de commandes agissant sur les couples moteurs au niveau de chaque axe. Cette partie est spécifique au robot considéré.
  – LM est une couche au dessus du niveau précédent, ce langage permet de programmer les déplacements à faire exécuter au robot en termes de repères cartésiens et de transformations géométriques (Cf. 2.4.1).
  – LM-OBJ (Langage de Manipulation de niveau OBJet) traduit les instructions de haut niveau en quelques commandes exprimées en LM. Ce dernier point, objet des paragraphes 3.2 à 3.7, est essentiel dans la réalisation de LM-OBJ. Nous allons tout d'abord exposer le principe général de l'interprétation de situations géométriques. Les paragraphes suivants présentent différentes méthodes envisagées pour cette interprétation. La dernière partie de ce chapitre expose la solution retenue pour résoudre ce problème.

3.1 Modèles géométriques nécessaires

3.1.1 Le problème

Comme cela a été exposé au paragraphe 2.2, la programmation des robots en terme des effets sur les objets manipulés nécessite l'utilisation de bases de données d'origine CAO décrivant l'espace de travail (objets, robots, positions, ...).

Le problème est donc de choisir une structure de données adaptée aux traitements envisagés (mise en relation d'entités géométriques : arêtes, faces, sommets). D'autre part, elle doit permettre une utilisation de LM-OBJ soit sous l'environnement SHARP, soit de façon autonome en tant que langage de programmation des robots. Ces deux façons de travailler ne doivent pas entraîner la coexistence de deux modules d'interprétation différents et, quelque soit l'environnement de travail, l'accès à une seule et même base de données est impératif ; ce qui permet de conserver un source identique quant à l'accès aux informations et à leur traitement.

3.1.2 Les solutions envisagées

Deux solutions sont à priori possibles : garder la base de données de SHARP décrite précédemment, ou en construire une nouvelle.

- **Conserver la base de données de SHARP**

La base des modèles géométriques de SHARP est détaillée en 2.4.3, elle contient toutes les informations nécessaires à l'interprétation des situations géométriques :

* description des objets par le détail de leur composition (faces, arêtes, sommets),
* situation des objets dans l'espace (repères),
* valeurs numériques carcatérisant les entités géométriques (coordonnées de points, équation de droites ou de plans, transformations géométriques).

En cas d'utilisation hors de l'environnement SHARP, l'interpréteur doit disposer de la base de données et des fonctions d'accès associées, ce qui nécessite d'isoler

cela dans le source de SHARP afin de pouvoir utiliser cette partie de façon "autonome". Beaucoup d'informations contenues dans le modèle sont alors inutiles, car ces dernières ont été introduites pour les besoins des modules de planification.

- **Créer une nouvelle structure de données**

Celle-ci contient des informations théoriquement identiques à la précédente : position des objets, valeurs des entités géométriques (points, droites, plans). Par contre, la structure des objets est moins complète du fait que l'on peut supprimer de nombreux chaînages ; ceux qui permettent de parcourir la structure en "remontant" (par exemple, appartenance de telle arête à telles faces). Dans la nouvelle structure nous donnons la possibilité de ne décrire que les entités utiles, et pas forcément la structure complète d'un objet.

La structure hiérarchique de la description géométrique des objets est évidemment conservée dans le modèle : solides, composants, faces, arêtes, sommets et repères. Mais seules les informations nécessaires sur les entités sont explicitement représentées, à savoir :

solide : son type, le nom de son repère de référence, éventuellement ses composants et son solide d'appartenance.

composant : son type, la liste de ses faces, éventuellement le nom de son solide d'appartenance.

face : sa nature (dans notre cas il s'agit uniquement d'une face plane), un point lui appartenant, une normale unitaire, éventuellement le nom du solide auquel il appartient, éventuellement la liste de ses arêtes.

arête : le nom de ses points extrêmes, les coordonnées d'un vecteur unitaire directeur, la nature de l'arête (dans notre cas ce peut être uniquement une droite), éventuellement le nom du solide auquel il appartient.

point : ses coordonnées et éventuellement le nom du solide auquel il appartient.

repère : sa position exprimée dans le repère absolu.

Dans le cas de l'utilisation d'une telle structure, dite "structure minimale", le système doit préalablement initialiser la nouvelle structure par recopie des informations nécessaires contenues dans la base de modèles géométriques de SHARP. Après exécution de la tâche, ou de la sous-tâche, la cohérence du modèle d'origine

est maintenue en modifiant la position des objets déplacés au cours des différentes manipulations. Il en est de même pour les liens rigides qui ont pu être créés ou détruits par des opérations de saisie ou de lâcher.

3.1.3 La solution retenue

Nous avons retenu la deuxième solution qui consite à créer une nouvelle base ne contenant que les informations nécessaires. Les principales raisons de ce choix sont les suivantes :

* *La base est restreinte* par rapport à celle de SHARP : elle ne contient que les informations nécessaires, ce qui évite de manipuler de nombreuses données et fonctions inutiles pour l'exécution de la tâche. Ceci permet en particulier de réduire la taille de l'interpréteur en supprimant le mécanisme de démons, utilisé pour le maintien de la cohérence de prédicats, ainsi que les chaînages "arrières".

* L'utilisateur n'aura *qu'un minimum de données à saisir* pour décrire son espace de travail (position des objets, description des seules entités géométriques qu'il veut utiliser) en cas d'utilisation de LM-OBJ en autonome.

* Mise à part la phase d'initialisation différente selon le mode d'utilisation de LM-OBJ et la phase finale (mise à jour de la base des modèles géométriques de SHARP) réservée en fonctionnement sous SHARP, *le code écrit est le même dans les deux cas.*

* Cette solution *privilégie l'utilisation en mode autonome* ce qui permet de voir en LM-OBJ un langage de programmation de robots à part entière.

* En utilisation dans l'environnement SHARP, la base de modèles géométriques originelle n'est modifiée par les manipulations effectuées que si cela s'avère nécessaire pour une tâche ultérieure confiée à SHARP.

Un inconvénient est cependant à noter dans le cas d'un appel à l'interpréteur du langage par SHARP :

* duplication des informations utiles dans la structure minimale (phase d'initialisation).

Toutefois ceci ne conduit pas à une dégradation d'utilisation de SHARP. En effet, SHARP est un logiciel complexe et de taille conséquente, aussi l'obligation d'appeler une phase d'initialisation et une phase de remise à jour, pour une manipulation complète, ne modifie pas ses performances. D'autre part, le recours à l'interpréteur de LM-OBJ n'est pas systématique, sous l'environnement SHARP. Un module de simulation graphique permet d'exécuter fictivement le plan d'action calculé, et ce jusqu'à ce qu'il soit correct. Ce n'est qu'une fois cette mise au point terminée que la manipulation réelle peut être effectuée.

Afin de mettre en œuvre cette solution, nous avons développé deux types de fonctions :

- **Création du modèle :**

En cas de fonctionnement sous l'environnement de programmation automatique, l'initialisation du modèle est faite à partir des informations du modèle de SHARP. La possibilité est alors laissée à l'utilisateur de préciser le nom des objets et des éléments du robot qu'il veut dupliquer dans la structure minimale, et ce afin d'éviter de copier des données présentes dans le modèle d'origine qui sont inutiles pour l'exécution du plan d'actions.

En utilisation autonome, l'utilisateur dispose de fonctions permettant de créer et initialiser les objets ou les entités géométriques qu'il souhaite manipuler. Cela peut se faire par dialogue en début de session sous LM-OBJ, ou à n'importe quel moment du travail par l'appel aux fonctions décrites ci-dessus.

- **Retour des informations vers le modèle d'origine :**

En fonctionnement sous l'environnement de programmation automatique, la fin de l'exécution conduit à mettre à jour les positions des objets dans la base de SHARP. Dans cette optique, LM-OBJ gére un historique des déplacements effectués sur les objets, et des opérations de saisie et de lâcher qui modifient les liens rigides entre objets.

3.2 Principe de l'interprétation des situations géométriques

Interpréter une situation géométrique présente dans la spécification d'une instruction conduit à extraire et calculer à partir du paramètre <situation-géométrique> le repère à déplacer ainsi que la transformation géométrique à lui appliquer. Il faut alors résoudre une combinaison de relations géométriques qui, vu la nature des entités sur lesquelles le système travaille, sont au nombre de neuf :

point-point

point-droite

point-plan

droite-point

droite-droite

droite-plan

plan-point

plan-droite

plan-plan.

Nous allons étudier plusieurs alternatives permettant de résoudre une situation géométrique à réaliser. Pour chacune d'elles, nous exposerons la représentation des entités utilisées, la résolution proprement dite et les inconvénients de la méthode. Nous allons étudier successivement les méthodes suivantes : la "résolution générale", la "pré-résolution" au cas par cas, l'interpréteur géométrique de LM-GEO, la résolution dans le cas de relations isolées, et la "résolution séquentielle".

Remarque :

Les situations qui sont des extensions au langage permettant d'utiliser des instructions de plus bas niveau (Cf. paragraphe 2.2), ne posent aucun problème quant à la traduction en instructions LM car elles sont du même niveau :

– (REALISER-S (*repère-1* SUR *repère-2*))

devient

LIER *repère-1* A *repère-2* ;

DEPLACER *repère-1* A *repère-2* ;

– (REALISER-S (*repère transformation*))

devient

LIER *repère* A ROBOT ;

DEPLACER *repère* DE *transformation* ;

– (REALISER-S (θ_1 θ_2 θ_3 θ_4 θ_5 θ_6))

devient

DEPLACER-AXE 1 A θ_1 ;

DEPLACER-AXE 2 A θ_2 ;

DEPLACER-AXE 3 A θ_3 ;

DEPLACER-AXE 4 A θ_4 ;

DEPLACER-AXE 5 A θ_5 ;

DEPLACER-AXE 6 A θ_6 ;

Notations utilisées tout au long de ce document :

$\times$ = produit scalaire de deux vecteurs,
$\wedge$ = produit vectoriel de deux vecteurs,
$*$ = produit d'une matrice par un vecteur,
O = vecteur nul,
$\|X\|$ = norme du vecteur X.

3.3 Méthode de résolution générale

Après avoir défini la façon de représenter les concepts manipulés, le principe est de traduire chaque relation géométrique de la situation en équations analytiques. Nous obtenons ainsi un système d'équations à résoudre. L'objet de ce paragraphe est de savoir si la représentation des entités et si le système de résolution proposé est bien adapté pour l'interprétation des relations géométriques.

3.3.1 Représentation des entités et des transformations géométriques

La représentation des entités est issue de la base de données présentée en 3.1.

POINT P = (x y z)

Un point est représenté par trois coordonnées exprimées dans le repère propre de l'objet, ou dans celui de l'univers si c'est un point "virtuel" et non pas un sommet.

DROITE D = (PD , V) avec PD = (x_d y_d z_d) et V = (vx vy vz)

Elle est décrite par un point PD lui appartenant et un vecteur directeur V. Tout point X appartenant à cette droite vérifie l'équation vectorielle [Vyg 75] :

$(X - PD) \wedge V = O.$

PLAN F = (PF , N) avec PF = (x_f y_f z_f) et N = (nx ny nz)

Un plan est défini par un point PF lui appartenant et par une de ses normales N. Tout point X appartenant au plan vérifie l'équation vectorielle :

$N \times (X - PF) = 0.$

TRANSFORMATION matrice de coordonnées homogènes composée d'une translation pure (T) et d'une rotation pure (R).

$$Tr = \begin{pmatrix} r_{11} & r_{12} & r_{13} & tx \\ r_{21} & r_{22} & r_{23} & ty \\ r_{31} & r_{32} & r_{33} & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_1 & R_2 & R_3 & T \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Par définition cette matrice vérifie les propriétés suivantes :

$$\begin{cases} \|R_1\| = \|R_2\| = \|R_3\| = 1 \\ R_1.R_2 = R_1.R_3 = R_2.R_3 = 0 \end{cases}$$

3.3.2 Expression analytique des relations

Chaque type de relation géométrique est décrite par une ou plusieurs équations vectorielles dans lesquelles l'inconnue est la transformation Tr. Utilisant les notations du paragraphe précédent, notamment pour Tr, les équations sont ensuite écrites sous forme paramétrique. Les inconnues sont alors les réels rx_1, rx_2, rx_3, ry_1, ry_2, ry_3, rz_1, rz_2, rz_3, tx, ty, tz.

• **Relation "POINT-POINT" : (P1 SUR P2)**

avec P1 = (x_1 y_1 z_1) et P2 = (x_2 y_2 z_2).

La transformation Tr permettant d'amener le point P1 sur le point P2 peut s'exprimer comme suit :

$Tr * P1 = P2$

Cette équation vectorielle indique que le point déplacé ($Tr * P1$) doit coïncider avec le point P2. Les équations paramétriques sont alors les suivantes :

$$(3.1)\quad \begin{cases} x_1 \cdot rx_1 + y_1 \cdot ry_1 + z_1 \cdot rz_1 + tx = x_2 \\ x_1 \cdot rx_2 + y_1 \cdot ry_2 + z_1 \cdot rz_2 + ty = y_2 \\ x_1 \cdot rx_3 + y_1 \cdot ry_3 + z_1 \cdot rz_3 + tz = z_2 \end{cases}$$

• **Relation "POINT-DROITE" : (P1 SUR D2)**

avec P1 = (x_1 y_1 z_1)
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

La transformation Tr permettant d'amener le point P1 sur la droite D2 peut s'exprimer par l'équation vectorielle :

$(Tr * P1 - P2) \wedge V2 = O$

qui traduit que le point déplacé ($Tr * P1$) doit appartenir à la droite D2 et donc comme tout point X de cette droite, vérifier l'équation $(X - P2) \wedge V2 = O$. Les équations paramétriques correspondantes sont alors les suivantes :

$$
(3.2)\quad \begin{cases}
vz_2 \cdot x_1 \cdot rx_2 + vz_2 \cdot y_1 \cdot ry_2 + vz_2 \cdot z_1 \cdot rz_2 + vz_2 \cdot ty - vy_2 \cdot x_1 \cdot rx_3 \\
-vy_2 \cdot y_1 \cdot ry_3 - vy_2 \cdot z_1 \cdot rz_3 - vy_2 \cdot tz \;=\; vy_2 \cdot z_2 - vz_2 \cdot y_2 \\
-vz_2 \cdot x_1 \cdot rx_1 - vz_2 \cdot y_1 \cdot ry_1 - vz_2 \cdot z_1 \cdot rz_1 - vz_2 \cdot tx + vx_2 \cdot x_1 \cdot rx_3 \\
+vx_2 \cdot y_1 \cdot ry_3 + vx_2 \cdot z_1 \cdot rz_3 + vx_2 \cdot tz \;=\; vz_2 \cdot x_2 - vx_2 \cdot z_2 \\
vy_2 \cdot x_1 \cdot rx_1 + vy_2 \cdot y_1 \cdot ry_1 + vy_2 \cdot z_1 \cdot rz_1 + vy_2 \cdot tx - vx_2 \cdot x_1 \cdot rx_2 \\
-vx_2 \cdot y_1 \cdot ry_2 - vx_2 \cdot z_1 \cdot rz_2 - vx_2 \cdot ty \;=\; vx_2 \cdot y_2 - vy_2 \cdot x_2
\end{cases}
$$

- **Relation "POINT-PLAN' : (P1 SUR F2)**

avec P1 = ($x_1\ y_1\ z_1$)
et F2 = (P2 , N2) où P2 = ($x_2\ y_2\ z_2$), N2 = ($nx_2\ ny_2\ nz_2$).

La transformation Tr permettant d'amener le point P1 sur le plan F2 peut s'exprimer comme suit :

$$N2 \times (Tr * P1 - P2) = 0$$

Cette équation traduit le fait que le point déplacé ($Tr * P1$) doit appartenir au plan F2. Comme tout point X de ce plan, il vérifie $N2 \times (X - P2) = 0$. L'équation paramétrique correspondante est alors la suivante :

$$
(3.3)\quad \begin{array}{l}
nx_2 \cdot x_1 \cdot rx_1 + nx_2 \cdot y_1 \cdot ry_1 + nx_2 \cdot z_1 \cdot rz_1 + nx_2 \cdot tx + ny_2 \cdot x_1 \cdot rx_2 \\
+ny_2 \cdot y_1 \cdot ry_2 + ny_2 \cdot z_1 \cdot rz_2 + ny_2 \cdot ty + nz_2 \cdot x_1 \cdot rx_3 + nz_2 \cdot y_1 \cdot ry_3 + \\
nz_2 \cdot z_1 \cdot rz_3 + nz_2 \cdot tz \;=\; nx_2 \cdot x_2 + ny_2 \cdot y_2 + nz_2 \cdot z_2
\end{array}
$$

- **Relation "DROITE-POINT" : (D1 SUR P2)**

avec P2 = ($x_2\ y_2\ z_2$),
et D1 = (P1 , V1) où P1 = ($x_1\ y_1\ z_1$), V1 = ($vx_1\ vy_1\ vz_1$).

La transformation Tr permettant d'amener la droite D1 sur le point P2 peut s'exprimer comme suit :

$$(P2 - Tr \times P1) \wedge Tr * V1 = O$$

Cette équation vectorielle traduit que le point P2 doit appartenir à la droite déplacée ($Tr*P1, Tr*V1$). Comme tout point X de cette "nouvelle" droite il doit vérifier l'équation $(X - Tr*P1) \wedge Tr*V1 = O$. Les équations paramétriques correspondantes sont les suivantes :

$$
(3.4)\quad \left\{
\begin{array}{l}
vx_1 \cdot y_2 \cdot rx_3 + vy_1 \cdot y_2 \cdot ry_3 + vz_1 \cdot y_2 \cdot rz_3 + y_2 \cdot tz \\
-vx_1 \cdot z_2 \cdot rx_2 + vy_1 \cdot z_2 \cdot ry_2 + vz_1 \cdot z_2 \cdot rz_2 + z_2 \cdot ty \\
+(vy_1 \cdot x_1 - vx_1 \cdot y_1) \cdot rx_3 \cdot ry_2 + (vz_1 \cdot x_1 - vx_1 \cdot z_1) \cdot rx_3 \cdot rz_2 \\
+(vx_1 \cdot y_1 - vy_1 \cdot x_1) \cdot ry_3 \cdot rx_2 + (vz_1 \cdot y_1 - vy_1 \cdot z_1) \cdot ry_3 \cdot rz_2 \\
+(vx_1 \cdot z_1 - vz_1 \cdot x_1) \cdot rz_3 \cdot rx_2 + (vy_1 \cdot z_1 - vz_1 \cdot y_1) \cdot rz_3 \cdot ry_2 \\
+(x_1 - vx_1) \cdot rx_3 \cdot ty + (y_1 - vy_1) \cdot ry_3 \cdot ty + (z_1 - vz_1) \cdot rz_3 \cdot ty \\
+(vx_1 - x_1) \cdot rx_2 \cdot tz + (vy_1 - y_1) \cdot ry_2 \cdot tz + (vz_1 - z_1) \cdot rz_2 \cdot tz \\
\quad = \quad 0 \\
\\
-vx_1 \cdot x_2 \cdot rx_3 - vy_1 \cdot x_2 \cdot ry_3 - vz_1 \cdot x_2 \cdot rz_3 - x_2 \cdot tz \\
+vx_1 \cdot z_2 \cdot rx_1 + vy_1 \cdot z_2 \cdot ry_1 + vz_1 \cdot z_2 \cdot rz_1 + z_2 \cdot tx \\
+(vx_1 \cdot y_1 - vy_1 \cdot x_1) \cdot rx_3 \cdot ry_1 + (vx_1 \cdot z_1 - vz_1 \cdot x_1) \cdot rx_3 \cdot rz_1 \\
+(vy_1 \cdot x_1 - vx_1 \cdot y_1) \cdot ry_3 \cdot rx_1 + (vy_1 \cdot z_1 - vz_1 \cdot x_1) \cdot ry_3 \cdot rz_1 \\
+(vz_1 \cdot x_1 - vx_1 \cdot z_1) \cdot rz_3 \cdot rx_1 + (vz_1 \cdot y_1 - vy_1 \cdot z_1) \cdot rz_3 \cdot ry_1 \\
+(vx_1 - x_1) \cdot rx_3 \cdot tx + (vy_1 - y_1) \cdot ry_3 \cdot tx + (vz_1 - z_1) \cdot rz_3 \cdot tx \\
+(x_1 - vx_1) \cdot rx_1 \cdot tz + (y_1 - vy_1) \cdot ry_1 \cdot tz + (z_1 - vz_1) \cdot rz_1 \cdot tz \\
\quad = \quad 0 \\
\\
vx_1 \cdot x_2 \cdot rx_2 + vy_1 \cdot x_2 \cdot ry_2 + vz_1 \cdot x_2 \cdot rz_2 + x_2 \cdot ty \\
-vx_1 \cdot y_2 \cdot rx_1 + vy_1 \cdot y_2 \cdot ry_1 + vz_1 \cdot y_2 \cdot rz_1 + y_2 \cdot tx \\
+(vx_1 \cdot y_1 - vy_1 \cdot x_1) \cdot rx_1 \cdot ry_2 + (vy_1 \cdot x_1 - vx_1 \cdot y_1) \cdot ry_1 \cdot rx_2 \\
+(vz_1 \cdot x_1 - vx_1 \cdot z_1) \cdot rz_1 \cdot rx_2 + (vx_1 \cdot z_1 - vz_1 \cdot x_1) \cdot rx_1 \cdot rz_2 \\
+(vy_1 \cdot z_1 - vz_1 \cdot y_1) \cdot rz_2 \cdot ry_1 + (vz_1 \cdot y_1 - vy_1 \cdot z_1) \cdot ry_2 \cdot rz_1 \\
+(x_1 - vx_1) \cdot rx_2 \cdot tx + (y_1 - vy_1) \cdot ry_2 \cdot tx + (z_1 - vz_1) \cdot rz_2 \cdot tx \\
+(vx_1 - x_1) \cdot rx_1 \cdot ty + (vy_1 - y_1) \cdot ry_1 \cdot ty + (vz_1 - z_1) \cdot rz_1 \cdot ty \\
\quad = \quad 0
\end{array}
\right.
$$

- **Relation "DROITE-DROITE" : (D1 SUR D2)**

avec D1 = (P1 , V1) où P1 = (x_1 y_1 z_1), V1 = (vx_1 vy_1 vz_1),
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

Afin que la relation soit vérifiée il faut qu'après le déplacement :

1. Les deux droites soient parallèles, donc leurs vecteurs directeurs doivent être colinéaires (ce qui s'exprime par leur produit vectoriel nul).
2. Un point de la droite déplacée ($Tr*P1$) appartient à l'autre droite, et comme tout point X de celle-ci il doit vérifier $(X - P2) \wedge V2 = O$.

La droite D'1, qui est la droite D1 transformée, est décrite par $(Tr*P1, Tr*V1)$. Les équations traduisant les exigences 1 et 2 sont donc les suivantes :

$$\begin{cases} (Tr*V1) \wedge V2 = O \\ (Tr*P1 - P2) \wedge V2 = O \end{cases}$$

Les équations paramétriques correspondantes sont alors les suivantes :

$$(3.5)\quad \begin{cases} vz_2 \cdot vx_1 \cdot rx_2 + vz_2 \cdot vy_1 \cdot ry_2 + vz_2 \cdot vz_1 \cdot rz_2 + vz_2 \cdot ty \\ -vy_2 \cdot vx_1 \cdot rx_3 - vy_2 \cdot vy_1 \cdot ry_3 - vy_2 \cdot vz_1 \cdot rz_3 - vy_2 \cdot tz \\ \quad = \quad 0 \\ \\ -vz_2 \cdot vx_1 \cdot rx_1 - vz_2 \cdot vy_1 \cdot ry_1 - vz_2 \cdot vz_1 \cdot rz_1 - vz_2 \cdot tx \\ +vx_2 \cdot vx_1 \cdot rx_3 + vx_2 \cdot vy_1 \cdot ry_3 + vx_2 \cdot vz_1 \cdot rz_3 + vx_2 \cdot tz \\ \quad = \quad 0 \\ \\ vy_2 \cdot vx_1 \cdot rx_1 + vy_2 \cdot vy_1 \cdot ry_1 + vy_2 \cdot vz_1 \cdot rz_1 + vy_2 \cdot tx \\ -vx_2 \cdot vx_1 \cdot rx_2 - vx_2 \cdot vy_1 \cdot ry_2 - vx_2 \cdot vz_1 \cdot rz_2 - vx_2 \cdot ty \\ \quad = \quad 0 \\ \\ vz_2 \cdot x_1 \cdot rx_2 + vz_2 \cdot y_1 \cdot ry_2 + vz_2 \cdot z_1 \cdot rz_2 + vz_2 \cdot ty - vy_2 \cdot x_1 \cdot rx_3 \\ -vy_2 \cdot y_1 \cdot ry_3 - vy_2 \cdot z_1 \cdot rz_3 - vy_2 \cdot tz \quad = \quad vy_2 \cdot z_2 - vz_2 \cdot y_2 \\ \\ -vz_2 \cdot x_1 \cdot rx_1 - vz_2 \cdot y_1 \cdot ry_1 - vz_2 \cdot z_1 \cdot rz_1 - vz_2 \cdot tx + vx_2 \cdot x_1 \cdot rx_3 \\ +vx_2 \cdot y_1 \cdot ry_3 + vx_2 \cdot z_1 \cdot rz_3 + vz_2 \cdot tz \quad = \quad vz_2 \cdot x_2 - vx_2 \cdot z_2 \\ \\ vy_2 \cdot x_1 \cdot rx_1 + vy_2 \cdot y_1 \cdot ry_1 + vy_2 \cdot z_1 \cdot rz_1 + vy_2 \cdot tx - vx_2 \cdot x_1 \cdot rx_2 \\ -vx_2 \cdot y_1 \cdot ry_2 - vx_2 \cdot z_1 \cdot rz_2 - vx_2 \cdot ty \quad = \quad vx_2 \cdot y_2 - vy_2 \cdot x_2 \end{cases}$$

• **Relation "DROITE-PLAN" : (D1 SUR F2)**

avec D1 = (P1 , V1) où P1 = (x_1 y_1 z_1), V1 = (vx_1 vy_1 vz_1),
et F2 = (P2 , N2) où P2 = (x_2 y_2 z_2), N2 = (nx_2 ny_2 nz_2).

Après le déplacement la relation géométrique est vérifiée si :

1. La droite et le plan sont parallèles, c'est à dire le vecteur directeur de la droite est perpendiculaire à la normale au plan (leur produit scalaire est nul).

2. Un point de la droite transformée appartient au plan, comme tout point X de ce dernier il doit vérifier $N2 \times (X - P2) = 0$.

La droite transformée D'1 décrite par $(Tr*P1, Tr*V1)$ doit respecter les contraintes 1 et 2. On obtient donc les équations suivantes :

$$\begin{cases} (Tr * V1) \times N2 = 0 \\ N2 \times (Tr * P1 - P2) = 0 \end{cases}$$

Les équations paramétriques correspondantes sont alors les suivantes :

$$(3.6) \quad \begin{cases} nx_2 \cdot vx_1 \cdot rx_1 + nx_2 \cdot vy_1 \cdot ry_1 + nx_2 \cdot vz_1 \cdot rz_1 + nx_2 \cdot tx \\ +ny_2 \cdot vx_1 \cdot rx_2 + ny_2 \cdot vy_1 \cdot ry_2 + ny_2 \cdot vz_1 \cdot rz_2 + ny_2 \cdot ty \\ +nz_2 \cdot vx_1 \cdot rx_3 + nz_2 \cdot vy_1 \cdot ry_3 + nz_2 \cdot vz_1 \cdot rz_3 + nz_2 \cdot tz = 0 \\ \\ nx_2 \cdot x_1 \cdot rx_1 + nx_2 \cdot y_1 \cdot ry_1 + nx_2 \cdot z_1 \cdot rz_1 + nx_2 \cdot tx \\ +ny_2 \cdot x_1 \cdot rx_2 + ny_2 \cdot y_1 \cdot ry_2 + ny_2 \cdot z_1 \cdot rz_2 + ny_2 \cdot ty \\ +nz_2 \cdot x_1 \cdot rx_3 + nz_2 \cdot y_1 \cdot ry_3 + nz_2 \cdot z_1 \cdot rz_3 + nz_2 \cdot tz \\ = nx_2 \cdot x_2 + ny_2 \cdot y_2 + nz_2 \cdot z_2 \end{cases}$$

- **Relation "PLAN-POINT" : (F1 SUR P2)**

avec P2 = (x_2 y_2 z_2) ,
et F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1).

La transformation Tr permettant d'amener le plan F1 sur le point P2 peut s'exprimer comme suit :

$$Tr * N1 \times (P2 - Tr * P1) = 0$$

Cette équation traduit le fait que le plan déplacé ($Tr*P1, Tr*N1$) doit contenir le point P2. Ce point doit donc vérifier l'équation d'appartenance au plan. L'équation

paramétrique correspondante est la suivante :

$$
\begin{array}{l}
nx_1 \cdot x_2 \cdot rx_1 + ny_1 \cdot x_2 \cdot ry_1 + nz_1 \cdot x_2 \cdot rz_1 + x_2 \cdot tx - nx_1 \cdot x_1 \cdot rx_1^2 \\
-ny_1 \cdot y_1 \cdot ry_1^2 - nz_1 \cdot z_1 \cdot rz_1^2 - tx^2 - (nx_1 \cdot y_1 + ny_1 \cdot x_1) \cdot rx_1 \cdot ry_1 \\
-(nx_1 \cdot z_1 + nz_1 \cdot x_1) \cdot rx_1 \cdot rz_1 - (ny_1 \cdot z_1 + nz_1 \cdot y_1) \cdot ry_1 \cdot rz_1 \\
-(ny_1 + y_1) \cdot ry_1 \cdot tx - (nx_1 + x_1) \cdot rx_1 \cdot tx - (nz_1 + z_1) \cdot rz_1 \cdot tx \\
+nx_1 \cdot x_2 \cdot rx_2 + ny_1 \cdot x_2 \cdot ry_2 + nz_1 \cdot x_2 \cdot rz_2 + x_2 \cdot ty - nx_1 \cdot x_1 \cdot rx_2^2 \\
-ny_1 \cdot y_1 \cdot ry_2^2 - nz_1 \cdot z_1 \cdot rz_2^2 - ty^2 - (nx_1 \cdot y_1 + ny_1 \cdot x_1) \cdot rx_2 \cdot ry_2 \\
-(nx_1 \cdot z_1 + nz_1 \cdot x_1) \cdot rx_2 \cdot rz_2 - (ny_1 \cdot z_1 + nz_1 \cdot y_1) \cdot ry_2 \cdot rz_2 \\
-(ny_1 + y_1) \cdot ry_2 \cdot ty - (nx_1 + x_1) \cdot rx_2 \cdot ty - (nz_1 + z_1) \cdot rz_2 \cdot ty \\
+nx_1 \cdot x_2 \cdot rx_3 + ny_1 \cdot x_2 \cdot ry_3 + nz_1 \cdot x_2 \cdot rz_3 + x_2 \cdot tz - nx_1 \cdot x_1 \cdot rx_3^2 \\
-ny_1 \cdot y_1 \cdot ry_3^2 - nz_1 \cdot z_1 \cdot rz_3^2 - tz^2 - (nx_1 \cdot y_1 + ny_1 \cdot x_1) \cdot rx_3 \cdot ry_3 \\
-(nx_1 \cdot z_1 + nz_1 \cdot x_1) \cdot rx_3 \cdot rz_3 - (ny_1 \cdot z_1 + nz_1 \cdot y_1) \cdot ry_3 \cdot rz_3 \\
-(ny_1 + y_1) \cdot ry_3 \cdot tz - (nx_1 + x_1) \cdot rx_3 \cdot tz - (nz_1 + z_1) \cdot rz_3 \cdot tz = 0
\end{array}
\tag{3.7}
$$

- **Relation "PLAN-DROITE" : (F1 SUR D2)**

avec F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1),
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

La transformation Tr permettant d'amener le plan F1 sur la droite D2 s'exprime à l'aide des équations suivantes :

$$
\begin{cases}
(Tr * N1) \times V2 = 0 \\
(Tr * P1 - P2) \wedge V2 = O
\end{cases}
$$

Ces deux équations traduisent respectivement les propriétés suivantes :

1. Le plan déplacé doit être parallèle à la droite et donc, le vecteur normal transformé ($Tr * N1$) doit être perpendiculaire à V2, vecteur directeur de la droite D2. Pour cela il faut que leur produit scalaire soit nul.

2. Un point du plan déplacé ($Tr * P1$) doit appartenir à la droite D2 et donc vérifier comme tout point X de cette droite la relation $(X - P2) \wedge V2 = O$.

Les équations paramétriques correspondantes sont les suivantes :

$$
(3.8)\quad \begin{cases}
nx_1 \cdot vx_2 \cdot rx_1 + ny_1 \cdot vx_2 \cdot ry_1 + nz_1 \cdot vx_2 \cdot rz_1 + vx_2 \cdot tx \\
+nx_1 \cdot vy_2 \cdot rx_2 + ny_1 \cdot vy_2 \cdot ry_2 + nz_1 \cdot vy_2 \cdot rz_2 + vy_2 \cdot ty \\
+nx_1 \cdot vz_2 \cdot rx_3 + ny_1 \cdot vz_2 \cdot ry_3 + nz_1 \cdot vz_2 \cdot rz_3 + vz_2 \cdot tz = 0 \\
\\
vz_2 \cdot x_1 \cdot rx_2 + vz_2 \cdot y_1 \cdot ry_2 + vz_2 \cdot z_1 \cdot rz_2 + vz_2 \cdot ty - vy_2 \cdot x_1 \cdot rx_3 \\
-vy_2 \cdot y_1 \cdot ry_3 - vy_2 \cdot z_1 \cdot rz_3 - vy_2 \cdot tz = vy_2 \cdot z_2 - vz_2 \cdot y_2 \\
\\
-vz_2 \cdot x_1 \cdot rx_1 - vz_2 \cdot y_1 \cdot ry_1 - vz_2 \cdot z_1 \cdot rz_1 - vz_2 \cdot tx + vx_2 \cdot x_1 \cdot rx_3 \\
+vx_2 \cdot y_1 \cdot ry_3 + vx_2 \cdot z_1 \cdot rz_3 + vx_2 \cdot tz = vz_2 \cdot x_2 - vx_2 \cdot z_2 \\
\\
vy_2 \cdot x_1 \cdot rx_1 + vy_2 \cdot y_1 \cdot ry_1 + vy_2 \cdot z_1 \cdot rz_1 + vy_2 \cdot tx - vx_2 \cdot x_1 \cdot rx_2 \\
-vx_2 \cdot y_1 \cdot ry_2 - vx_2 \cdot z_1 \cdot rz_2 - vx_2 \cdot ty = vx_2 \cdot y_2 - vy_2 \cdot x_2
\end{cases}
$$

- **Relation "PLAN-PLAN" : (F1 SUR F2)**

avec F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1),
et F2 = (P2 , N2) où P2 = (x_2 y_2 z_2), N2 = (nx_2 ny_2 nz_2).

La transformation Tr permettant d'amener le plan F1 sur le plan F2 s'exprime à l'aide des équations vectorielles suivantes :

$$
\begin{cases}
(Tr * N1) \wedge N2 = O \\
N2 \times (Tr * P1 - P2) = 0
\end{cases}
$$

Ces deux équations traduisent respectivement les propriétés suivantes :

1. Les deux plans doivent être parallèles, donc leurs normales doivent être colinéaires. Le produit vectoriel de la normale du plan déplacé ($Tr * N1$) par la normale N2 doit donc être le vecteur nul.

2. Un point du plan déplacé ($Tr * P1$) doit appartenir à F2, et vérifier comme tout point X de ce plan l'équation $N2 \times (X - P2) = 0$.

Les équations paramétriques correspondantes sont les suivantes :

$$
(3.9)\quad \begin{cases}
nz_2 \cdot nx_1 \cdot rx_2 + nz_2 \cdot ny_1 \cdot ry_2 + nz_2 \cdot nz_1 \cdot rz_2 + nz_2 \cdot ty \\
-ny_2 \cdot nx_1 \cdot rx_3 - ny_2 \cdot ny_1 \cdot ry_3 - ny_2 \cdot nz_1 \cdot rz_3 - ny_2 \cdot tz \\
\quad = \quad ny_2 \cdot nz_2 - nz_2 \cdot ny_2 \\
\\
-nz_2 \cdot nx_1 \cdot rx_1 - nz_2 \cdot ny_1 \cdot ry_1 - nz_2 \cdot nz_1 \cdot rz_1 - nz_2 \cdot tx \\
+nx_2 \cdot nx_1 \cdot rx_3 + nx_2 \cdot ny_1 \cdot ry_3 + nx_2 \cdot nz_1 \cdot rz_3 + nx_2 \cdot tz \\
\quad = \quad nz_2 \cdot nx_2 - nx_2 \cdot nz_2 \\
\\
ny_2 \cdot nx_1 \cdot rx_1 + ny_2 \cdot ny_1 \cdot ry_1 + ny_2 \cdot nz_1 \cdot rz_1 + ny_2 \cdot tx \\
-nx_2 \cdot nx_1 \cdot rx_2 - nx_2 \cdot ny_1 \cdot ry_2 - nx_2 \cdot nz_1 \cdot rz_2 - ny_2 \cdot ty \\
\quad = \quad nx_2 \cdot ny_2 - ny_2 \cdot nx_2 \\
\\
nx_2 \cdot x_1 \cdot rx_1 + nx_2 \cdot y_1 \cdot ry_1 + nx_2 \cdot z_1 \cdot rz_1 + nx_2 \cdot tx + ny_2 \cdot x_1 \cdot rx_2 \\
+ny_2 \cdot y_1 \cdot ry_2 + ny_2 \cdot z_1 \cdot rz_2 + ny_2 \cdot ty + nz_2 \cdot x_1 \cdot rx_3 + nz_2 \cdot y_1 \cdot ry_3 \\
+nz_2 \cdot z_1 \cdot rz_3 + nz_2 \cdot tz \quad = \quad nx_2 \cdot x_2 + ny_2 \cdot y_2 + nz_2 \cdot z_2
\end{cases}
$$

3.3.3 Principe de résolution

Une situation géométrique est écrite sous forme d'un système d'équations paramétriques à douze inconnues. Le principe est donc de résoudre un système de douze équations à douze inconnues, la solution devant ensuite vérifier les relations : $\|\mathrm{r1}\| = \|\mathrm{r2}\| = \|\mathrm{r3}\| = 1$ et $\mathrm{r1} \cdot \mathrm{r2} = \mathrm{r1} \cdot \mathrm{r3} = \mathrm{r2} \cdot \mathrm{r3} = 0$.

Le système est alors résolu par la méthode de réduction de Gauss.

3.3.4 Avantages et limites de la méthode

Cette méthode présente certains inconvénients :

* La principale difficulté réside dans le fait que les équations 3.4 (cas droite-point) et 3.7 (cas plan-point) ne sont pas linéaires. Elles ne peuvent pas être résolues selon le principe de résolution "générale" exposé ci-dessus puisque celle-ci ne prend en compte que des équations linéaires.

  Si nous choisissons une telle méthode, les cas droite-point et plan-point sont alors interdits.

* Il faut rechercher douze inconnues. Le nombre d'équations paramétriques pour une relation varie de un à six (six dans le cas d'une relation droite-droite). Donc, la résolution d'une seule relation oblige à résoudre un système

de une à six équations linéaires à douze inconnues, ce qui conduit à une indétermination.

Une telle méthode ne permet pas de résoudre une situation géométrique ne comportant qu'une seule relation. Ceci traduit le fait que dans notre système une relation géométrique ne décrit jamais de façon unique une position relative entre deux objets.

* Pour obtenir une solution unique il faut donner plusieurs relations induisant un système d'au moins douze équations. Ceci parait normal vu le principe de résolution employé. Cependant, cela nécessite de fournir de nombreuses relations alors que l'utilisateur dans certains cas, peut avoir besoin de mettre deux objets en relation sans position très précise. Par exemple, face-1 SUR face-2, car il souhaite ensuite réaliser d'autres relations qui ne nécessitent pas de préciser de quelle manière les faces doivent être, autre que l'une contre l'autre.

En conclusion :
Cette méthode ne permet pas de résoudre une situation comportant :

- une relation droite-point ou plan-point,

- un nombre insuffisant d'équations (en particulier une seule relation dans la situation).

Il est donc nécessaire de développer une méthode permettant de résoudre ces deux points.

3.3.5 Méthode de "pré-résolution générale"

Afin de pallier aux insuffisances de la méthode de résolution générale, nous avons envisagé de traiter les situations géométriques par étude de cas. Compte tenu du fait qu'il y a peu de relations différentes, toutes les combinaisons possibles de relations sont étudiées auparavant afin d'avoir un traitement type pour chaque combinaison possible.

Cette méthode pose toutefois deux inconvénients majeurs :

* le nombre de combinaisons possibles est important : trente possibilités différentes pour une combinaison de deux relations, deux cent vingt neuf pour trois relations ;

* il devient très difficile de rajouter d'autres cas.

En conclusion, cette méthode est fastidieuse à développer, mais surtout elle poserait beaucoup trop de problèmes pour les extensions futures. Aussi nous avons abandonné l'idée d'utiliser une telle méthode ou du moins sous une forme d'étude de cas aussi générale. Nous verrons par la suite qu'elle peut être employée pour résoudre une seule relation en la combinant avec un "système itératif".

3.4 Utilisation de l'interpréteur géométrique de LM-GEO

LM-GEO est une extension du langage de manipulation LM, afin de rendre possible la définition de positions d'objets par des relations géométriques explicites sur ces objets.

LM-GEO [Maz 87],[Maz 82b] comporte un module de résolution permettant de déduire la position relative entre deux objets à partir d'un ensemble de relations géométriques décrites symboliquement. C'est cette partie d'interprétation des situations géométriques qui nous intéresse ici.

Par rapport à la méthode générale présentée précédemment, elle nous permet d'obtenir une solution avec moins de relations dans une situation géométrique. En effet l'interpréteur géométrique de LM ramène les relations en un système "pseudo-linéaire" à cinq inconnues. Nous allons décrire cette méthode au paragraphe suivant.

Son intérêt :

L'intérêt qu'il représente dans notre cas, réside dans sa partie résolution de situations géométriques.

Il calcule la position relative entre deux objets à partir de relations géométriques, ce qui nous permet d'en déduire la transformation de passage de la position initiale à la situation finale. Le problème présenté dans cette partie (traduction d'une situation géométrique en un repère déplaçable et une transformation) peut alors être résolu.

Dans l'approche développée, il se place comme système de résolution de situations géométriques décrites symboliquement.

3.4.1 Principe de résolution de LM-GEO

Les relations traitées par le système LM-GEO mettent en jeu des points, des droites et des plans. Cela conduit à traiter les opérateurs relationnels suivants :

point-sur-point

plan-contient-point

plan-sur-droite

droite-contient-point

contre [distance]

coplanaire

aligne [distance].

Ceux-ci se rapprochent des relations décrites en 3.2. Il convient de noter que ces relations ne sont pas symétriques. Une relation "plan-sur-droite" conduit par exemple à des équations linéaires ce qui n'est pas le cas pour la relation "droite-sur-plan" (c'est pour cela qu'elle ne figure pas dans la liste des relations que traite LM-GEO).

Représentation des entités géométriques

Les plans, droites et points sont représentés à l'aide du formalisme de Plücker [Bra 53].

* Un point est représenté par un vecteur (a_0).
* Une droite est représentée par un couple de vecteurs (a_0, a) où a est un vecteur unitaire et a_0 un vecteur tel que tout point m appartient à (a_0, a) si et seulement si $m \wedge a = a_0$.
* Un plan est représenté par un doublet (σ, a) où σ est un scalaire et a un vecteur unitaire. Un point m appartient à (σ, a) si et seulement si $m \times a = \sigma$.

Changement de repère de référence pour la résolution :

Ce changement de repère est effectué pour que les deux objets aient l' axe "Z" de leur repère identique. Ainsi les deux repères ne diffèrent plus que par une translation et une rotation autour de Z, ce qui permet de réduire le nombre d'inconnues.

- **Translation :** Soient F_A et F'_A deux repères de références tels que F'_A soit déduit de F_A par une translation T. Si t représente le vecteur translation dans F_A alors nous pouvons écrire les relations suivantes :

  - point : $[(a_0)]F_A \iff [(a_0 - t)]F'_A$
  - droite : $[(a_0, a)]F_A \iff [(a_0 - t \wedge a, a)]F'_A$
  - plan : $[(\sigma, a)]F_A \iff [(\sigma - t \times a, a)]F'_A$.

- **Rotation :** Si F_B est déduit de F'_A par une rotation R et si r est l'opérateur de rotation qui transforme un vecteur $[v]F_B$ en $[r \star v]F'_A$, alors en supposant que r^{-1} est l'opérateur de rotation inverse de r, on peut écrire :

  - point : $[(a_0)]F'_A \iff [(r^{-1} \star a_0)]F_B$
  - droite : $[(a_0, a)]F'_A \iff [(r^{-1} \star a_0, r^{-1} \star a)]F_B$
  - plan : $[(\sigma, a)]F'_A \iff [(\sigma, r^{-1} \star a)]F_B$.

Principe de résolution

Les relations exposées à la page précédente sont alors décrites en équations numériques à l'aide des représentations données ci-dessus. La résolution est détaillée dans [Maz 82], [Maz 82b] et [Maz 87]. Elle consiste à trouver en premier lieu une "direction commune" aux deux relations traitées afin de se ramener à des équations "pseudo-linéaires" par changement de repère. Les cinq inconnues du système sont alors les suivantes : la translation (tx ty tz) et les paramètres de la rotation autour de l'axe z, c et s qui doivent respecter la propriété $c^2 + s^2 = 1$. Les relations sont alors traduites en équations paramétriques à l'aide de ces cinq inconnues. Le système ainsi obtenu avec les inconnues tx, ty, tz, c et s est linéaire, sachant que ses solutions doivent vérifier la relation $c^2 + s^2 = 1$.

En sortie, le système fournit la transformation $T_{AB'}$ qui est la position de l'objet mobile par rapport à l'objet "fixe" (voir figure 3.1).

3.4.2 Interfaçage avec LM-OBJ

Utiliser cet algorithme dans l'interpréteur du langage cible de SHARP nécessite un interface d'entrée permettant de coder les entités géométriques et les relations dans le formalisme adéquat, et un interface de sortie permettant de convertir les solutions en "déplacements".

Interface d'entrée

Dans la base de données, *les points* sont représentés par leurs coordonnées dans le repère du solide auquel ils appartiennent. Ils seront décrits pour LM-GEO par le vecteur entre l'origine du repére de l'objet(O) et le point étudié (P) :
V_p = P - O = P.
Les droites décrites par un point et un vecteur directeur unitaire, doivent l'être par le même vecteur unitaire et un vecteur orthogonal : différence entre la projection

orthogonale de l'origine sur la droite et de l'origine elle-même .
Les plans définis par un point (p) et une normale (n) doivent l'être par la normale unitaire et la constante σ ($\sigma = p \wedge n$).

Les relations traitées sont du type (entite-1 SUR entite-2). Elles doivent être traduites en équations numériques au format LM-GEO (liste-de-valeurs-numériques "relation" liste-de-valeurs-numériques) où relation est un des opérateurs cités en 3.4.1 . La liste de valeurs numériques contient en fonction du type de l'entité : le vecteur point, le vecteur directeur unitaire et le vecteur orthogonal ou encore le vecteur normal unitaire et la constante σ.

Interface de sortie

Les résultats obtenus peuvent se diviser en trois catégories :

* Pas de solution pour la situation géométrique décrite par l'utilisateur (système incohérent).

* Le système étant sous-contraint, aucune solution n'est proposée alors qu'il en existe une infinité. Dans ce cas le système envoie une solution du type (L 3), ou L indique que le système est sous-contraint (si le système n'accepte que deux, trois ou quatre solutions alors elles sont calculées).

* Une transformation donnant la position relative entre les deux objets. Il faut alors en déduire la transformation représentant le mouvement à appliquer à l'objet mobile afin de le positionner à l'endroit voulu.

Calcul du déplacement

Soient un objet A mobile et un objet B immobile, la position de A est connue par son repère T_A (transformation par rapport au référentiel de base), celle de B par son repère T_B. LM-GEO donne la position relative entre l'objet B et la position souhaitée pour l'objet A désignée A', cette transformation est notée $T_{A'B}$ (figure 3.1).

Il faut calculer $T_{AA'}$ la transformation permettant d'amener l'objet mobile à la position souhaitée qui peut s'écrire :
$T_{AA'} = T_{A'B} \circ T_B \circ T_A^{-1}$.

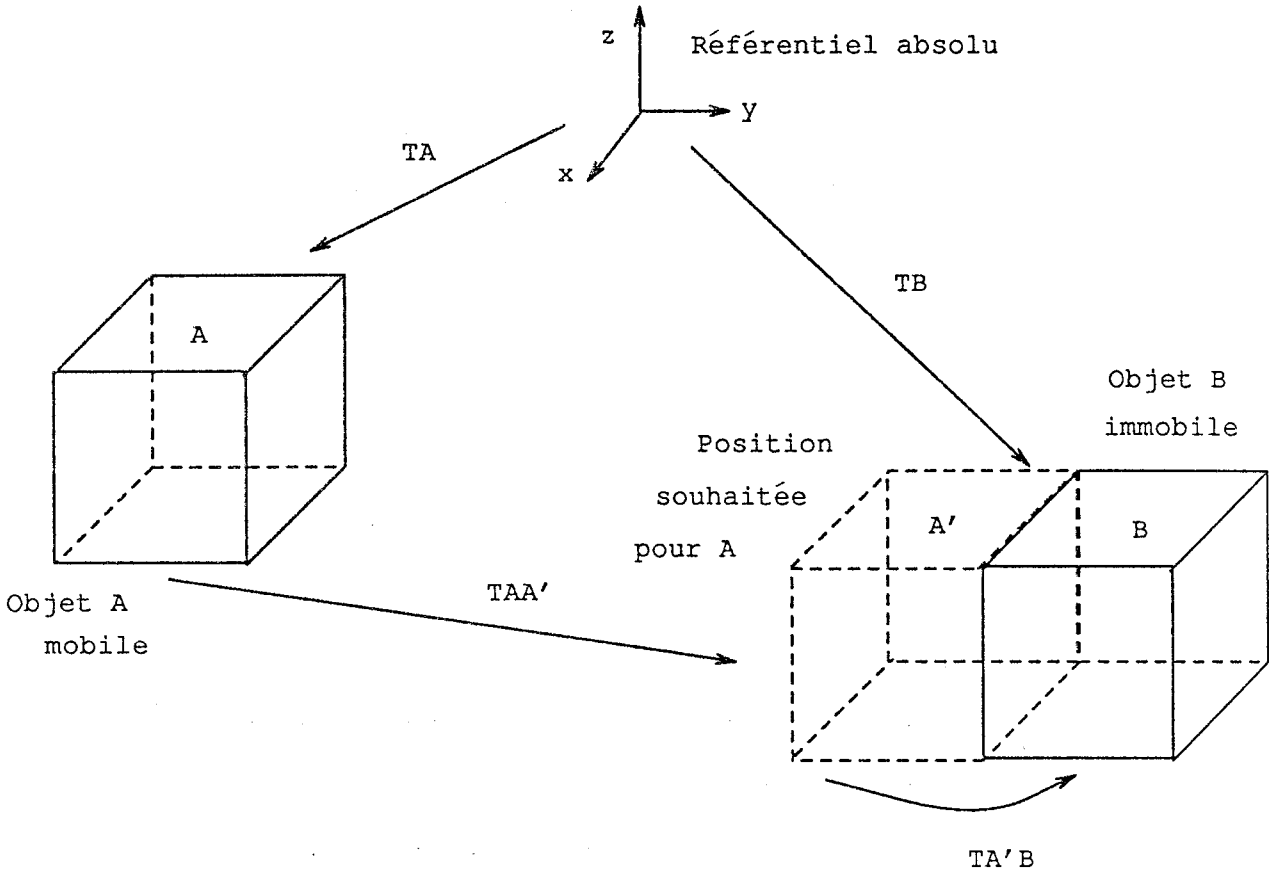


Figure 3.1: position relative entre les deux objets

3.4.3 Avantages et limites de la méthode

Cette méthode présente des limitations qui conduisent à restreindre les situations pouvant être étudiées (en particulier à cause du fait qu'il n'est pas toujours possible d'obtenir des équations pseudo-linéaires).

* Une relation *plan-plan* ou une relation *droite-droite* ou encore *deux relations point-point* doivent obligatoirement être présentes dans la situation analysée, afin de pouvoir trouver une "direction commune" pour la résolution (c'est à dire fixer une rotation).

* La *position relative* entre les deux objets doit être *parfaitement définie* par l'ensemble des relations de la situation géométrique, sinon il existe une infinité de solutions et le système n'en calcule en général aucune car les équations représentent un système sous contraint ; il peut faire le calcul que s'il n'y a que quelques solutions (trois ou quatre). En particulier il est obligatoire de

donner au moins deux relations.

* Les relations plan-droite, plan-point et droite-point peuvent être traitées par la méthode, alors que les relations symétriques (droite-plan, point-plan, point-droite) ne peuvent pas être prises en compte, car elles conduisent à un système d'équations non linéaire (ce qui ne peut pas être résolu par l'algorithme employé). Une solution possible dans ce cas est d'inverser toutes les relations : l'objet fixe apparait en première position dans la relation. En fin de calcul la transformation est inversée pour obtenir le déplacement correct à effectuer. Toutefois il est clair que si *une relation de chaque groupe* est présente, alors il n'est *pas possible de résoudre le système* par cette méthode.

L'avantage principal de cet algorithme est que, dans le domaine d'utilisation décrit ci-dessus, toute solution existante sera trouvée. Il nous parait alors intéressant d'utiliser cette méthode lorsque cela est possible. Nous allons étudier d'autres méthodes permettant de trouver une solution lorsqu'il n'est pas possible d'employer l'interpréteur géométrique de LM-GEO.

3.5 Méthode de résolution dans le cas de relations isolées

Une des limites d'utilisation de LM-GEO est qu'il faut suffisamment de relations pour définir sans ambiguïté la position relative de deux corps (c'est à dire suffisamment de contraintes et donc d'équations). L'utilisateur peut souhaiter décrire une situation à l'aide d'une seule relation s'il n'a pas dans l'immédiat besoin d'une position très précise d'un objet vis à vis d'un autre (en particulier une face sur une autre sans autre spécification).

Comme nous l'avons constaté précédemment, dans notre cas une relation ne décrit jamais de façon unique la position relative entre deux objets. Aussi doit on rajouter d'autres contraintes afin de n'obtenir qu'une solution au lieu d'une infinité.

3.5.1 Contraintes supplémentaires introduites

Afin de restreindre l'ensemble des solutions possibles en cas d'ambiguïté, nous avons introduit deux contraintes supplémentaires. Les contraintes doivent être compatibles avec les objectifs visés par notre système : trouver un mouvement si possible "minimal" (soit éviter les mouvements inutiles).

1. Le déplacement calculé doit être minimal (la plus petite translation en particulier).
2. Les rotations sont utilisées uniquement lorsque cela s'avère indispensable (amener deux droites ou deux plans à être parallèles, ...).

Ces contraintes supplémentaires conduisent donc à choisir la solution optimale parmi toutes les solutions possibles. Le système obtenu par introduction de ces nouvelles contraintes n'admet alors qu'une seule solution. Mais il est nécessaire pour formaliser ces nouvelles contraintes, d'analyser les différents cas possibles.

3.5.2 Analyse de chaque relation

Le formalisme utilisé pour décrire les équations et les inconnues est identique à celui employé en 3.3.

- **Relation "POINT-POINT" : (P1 SUR P2)**

avec P1 = (x_1 y_1 z_1) et P2 = (x_2 y_2 z_2).

En vertu du principe 1 de mouvement minimum, nous n'utilisons alors qu'un mouvement de translation. Les inconnues qui restent sont donc *tx, ty, tz* (la matrice de rotation étant la matrice identité). Les équations paramétriques numérotées 3.1 deviennent :

$$\begin{cases} x_1 + tx = x_2 \\ y_1 + ty = y_2 \\ z_1 + tz = z_2 \end{cases}$$

La solution correspondante est alors la suivante : T = P2 - P1.

• Relation "POINT-DROITE" : (P1 SUR D2)

avec P1 = (x_1 y_1 z_1)
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

En vertu de la contrainte 1 de déplacement minimal nous n'utilisons qu'un mouvement de translation. Les inconnues qui restent sont donc *tx ty tz* (la matrice de rotation étant la matrice identité). Le vecteur translation T est alors perpendiculaire au vecteur directeur de la droite (afin d'obtenir un déplacement minimal), et le point déplacé P1' doit appartenir à la droite D2. Nous obtenons ainsi deux équations :

$$\begin{cases} (T \times V2) = 0 \\ P1' = P2 + t \cdot V2 \ \ avec \ \ t \ \ un \ \ réel \end{cases}$$

Nous pouvons écrire T = P1' - P1. Nous obtenons alors la solution :

$$\begin{cases} T = P2 + t \cdot V2 - P1 \\ t = \frac{V2 \times (P1 - P2)}{\|V2\|} \end{cases}$$

• Relation "POINT-PLAN' : (P1 SUR F2)

avec P1 = (x_1 y_1 z_1)
et F2 = (P2 , N2) où P2 = (x_2 y_2 z_2), N2 = (nx_2 ny_2 nz_2).

En vertu de la contrainte 1 de déplacement minimum nous n'utilisons qu'un mouvement de translation. Les inconnues qui restent sont donc *tx ty tz* (la matrice de

rotation étant la matrice identité). Le vecteur translation T est alors perpendiculaire au plan F2, donc T est colinéaire à la normale du plan F2 notée N2 (afin d'obtenir un déplacement minimum). De même le point déplacé doit appartenir au plan F2. Nous obtenons donc deux équations :

$$\begin{cases} T = t \cdot N2 \\ N2 \times (P1 + T - P2) = 0 \end{cases}$$

La solution est alors la suivante :

$$\begin{cases} T = t \cdot N2 \\ t = \frac{N2 \times (P2 - P1)}{\|N2\|} \end{cases}$$

- **Relation "DROITE-POINT" : (D1 SUR P2)**

avec P2 = (x_2 y_2 z_2),
et D1 = (P1 , V1) où P1 = (x_1 y_1 z_1), V1 = (vx_1 vy_1 vz_1).

Nous résolvons la relation "symétrique" (P2 SUR D2) comme un cas "point-droite", la translation obtenue est alors inversée.

- **Relation "DROITE-DROITE" : (D1 SUR D2)**

avec D1 = (P1 , V1) où P1 = (x_1 y_1 z_1), V1 = (vx_1 vy_1 vz_1),
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

Nous distinguons trois cas :

Cas 1 : D1 et D2 sont parallèles : $V1 \times V2 = 0$

Dans ce cas nous ramenons le problème à la projection d'un point de D1 (par exemple P1) sur la droite D2. Il s'agit alors de résoudre une relation "point-droite".

Cas 2 : D1 et D2 sont sécantes :

Il faut alors calculer :

- *Le point d'intersection* entre les deux droites noté X = (x y z). Il peut s'exprimer à l'aide des équations paramétriques des droites D1 et D2 :

$$X = P1 + t_1 \cdot V1 = P2 + t_2 \cdot V2, avec t_1 et t_2$$

deux réels.

D'où les trois équations à deux inconnues t_1 et t_2 à résoudre :

$$\begin{cases} t_1 \cdot vx_1 - t_2 \cdot vx_2 = x_2 - x_1 \\ t_1 \cdot vy_1 - t_2 \cdot vy_2 = y_2 - y_1 \\ t_1 \cdot vz_1 - t_2 \cdot vz_2 = z_2 - z_1 \end{cases}$$

- *Un des angles minimaux* entre ces deux droites noté α :

$$\cos \alpha = \frac{V1 \times V2}{\|V1\| \cdot \|V2\|}$$

- *L'axe de rotation* est perpendiculaire aux deux droites. Le vecteur directeur de cet axe est noté A = (ax ay az).

$$\begin{cases} V1 \times A = 0 \\ V2 \times A = 0 \end{cases}$$

Il faut alors résoudre un système de deux équations à trois inconnues sachant que l'origine de l'axe est le point X (la rotation sera appliquée en un repère centré en X).

$$\begin{cases} vx_1 \cdot ax + vy_1 \cdot ay + vz_1 \cdot az = 0 \\ vx_2 \cdot ax + vy_2 \cdot ay + vz_2 \cdot az = 0 \end{cases}$$

SOLUTION : La transformation permettant d'amener D1 sur D2 est une rotation
- d'angle

$$\alpha = \arccos \frac{V1 \times V2}{\|V1\| \cdot \|V2\|}$$

- autour de l'axe A = (ax , ay , az) avec

$$\begin{cases} ax = vy_1 \cdot vz_2 - vz_1 \cdot vy_2 \\ ay = vz_1 \cdot vx_2 - vx_1 \cdot vz_2 \\ az = vx_1 \cdot vy_2 - vy_1 \cdot vx_2 \end{cases}$$

- appliquée en un point centré en X = P1 + t · V1 avec $t = \frac{vy_2 \cdot (x_2 - x_1) - vx_2 \cdot (x_2 - x_1)}{vx_2 \cdot vy_1 - vx_1 \cdot vy_2}$

Cas 3 : D1 et D2 sont quelconques.

- Comme précédemment il faut calculer *l'angle* α entre les deux droites.

- *L'axe de rotation* est la perpendiculaire commune aux deux droites. Pour trouver sa direction il faut calculer l'intersection entre les deux plans dont les normales sont les vecteurs directeurs des deux droites. Nous obtenons alors deux équations :

$$\begin{cases} |\, X - P1, V1, V1 \wedge V2 \,| &= 0 \\ |\, X - P2, V2, V1 \wedge V2 \,| &= 0 \end{cases}$$

- L'origine de l'axe est K1 *le point d'intersection* entre la perpendiculaire commune et la droite D1. Ce point a pour valeur K1 = P1 + t · V1. D'autre part il vérifie l'équation du plan [X - P2 , V2 , V1 ∧ V2] = 0, dont la normale est notée NP2 = (npx_2 npy_2 npz_2). Nous obtenons alors une équation à une inconnue t :

$npx_2 \cdot (x_1 + t \cdot vx_1 - x_2) + npy_2 \cdot (y_1 + t \cdot vy_1 - y_2) + npz_2 \cdot (z_1 + t \cdot vz_1 - z_2) = 0$

SOLUTION : La transformation est une rotation

- d'angle

$$\alpha = \arccos \frac{V1 \times V2}{\|V1\| \cdot \|V2\|}$$

- autour de l'axe A = (ax ay az), avec

$$\begin{cases} ax = (vz_2 \cdot vy_1 - vy_2 \cdot vz_1) \cdot \|V1 \wedge V2\| \\ ay = (vx_2 \cdot vz_1 - vz_2 \cdot vx_1) \cdot \|V1 \wedge V2\| \\ az = (vy_2 \cdot vx_1 - vx_2 \cdot vy_1) \cdot \|V1 \wedge V2\| \end{cases}$$

- appliquée au point K1 = P1 + t · V1 avec

$$t = \frac{NP2 \times (P2 - P1)}{V1 \times NP2}$$

- à composer avec la translation permettant d'amener K1 sur K2 : K2 le point d'intersection entre la perpendiculaire commune et D2.

T = K2 - K1 = P2 + t_2 · V2 - P1 + t · V1

$T = P2 + \frac{NP1 \times (P2-P1)}{V2 \times NP1} \cdot V2 - P1 - \frac{NP2 \times (P2-P1)}{V1 \times NP2} \cdot V1$

• **Relation "DROITE-PLAN" : (D1 SUR F2)**

avec D1 = (P1 , V1) où P1 = (x_1 y_1 z_1), V1 = (vx_1 vy_1 vz_1),
et F2 = (P2 , N2) où P2 = (x_2 y_2 z_2), N2 = (nx_2 ny_2 nz_2).

Nous distinguons deux cas :

Cas 1 : D1 et F2 sont parallèles : V1 × N2 = 0.
Le problème est ramené à la projection d'un point de D1 (par exemple P1) sur le plan F2. Il s'agit alors de résoudre une relation "point-plan".

Cas 2 : D1 et F2 sont sécants.
Il faut alors calculer :

- *Le point d'intersection* entre la droite et le plan noté X = (x y z).

$$\begin{cases} X = P1 + t \cdot V1 \ \ avec \ \ t \ \ un \ \ réel \\ N2 \times (X - P2) = 0 \end{cases}$$

D'où N2 × (p1 + t · V1 - P2) = 0, nous obtenons alors

$$t = \frac{N2 \times (P2 - P1)}{N2 \times V1}$$

- *l'angle* α entre la droite et le plan à l'aide de l'équation

$$\sin \alpha = \frac{V1 \times V2}{\|V1\| \cdot \|V2\|}$$

- *L'axe de rotation* dont l'origine est le point d'intersection X entre la droite et le plan et le vecteur directeur A = (ax ay az), est perpendiculaire à D1 et dans F2.

$$\begin{cases} A \times V1 = 0 \\ A \times N2 = 0 \end{cases}$$

D'où deux équations paramétriques à trois inconnues à résoudre.

$$\begin{cases} vx_1 \cdot ax + vy_1 \cdot ay + vz_1 \cdot az = 0 \\ nx_2 \cdot ax + ny_2 \cdot ay + nz_2 \cdot az = 0 \end{cases}$$

SOLUTION : La transformation recherchée est alors la rotation
- d'angle

$$\alpha = \arcsin \frac{V1 \times V2}{\|V1\| \cdot \|V2\|}$$

- autour de l'axe A = (ax ay az), avec :

$$\begin{cases} ax = vy_1 \cdot vz_2 - vy_2 \cdot vz_1 \\ ay = vz_1 \cdot vx_2 - vz_2 \cdot vx_1 \\ az = vx_1 \cdot vy_2 - vx_2 \cdot vy_1 \end{cases}$$

- et centrée au point X = P1 + t. V1 avec

$$t = \frac{N2 \times (P2 - P1)}{N2 \times V1}$$

• **Relation "PLAN-POINT" : (F1 SUR P2)**

avec P2 = (x_2 y_2 z_2) ,
et F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1).

Nous résolvons la relation "symétrique" (P2 SUR F1) comme un cas "point-plan", la translation obtenue est alors inversée.

• **Relation "PLAN-DROITE" : (F1 SUR D2)**

avec F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1),
et D2 = (P2 , V2) où P2 = (x_2 y_2 z_2), V2 = (vx_2 vy_2 vz_2).

Nous résolvons la relation "symétrique" (D2 SUR F1) comme un cas "droite-plan", la transformation obtenue est alors inversée.

• **Relation "PLAN-PLAN" : (F1 SUR F2)**

avec F1 = (P1 , N1) où P1 = (x_1 y_1 z_1), N1 = (nx_1 ny_1 nz_1),
et F2 = (P2 , N2) où P2 = (x_2 y_2 z_2), N2 = (nx_2 ny_2 nz_2).

Nous distinguons deux cas :

Cas 1 : F1 et F2 sont parallèles : N1 $\wedge$ N2 = O.

Dans ce cas le problème est ramené à la projection d'un point du plan F1 (par exemple P1) sur F2. Il s'agit alors de résoudre une relation "point-plan".

Cas 2 : F1 et F2 sont sécants.

Il faut alors calculer :

- *L'angle α minimal* entre les deux normales des plans à l'aide de l'équation

$$\cos\alpha = \frac{N1 \times N2}{\|N1\| \cdot \|N2\|}$$

- *La droite intersection* des deux plans qui sera alors l'axe de rotation. Son vecteur directeur est perpendiculaire aux deux normales.

$$\begin{cases} A \times N1 = 0 \\ A \times N2 = 0 \end{cases}$$

Il suffit alors de résoudre un système de deux équations à trois inconnues :

$$\begin{cases} nx_1 \cdot ax + ny_1 \cdot ay + nz_1 \cdot az = 0 \\ nx_2 \cdot ax + ny_2 \cdot ay + nz_2 \cdot az = 0 \end{cases}$$

- *Le point X* origine de l'axe de rotation est un point quelconque de la droite d'intersection des deux plans :

$$\begin{cases} nx_1 \cdot x + ny_1 \cdot y + nz_1 \cdot z = nx_1 \cdot x_1 + ny_1 \cdot y_1 + nz_1 \cdot z_1 \\ nx_2 \cdot x + ny_2 \cdot y + nz_2 \cdot z = nx_2 \cdot x_2 + ny_2 \cdot y_2 + nz_2 \cdot z_2 \end{cases}$$

SOLUTION : La transformation permettant d'amener F1 sur F2 est la rotation

- d'angle

$$\alpha = \arccos \frac{N1 \times N2}{\|N1\| \cdot \|N2\|}$$

- autour de l'axe A = (ax ay az) avec

$$\begin{cases} ax = ny_1 \cdot nz_2 - nz_1 \cdot ny_2 \\ ay = nz_1 \cdot nx_2 - nx_1 \cdot nz_2 \\ az = nx_1 \cdot ny_2 - ny_1 \cdot nx_2 \end{cases}$$

- centrée en X = (x y z) avec (si $nx_1 \cdot ny_2 - ny_1 \cdot nx_2 \neq 0$) :

$$\begin{cases} x = \frac{ny_2 \cdot (N1 \times P1) - ny_1 \cdot (N2 \times P2)}{nx_1 \cdot ny_2 - ny_1 \cdot nx_2} \\ y = \frac{nx_1 \cdot (N2 \times P2) - ny_2 \cdot (N1 \times P1)}{nx_1 \cdot ny_2 - ny_1 \cdot nx_2} \\ z = 0 \end{cases}$$

Remarques :

Le calcul est exécuté sur les valeurs des entités géométriques exprimées dans le repère de l'objet mobile et non pas sur les valeurs exprimées dans le repère du solide auquel elles appartiennent. Nous obtenons ainsi directement la transformation à appliquer à l'objet mobile.

Les relations sont réorganisées de manière à avoir l'objet mobile en première position dans la relation et le fixe en seconde ; sauf pour les cas plan-droite, plan-point et droite-point où nous inversons ensuite la transformation obtenue.

3.5.3 Avantages et limites de la méthode

Cette méthode garantit une solution pour chaque relation sous la forme d'un déplacement minimal. Les situations décrites comportent en général plus d'une relation, et l'utilisation couplée de cette méthode avec celle de LM-GEO ne résoud pas tous les problèmes : système sous-contraint, présence obligatoire d'une relation plan-plan ou droite-droite ou deux relations point-point.

Le paragraphe suivant présente comment il est possible d'itérer le processus de résolution d'une relation afin de résoudre une situation comportant n relations.

3.6 Méthode de résolution dans le cas de n relations

La composition des transformations obtenues par la technique précédente appliquée à chaque relation prise isolément ne permet pas d'atteindre le but souhaité, car il faut appliquer la contrainte de déplacement minimale sur l'ensemble des relations. Mais une approche globale n'est pas non plus envisageable car elle conduirait à un nombre trop important de cas.

Nous avons donc développé une méthode permettant une résolution "séquentielle" des relations. Les relations sont résolues l'une après l'autre en prenant garde de ne pas rompre les relations déjà réalisées. Cette méthode utilise à la fois la résolution dans le cas de relations isolées, un système de contraintes et un procédé itératif.

3.6.1 Principe de la méthode de résolution séquentielle

Les relations de la situation sont traitées les unes après les autres de façon séquentielle : une relation est résolue comme dans le cas d'une relation isolée (Cf. paragraphe 3.5). Si la transformation obtenue ne vérifie pas les contraintes imposées par les relations précédentes, alors la situation géométrique ne peut pas être résolue. Dans le cas contraire, les contraintes sont mises à jour avec celles imposées par la relation dernièrement résolue. Le repère de l'objet est alors déplacé (dans la base de modèles et non pas physiquement) afin que la relation soit réalisée, et ce pour les calculs suivants. Et ainsi de suite pour chaque relation de la situation géométrique. La transformation géométrique permettant le déplacement sous LM est la composition de toutes les transformations ainsi obtenues.

Remarque :

La première relation est traitée exactement comme au paragraphe 3.5, résolution dans le cas d'une relation isolée. Pour les relations suivantes, la méthode est légèrement différente. En effet, les projections ne se font plus orthogonalement mais dans la direction imposée par les contraintes. Ceci correspond à une relaxation des contraintes supplémentaires 1 et 2 imposées au paragraphe 3.5, relaxation obligatoire pour que les mouvements soient possibles.

Exemple d'utilisation :

Soient un objet A mobile et un objet B "fixe", essayons de réaliser la situation suivante : ((*face-dessous-A* SUR *face-dessus-B*) (*point-A* SUR *arête-inclinée-B*)).

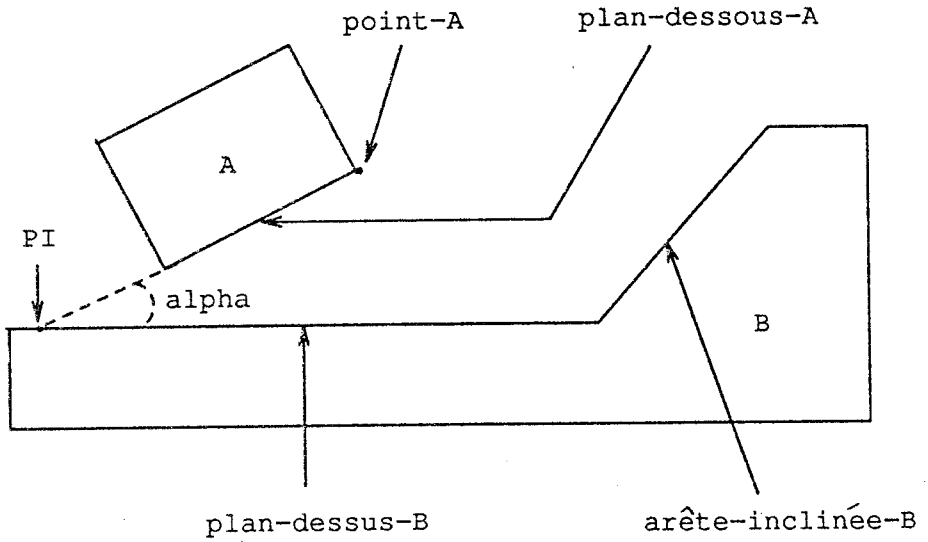


Figure 3.2: Exemple d'utilisation

La première relation est résolue par une rotation α appliquée au point d'intersection PI entre les plans porteurs des deux faces. Nous obtenons alors l'objet A sur l'objet B comme suit.

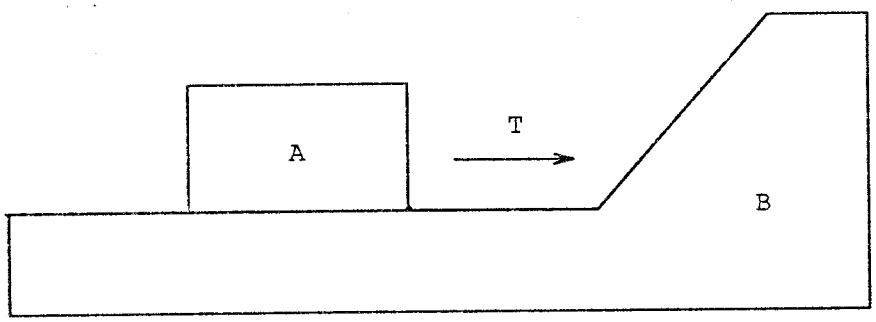


Figure 3.3: Position des deux objets après réalisation de la première relation

La seule translation possible, celle qui permet de maintenir les deux faces l'une contre l'autre, est sur le plan *face-dessus-B*. Les rotations sont permises autour d'un axe parallèle à la normale au plan *face-dessus-B*. La relation suivante ne peut pas être résolue si l'on projette orthogonalement *point-A* sur *arête-inclinée-B*. Il faut donc ne plus tenir compte de la contrainte de distance minimale qui ferait échouer

le calcul, mais faire la projection dans une direction perpendiculaire à la normale du plan *face-dessous-B*.

Algorithme de déroulement de la méthode :

1. Résolution de la première relation apparaissant dans la situation géométrique en utilisant la méthode : résolution dans le cas d'une relation isolée.
2. Initialisation des contraintes introduites dans la première relation. Elles représentent les degrés de liberté du manipulateur qui ne sont pas "bloqués" par la première relation. Chaque relation introduit un certain nombre de contraintes détaillées en 3.6.2.
3. La relation suivante est résolue selon la même méthode que dans le cas une relation isolée. Toutefois les translations, si elles sont contraintes ne correspondent plus à des projections perpendiculaires sur une droite ou un plan, mais à des projections dans la "direction autorisée".
4. Vérification des contraintes qui doivent être respectées par la transformation calculée à l'étape précédente. Par exemple : la dernière transformation est une rotation d'angle α-2 selon l'axe A2 à appliquer au point P2. Les contraintes permettaient une rotation quelconque autour d'un axe A1 et centrée en P1. Si A1 n'est pas parallèle à A2 et si P1 est différent de P2 alors les contraintes ne sont pas respectées (cas d'échec).
   Lorsque les contraintes ne sont pas respectées, il n'y a pas de solution possible avec notre méthode. Dans le cas contraire il reste à modifier la position courante de l'objet à l'aide de la transformation calculée.
5. Il faut combiner les contraintes déjà recensées avec celles imposées par la relation qui vient d'être réalisée. Nous obtenons ainsi un nouvel ensemble de contraintes.
6. Les relations suivantes sont étudiées une par une en itérant au niveau de l'étape 3.

3.6.2 Contraintes associées à chaque type de relation

Chaque type de relation impose donc des contraintes sur les translations et les rotations qui seront ensuite possibles. Ces contraintes sont combinées lors du traitement de la $n^{ième}$ relation afin d'interdire toute transformation pouvant conduire à la non conservation d'une relation établie précédemment. Ceci revient à analyser

les degrés de liberté contraints par les relations, mais aussi à relaxer les contraintes supplémentaires (mouvement minimal) qui ont été introduites pour la résolution.

Les notations utilisées sont les mêmes qu'en 3.3, la translation est notée T, une rotation est caractérisée par un angle α, le vecteur directeur de l'axe noté A = (ax ay az) et un point P origine du repère autour duquel se fera la rotation (= point où la rotation est appliquée).

- **Relation "POINT-POINT" : (P1 SUR P2)**

Toute translation est interdite, seule les rotations autour de P2 (ou P1 déplacé) sont autorisées. La prochaine transformation devra donc respecter les contraintes suivantes :

$$\begin{cases} translation \quad T = (0\ 0\ 0) \\ rotation \quad R = (\ \alpha,\ P2,\ A)\ avec\ \alpha\ et\ A\ quelconques. \end{cases}$$

- **Relation "POINT-DROITE" : (P1 SUR D2)**

Les translations sont autorisées le long de la droite D2. Seules les rotations autour du point P1' (P1 déplacé) sont permises. La prochaine transformation devra donc respecter les contraintes suivantes :

$$\begin{cases} translation \quad T = \gamma \cdot V2\ avec\ \gamma\ quelconque. \\ rotation \quad R = (\ \alpha,\ P1',\ A)\ avec\ \alpha\ et\ A\ quelconques. \end{cases}$$

- **Relation "POINT-PLAN" : (P1 SUR F2)**

Les translations sont autorisées sur le plan F2. Seules les rotations autour du point P1' (P1 déplacé) sont permises. La prochaine transformation devra respecter :

$$\begin{cases} translation \quad T : T \times N2 = 0 \\ rotation \quad R = (\ \alpha,\ P1',\ A)\ avec\ \alpha\ et\ A\ quelconques. \end{cases}$$

- **Relation "DROITE-POINT" : (D1 SUR P2)**

Le cas est identique au cas "point-droite". La prochaine transformation devra respecter :

$$\begin{cases} translation \quad T = \gamma \cdot V1' \ avec \ \gamma \ quelconque. \\ rotation \quad R = (\ \alpha,\ P2,\ A)\ avec\ \alpha\ et\ A\ quelconques. \end{cases}$$

- **Relation "DROITE-DROITE" : (D1 SUR D2)**

Les translations sont autorisées le long de la droite D2. Les rotations sont permises autour de l'axe D2. La prochaine transformation devra respecter :

$$\begin{cases} translation \quad T = \gamma \cdot V2 \ avec \ \gamma \ quelconque. \\ rotation \quad R = (\ \alpha,\ X,\ V2\ )\ avec\ X\ un\ point\ de\ D2 \\ \qquad\qquad\qquad\qquad\qquad\qquad et\ \alpha\ un\ angle\ quelconque. \end{cases}$$

- **Relation "DROITE-PLAN" : (D1 SUR F2)**

Les translations sont permises sur le plan F2. Les rotations sont possibles autour de la droite D1 déplacée, ou autour d'un axe parallèle à la normale au plan F2. La prochaine transformation devra respecter :

$$\begin{cases} translation \quad T : T \times N2 = 0 \\ rotation \quad R = (\ \alpha,\ X,\ \beta \cdot N2\ )\ avec\ X2\ un\ point\ de\ D1', \\ \qquad\qquad\qquad\qquad\qquad\qquad \alpha\ et\ \beta\ quelconques \\ \qquad ou \\ \qquad R = (\ \alpha,\ X1,\ V1'\ )\ avec\ \alpha\ quelconque \\ \qquad\qquad\qquad\qquad\qquad\qquad et\ X1\ un\ point\ de\ D1'. \end{cases}$$

- **Relation "PLAN-POINT" : (F1 SUR P2)**

Ce cas est identique à un cas "point-plan". La prochaine transformation devra respecter :

$$\begin{cases} translation \quad T : T \times N1' = 0 \\ rotation \quad R = (\ \alpha,\ P2,\ A\ )\ avec\ \alpha\ et\ A\ quelconques. \end{cases}$$

• **Relation "PLAN-DROITE" : (F1 SUR D2)**

Ce cas est identique à un cas "droite-plan". La prochaine transformation devra respecter :

$$\begin{cases} \textit{translation} \quad T \ : \ T \times N1' = \ 0 \\ \textit{rotation} \quad R \ = (\ \alpha,\ X1',\ \beta \cdot N1'\ ) \ \textit{avec } X1' \textit{ un point de } F2' \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{et } \alpha \textit{ et } \beta \textit{ quelconques} \\ \qquad\qquad \textit{ou} \\ \qquad\qquad R \ = (\ \alpha,\ X1,\ V2\ ) \ \textit{avec } \alpha \textit{ quelconque} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{et } X2 \textit{ un point de } D2. \end{cases}$$

• **Relation "PLAN-PLAN" : (F1 SUR F2)**

Les translations sont permises sur le plan F2, les rotations autour d'un axe parallèle à la normale de F2. La prochaine transformation devra respecter :

$$\begin{cases} \textit{translation} \quad T \ : \ T \times N2 = \ 0 \\ \textit{rotation} \quad R \ = (\ \alpha,\ X,\ \beta \cdot N2\ ) \ \textit{avec } X, \ \alpha \textit{ et } \beta \textit{ quelconques.} \end{cases}$$

3.6.3 Combinaison des contraintes

Notations :

- CV_i : les translations sont permises le long d'une droite de vecteur directeur V_i
- CN_i : les translations sont permises sur un plan de vecteur normal N_i
- $\emptyset T$: indique que toute translation est interdite.
- RP_i : les rotations sont possibles uniquement autour du point P_i
- RV_i : les rotations sont possibles uniquement autour d'un axe dont le vecteur directeur est parallèle à V_i
- RA_i : les rotations sont possibles uniquement autour d'une droite dont un point est PA_i et de vecteur directeur VA_i. $RA_i = (PA_i\ ,\ VA_i)$.
- $\emptyset R$: indique que toute rotation est interdite.

Table de combinaison des contraintes sur les translations :

Ancienne-contrainte (i)	Contrainte imposée par la dernière relation (j)	Résultat de la combinaison (k)
Aucune	Quelconque	(j)
CV_i	CV_j	si $V_i \wedge V_j = O$ alors V_i sinon, $\emptyset T$
	CN_j	si $V_i \times N_j = 0$ alors V_i sinon, $\emptyset T$
	$\emptyset T$	$\emptyset T$
CN_i	CV_j	si $N_i \times V_j = 0$ alors N_i sinon, $\emptyset T$
	CN_j	si $N_i \wedge N_j = 0$ alors N_i sinon, $\emptyset T$
	$\emptyset T$	$\emptyset T$
$\emptyset T$	quelconque	$\emptyset T$

Table de combinaison des contraintes sur les rotations :

Ancienne-contrainte (i)	Contrainte imposée par la dernière relation (j)	Résultat de la combinaison (k)
Aucune	Quelconque	(j)
RP_i	RP_j	si P_i=P_j alors RP_i sinon, $\emptyset R$
	RV_j	$RA_k = (P_i , V_j)$
	RA_j	si P_i=P_j alors $RA_k = (P_i , V_j)$ sinon, $\emptyset R$
	$\emptyset R$	$\emptyset R$
RV_i	RP_j	$RA_k = (P_j , V_i)$
	RV_j	si $V \times V_j = 0$ alors RV_i sinon $\emptyset R$
	RA_j	si $V \times V_j = 0$ alors RA_i sinon $\emptyset R$
	$\emptyset R$	$\emptyset R$
RA_i	RP_j	si P_i=P_j alors RA_i sinon $\emptyset R$
	RV_j	si $V_i \times V_j = 0$ alors RV_i sinon $\emptyset R$
	RA_j	si $V_i \times V_j = 0$ et P_i=P_j alors RA_i sinon $\emptyset R$
	$\emptyset R$	$\emptyset R$
$\emptyset R$	quelconque	$\emptyset R$

Les contraintes ainsi obtenues sont utilisées comme suit par le système :

1. Les translations sont impossibles lorsque la combinaison de contraintes donne $\emptyset T$. Les rotations sont impossibles lorsque la combinaison de contraintes donne $\emptyset R$. Quand nous obtenons $\emptyset T$ et $\emptyset R$ alors tous les degrés de liberté du manipulateur sont blocqués, tout mouvement est désormais interdit. La situation est réalisée uniquement si les relations non traitées de la situation sont vérifiées, sinon la situation n'est pas "résolvable" par cette méthode.

2. L'ensemble des translations possibles est non vide : si une translation est nécessaire pour la prochaine relation à réaliser, sa direction sera soit v_k soit perpendiculaire à n_k.

3. L'ensemble des rotations possibles est non vide :
   . Un point origine des rotations est imposé, dans ce cas toutes les rotations doivent se faire autour de ce point, sinon c'est l'échec.
   . La direction de l'axe de rotation est imposée, les rotations suivantes doivent se faire autout d'un axe dont le vecteur est parallèle à celui précisé dans les contraintes.
   Dans le cas de rotations, ces contraintes servent uniquement après un calcul à vérifier si la transformation obtenue est conforme, elles n'interviennent pas dans le calcul même.

3.6.4 Avantages et limites de la méthode

Cette méthode permet de traiter les cas ambigus, impossibles à traiter par la méthode de LM-GEO (Cf. paragraphe 3.5).

* Une solution est proposée même si les relations introduisent une indétermination (infinité de solutions).

* Il est possible de traiter les relations symétriques des relations point-droite, point-plan et droite-plan dans une expression "mixte".

* La présence d'une relation droite-droite, plan-plan ou de deux relations point-point n'est pas obligatoire (il n'est pas nécessaire de fixer une rotation dans l'expression).

Il faut toutefois noter que dans certains cas la résolution "séquentielle" des relations échoue, même s'il existe une solution. Ceci se produit lorsque la réalisation

d'une relation point-plan, point-droite ou point-point nécessite l'emploi d'une rotation, ce que le système ne sait pas faire.

Considérons par exemple un objet A dont les points extrêmes d'une arête sont PA1 et PA2 et un objet B dont deux arêtes sont DB1 et DB2. Nous nous proposons de réaliser la situation géométrique ((PA1 SUR DB1) (Pa2 SUR DB2)). Une solution consiste à projeter le point PA1 sur la droite DB1, puis à faire une rotation α autour d'un axe d'origine PA1 déplacé et perpendiculaire à DB1. Le système échoue car il propose une translation permettant d'amener PA1 sur DB1, mais ensuite aucune translation ne permet de mettre PA2 sur DB2 sans déplacer PA1.

Ceci vient du mécanisme de combinaison des contraintes qui est trop élémentaire (séparation translations et rotations en particulier), et du fait que l'on utilise la contrainte de déplacement uniquement en translation pour les relations point-point, point-droite et point-plan.

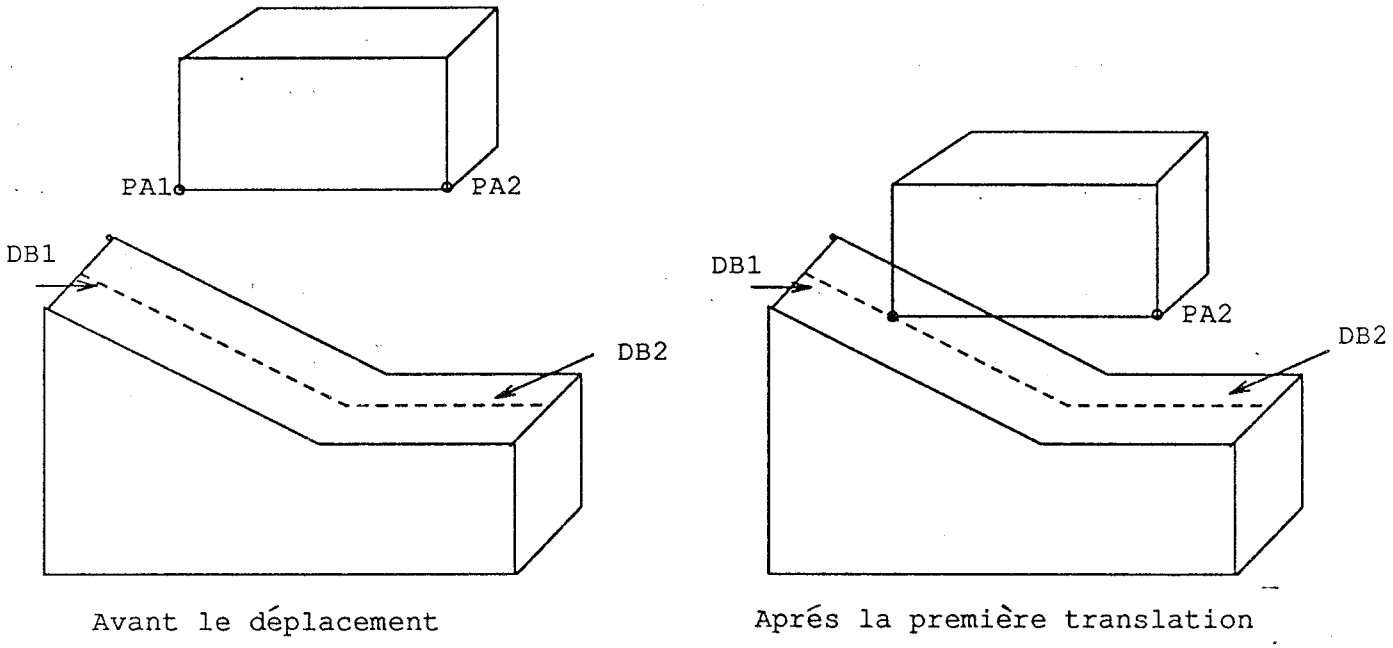


Figure 3.4: Exemple de cas d'échec

Cette restriction peut parfois être génante, aussi nous allons exposer dans le paragraphe suivant la solution retenue permettant de remédier à ces inconvénients et utilisable dans tous les cas.

3.7 Méthode appliquée dans LM-OBJ

La méthode que nous avons implanté dans LM-OBJ combine trois méthodes présentées précédemment :

1. La méthode de résolution pour une relation isolée est appliquée dans le cas où la situation ne comporte qu'une seule relation. Elle donne toujours une solution.
2. Dans le cas de n relations, la méthode de résolution de LM-GEO est employée, si les conditions d'utilisation sont vérifiées (Cf. paragraphe 3.2). Si une solution unique existe, cette solution sera alors trouvée par le système.
3. La méthode de "résolution séquentielle" est appelée en dernier lieu, lorsque la situation est hors du champ d'action de LM-GEO, ou lorsque LM-GEO n'a pas trouvé de solution à cause d'un système d'équations sous-contraint (infinité de solutions). Cette méthode est alors combinée avec la première méthode (résolution dans le cas d'une relation isolée)

Nous aurions pu utiliser uniquement une combinaison des méthodes 1 et 3. Cependant, il nous a semblé préférable de ne faire appel à la méthode de résolution "séquentielle" des relations que si la méthode de résolution de LM-GEO n'est pas utilisable ou ne donne pas de résultats. En effet, dans LM-GEO lorsqu'il existe une solution le système la trouve, ce qui n'est parfois pas le cas dans la méthode 3 (voir exemple précédent).

Chapitre 4

REALISATION TECHNIQUE

4.1 Implantation matérielle et logicielle

4.1.1 Description du matériel utilisé

Le robot : c'est un robot fixe de marque SCEMI à six degrés de liberté (six articulations rotoïdes). L'outil terminal est une pince à deux mors en "tout ou rien" (position ouverte ou fermée sans précision de l'écartement des mors).

L'armoire de commande : elle permet d'actionner le bras manipulateur par le contrôle de la rotation sur chaque axe, de la vitesse des moteurs, ...

Le calculateur HP-1000 : il pilote le robot par l'intermédiaire de l'armoire de commande. Les programmes permettant ce contrôle sont écrits en langage LM.

Le capteur de forces : il est utilisé pour mesurer les moments et les forces exercées au niveau du poignet du robot où il est fixé. Une carte 68000 effectue les calculs et dialogue avec le hp-1000 pour fournir les valeurs demandées. Le seul capteur utilisé est ce capteur de forces placé au niveau du poignet du robot.

Les valeurs fournies pas le capteur dans le repère robot sont les suivantes :
Fx : force mesurée selon l'axe X.
Fy : force mesurée selon l'axe Y.
Fz : force mesurée selon l'axe Z.
Mx : moment détecté autour de l'axe X.
My : moment détecté autour de l'axe Y.
Mz : moment détecté autour de l'axe Z.
Pour mesurer ces valeurs au niveau d'un objet tenu dans la pince du robot, on fait

l'hypothèse selon laquelle la liaison pince-objet est rigide ce qui permet d'affirmer que ces valeurs sont les mêmes que celles lues au niveau du poignet. Ceci est une approximation car en fait la liaison n'est pas rigide : présence de caoutchouc sur les mors de la pince.

Station de travail SUN : Tous les modules de SHARP et de l'interpréteur de LM-OBJ sont programmés en LISP sur une machine SUN.

4.1.2 Architecture matérielle

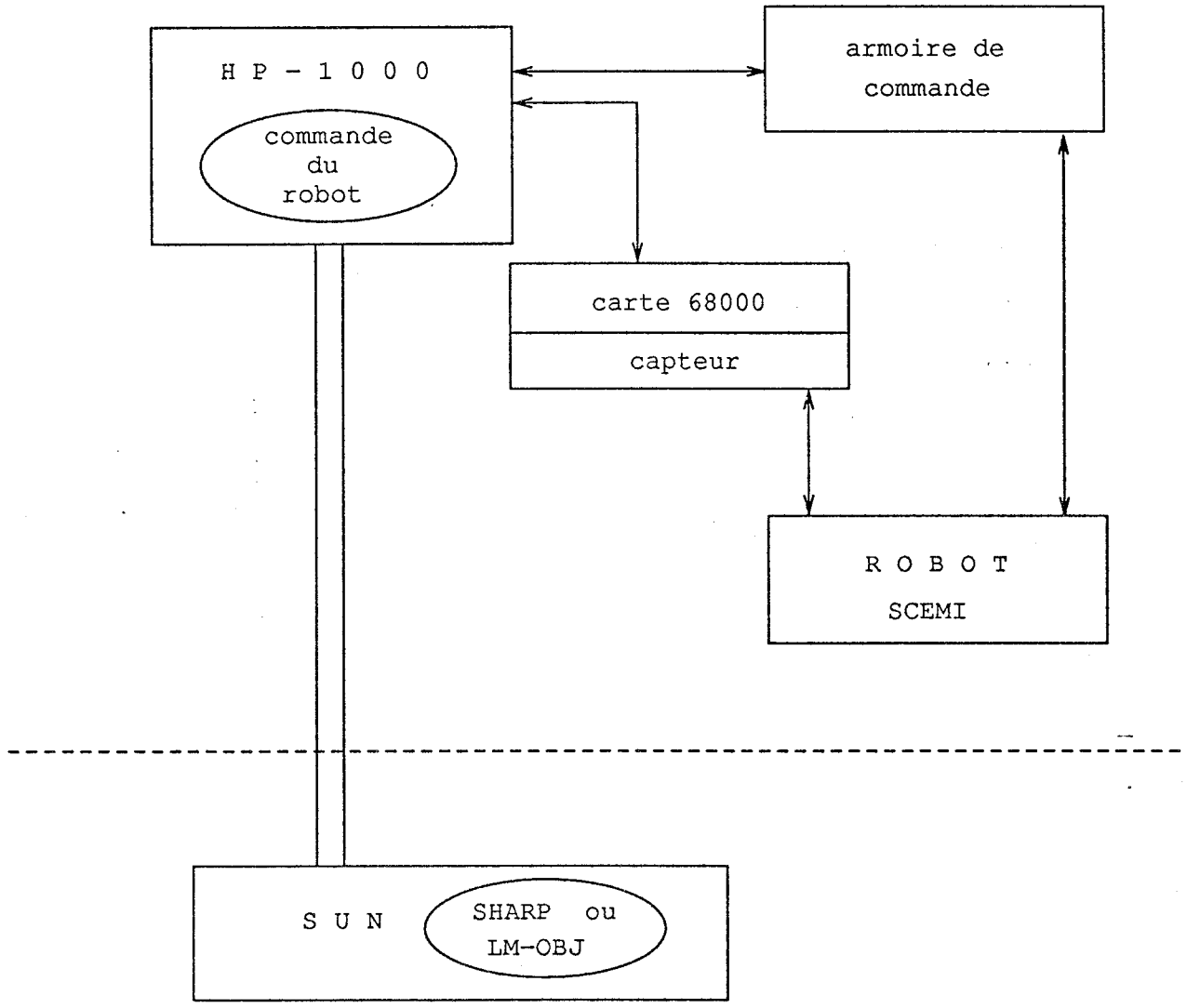


Figure 4.1: Architecture matérielle

4.1.3 Présentation du logiciel utilisé

Le langage de programmation employé est COMMON-LISP.

Le langage développé est en fait un interpréteur dont les instructions sont des fonctions écrites en LISP, ce qui permet de supprimer les phases d'analyse sémeatique et lexicale. Les instructions sont traduites en termes de repères cartésiens et de transformations géométriques sur le calculateur SUN qui transmet les informations (code instruction, valeur numérique des paramètres) à un programme LM installé sur HP1000 et ce, par l'intermédiaire d'une ligne série à 4800 bauds.

L'annexe C détaille la phase de connexion et d'initialisation du matériel et du logiciel.

4.1.4 Architecture logicielle

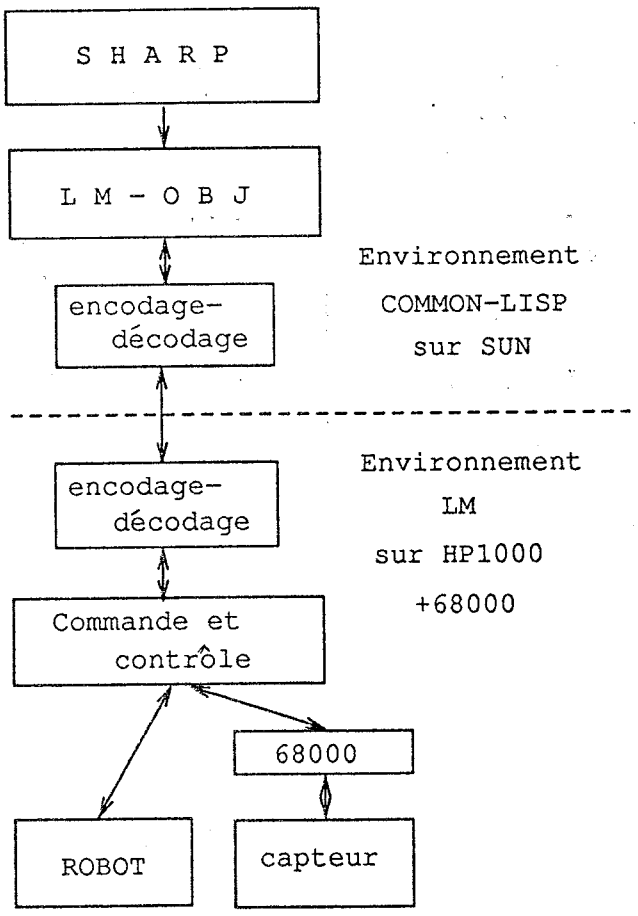


Figure 4.2: Architecture logicielle

4.2 La base de données

4.2.1 Description

Elle se compose de trois sortes de données différentes : les structures décrivant les entités géométriques, les variables globales et les constantes.

- **Les entités géométriques**

Les entités géométriques sont décrites par des structures LISP portant le même nom qu'elles : solide, composant, face, arête, point, repère. Leur description détaillée est donnée en 3.1.2.

Du point de vue implantation, ces structures portent le même nom que dans SHARP mais ne sont pas décrites de la même façon. Ceci est possible car le logiciel SHARP est divisé en différents "packages" : ensembles de fonctions ou de données utilisables sous un certain environnement. Un nouvel environnement a donc été créé, il se nomme INTERP et contient tout le source de l'interpréteur de LM-OBJ et sa base de données. Ainsi, il est possible d'avoir les mêmes noms de structures qui ne sont toutefois pas décrites de la même façon.

- **Les variables globales**

Des variables globales complètent cette structure, elles concernent :

La propagation des déplacements

éléments-déplaçables : c'est une liste contenant le nom des pièces qui font partie du bras manipulateur. A chaque déplacement toutes ces pièces bougent, il faut alors mettre à jour leur position.

objets-assemblés : c'est une liste contenant le nom des objets qui ont été assemblés et qui bougent de façon solidaire.

objets-susceptibles-de-bouger : c'est une liste contenant le nom de tous les objets qui vont bouger en même temps que le bras manipulateur et pour lesquels il faudra donc faire une mise-à-jour de la position après chaque déplacement. Cette liste contient les éléments déplaçables, les objets tenus dans la pince du robot.

objets-liés : Il s'agit de la liste des objets pris dans la pince du robot. En général c'est un objet lié à la pince du robot, toutefois, si cet objet est assemblé, cette liste

peut comporter plusieurs noms.

Variables de liaison avec le modèle géométrique de SHARP

**corr-solides* :* c'est une liste de paires dont le second élément est le nom du solide dans la base de données de LM-OBJ et le premier élément désigne ce même objet dans la base de modèles géométriques de SHARP (qui a donc servi à la création du premier).

**corr-composants* :* idem mais pour les entités de type "composant".

**corr-faces* :* idem mais pour les entités de type "face".

**corr-arêtes* :* idem mais pour les entités de type "arête".

**corr-points* :* idem mais pour les entités de type "point".

**corr-repères* :* idem mais pour les entités de type "repère".

Toutes ces variables sont de type "a-listes" c'est à dire composées de la façon suivante : ((elt-1-sharp . elt-1-interp) ... (elt-n-sharp . elt-n-interp)). Elles permettent, dans le cas d'une utilisation sous l'environnement SHARP, de savoir quel est le nom dans INTERP d'un objet ou d'une entité connue sous un certain nom dans l'environnement SHARP. En effet, les plans d'actions produits par le planificateur font référence aux entités par le nom qu'elles ont sous SHARP.

**dépendance-base-modèle* :* variable booléenne indiquant si l'interpréteur est utilisé sous l'environnement SHARP ou de façon indépendante.

**mise-à-jour-histo* :* cette variable décrit l'historique des instructions exécutées et intéressantes pour la mise-à-jour du modèle géométrique coté SHARP ; mise à jour qui se fait en fin de session sous LM-OBJ. Les instructions saisie, lâcher , assembler et les déplacements sont inscrits dans cet historique de la façon suivante :

saisie : (LIER nom-objet-1 nom-objet-saisi)
lâcher : (DELIER nom-objet-1 nom-objet-saisi)
assembler : (LIER nom-objet-1 nom-objet-assemblé)
déplacement : (DEPLACER nom-objet transformation).

Le déplacement se fait d'une façon un peu particulière : tous les mouvements ne sont pas répertoriés, tant qu'ils concernent les mêmes objets, ils sont composés afin de ne faire qu'une mise à jour en déplacement (entre le début et une saisie, entre une saisie et un lâcher, entre une saisie ou un lâcher et la fin de session). La création de *mise-à-jour-histo* se fait alors ainsi : une instruction de SAISIE

permet d'inscrire deux instructions : un DEPLACER et un LIER, de même pour une instruction d'assemblage, une instrution de LACHER engendre deux instructions de type DEPLACER et DELIER, une instruction de déplacement (REALISER-S, REALISER-C, CORRIGER, SUPPRIMER-C) n'engendre qu'une composition de transformations.

**transf-histo* :* C'est la transformation résumant le déplacement à appliquer aux objets concernés dans la base de SHARP. Il s'agit de la composition des transformations appliquées à un même objet entre deux changements d'état. Cette variable est utilisée dans *mise-à-jour-histo* lorsqu'il faut enregistrer un déplacement : (DEPLACER nom-objet *transf-histo*).

Variables concernant le calcul de déplacements

**sauve-état-courant* :* c'est la sauvegarde de la position de chacun des repères susceptibles de bouger durant le déplacement. Ceci permet une remise en état de la base en cas d'impossibilité de réaliser le déplacement alors que certains repères ont déjà été déplacés dans la base.

**repère-déplaçable* :* nom du repère attaché à l'objet concerné par le déplacement.

**nv-rep-dep* :* repère auquel sera appliqué physiquement le déplacement, il s'agit de *repère-déplaçable* éventuellement translaté en un point précis dans le cas d'une rotation.

**contraintes-mvt* :* liste décrivant les contraintes que doivent respecter les transformations géométriques permettant de réaliser une relation et ce, dans le cas où la méthode "résolution séquentielle" de relations est employée. Cette liste est composée de six élements : (*vdir vnorm p0 vect p1 vect-1*).

vdir : vecteur directeur d'une droite indiquant la direction de translation,
vnorm : vecteur normal à un plan sur lequel doivent se faire les translations. Si vdir et vnorm valent tous deux NIL, alors aucune direction de translation n'est imposée, s'ils valent tous deux T alors toute translation est interdite.

p0 : indique le point de rotation obligatoire en cas de rotation, si la valeur est NIL alors la rotation peut se faire autour de n'importe quel point, si la valeur est T alors toute rotation est interdite.
vect : donne le vecteur directeur de l'axe autour duquel doit se faire une éventuelle rotation.
p1 : point origine de l'axe de rotation de vecteur directeur vect-1.
vect-1 : donne le vecteur directeur de la droite autour de laquelle doit se faire

une éventuelle rotation (le point de rotation peut être n'importe quel point de cette droite).

**composition-transf* :* c'est une liste détaillant la transformation géométrique calculée pour une relation et ce, afin de vérifier si elle respecte les contraintes imposées. Cette liste comprend le vecteur translation, le point de rotation, l'axe de rotation et l'angle de rotation ; si une valeur est absente elle est remplacée par NIL. Cette variable est utile dans le cas d'un appel à la méthode de "résolution séquentielle des relations".

**occur-relations* :* il s'agit d'une liste de neuf éléments, dont chacun d'eux représente le nombre d'occurrences d'un type de relation rencontré dans la situation géométrique à étudier, et ce pour savoir si l'on peut utiliser l'interpréteur géométrique de LM-GEO pour la résolution des relations. L'ordre des relations est le suivant (point-point point-droite point-plan droite-point droite-droite droite-plan plan-point plan-droite plan-plan).

**sous-contraint* :* variable booléenne permettant de savoir si après un appel à l'interpréteur géométrique de LM-GEO le système présenté était sous-contraint ou non.

- Les constantes

axe-x : structure du vecteur représentant l'axe X (1.0 0.0 0.0).

axe-y : structure du vecteur représentant l'axe Y (0.0 1.0 0.0).

axe-z : structure du vecteur représentant l'axe Z (0.0 0.0 1.0).

**incertitude-position* :* c'est une variable représentant l'incertitude de position qui est pour l'instant une constante, mais qui par la suite devrait être une variable initialisée par un module de gestion des incertitudes.

4.2.2 Initialisation

Elle est différente selon l'environnement d'utilisation.

- Environnement SHARP

L'utilisateur doit donner la liste des noms des objets du robot qu'il veut voir figurer dans la base de LM-OBJ. Ceci permet d'initialiser la variable *éléments-déplaçables*.

Il doit aussi préciser la liste des solides ou des entités autres que ceux appartenant au robot et qu'il souhaite dupliquer dans la base de LM-OBJ.

Dans ces deux cas la procédure est la même. Le nom donné est :

celui d'un solide : il est entièrement dupliqué, c'est à dire que seules les informations décrites, citées en 3.1.2, sont conservées mais toute l'arborescence du solide est reconstruite : composants, faces, arêtes, points et repères.

autre que celui d'un solide : seule l'entité correspondante sera créée avec recopie du minimum d'informations. Deux autres structures, si elles n'existent pas déjà, seront construites : il s'agit d'une structure *solide* qui représente le solide d'appartenance de l'entité créée et qui comporte le nom du repère de l'objet. Puisque les valeurs numériques de l'entité sont données dans ce repère, une structure *repère* sera elle aussi construite comportant la position du solide.

Au fur et à mesure que les objets de la base de SHARP sont dupliqués sous INTERP, le champ "copie-minimale" de ces objets d'origine prend comme valeur le nom de la variable créée sous INTERP. Cela permet de savoir si une entité a été dupliquée et notamment lors de la copie d'une structure arborescente. Exemple : deux faces possèdent une arête commune, à la duplication de la première face il faut créer l'arête commune, à la recopie de la seconde face, le champ "copie-minimale" de l'arête commune permet de savoir qu'elle est déjà présente sous INTERP avec le nom contenu dans ce champ.

Lorsque tous les objets précisés par l'utilsateur ont été recopiés sous INTERP, tous les champs "copie-minimale" des entités de SHARP sont remis à NIL.

Les noms donnés aux entités construites sont obtenus de la façon suivante : le préfixe "bm-" précéde le nom du solide de référence qui est immédiatement suivi par un numéro imposé par la fonction LISP d'allocation de variables. Les autres entités sont créés à partir de ce nom auquel on rajoute un suffixe variable selon le type de l'entité : "-C-" pour les composants, "-F-" pour les faces, "-A-" pour les arêtes, "-P-" pour les points et "-R-" pour les repères. Tout ceci est suivi d'un numéro d'entité lui aussi imposé par la fonction LISP d'allocation de variables. Exemple : CUBE est le nom d'un solide sous SHARP, son nom sera BM-CUBE1 pour LM-OBJ, une arête de ce cube pourra se nommer BM-CUBE1-A-1.

La variable *dépendance-base-modèle* est positionnée à vrai, *objets-susceptibles-de-bouger* est initialisée à la valeur de *éléments-déplaçables*. Les variables de correspondance entre le nom donné sous SHARP et celui donné sous INTERP (*corr-solides*, *corr-composants*, ...) sont mis à jour au fur et à mesure de la création des entités. *transf-histo* est une transformation géométrique qui vaut au départ la transformation identité. Les autres variables de la base sont positionnées à NIL.

La duplication des informations se fait sous le contexte (ou "package") USER en direction des structures décrites dans le contexte INTERP. Si l'environnement est immédiatement changé en contexte INTERP, les structures ont toutes leurs valeurs préfixées par la chaîne "USER::". Pour éviter cette lourde manipulation, la nouvelle structure de données créée sous USER est copiée sur fichier disque pour être ensuite relue sous le contexte INTERP, ce qui permet d'avoir un "passage" par données et donc de supprimer les préfixes.

- **Utilisation en mode autonome de LM-OBJ**

Deux modes d'initialisation sont possibles :

* Rappel d'une sauvegarde effectuée lors de précédentes manipulations.
* Création d'une base de données. A l'heure actuelle, cette création n'est pas conviviale, en effet, l'utilisateur doit créer lui-même ses objets ou entités à l'aide de fonctions LISP associées aux structures déclarées et permettant ainsi de construire des variables. Exemple : "make-face" pour une structure de type "face" ou "make-arête" pour une structure de type "arête", ...

  D'autre part, il doit ranger lui-même dans *éléments-déplaçables* la liste des objets qu'il a créé et qui appartiennent à l'extrémité du bras manipulateur. Il doit forcément en exister au moins un.

4.2.3 Fin de session

Dans tous les cas, elle consiste à stopper le processus de communication entre les deux calculateurs.

Si l'interpréteur de LM-OBJ a été appelé sous l'environnement SHARP il faut alors :

* Ranger la dernière instruction de déplacement dans la liste *mise-à-jour-histo*. Elle représente tous les mouvements qui ont pu avoir lieu depuis la dernière instruction DEPLACER insérée dans la liste.

* Mettre à jour la base de modèles géométriques de SHARP à l'aide du contenu de *mise-à-jour-histo*.

Maintien de la cohérence de la base de modèles géométriques de SHARP

Le contenu de *mise-à-jour-histo* est écrit sur un fichier disque temporaire. En effet, pour travailler sur la base de SHARP il faut changer de "package" et retourner sous le contexte USER. A ce moment là, chacun des éléments de *mise-à-jour-histo* est indexé par le nom de contexte où la variable à été construite (par exemple : "INTERP::"). Or, le contenu de cette variable désigne des fonctions et des objets de l'environnement SHARP. Une écriture sur fichier sous INTERP, puis une relecture sous USER permettent de passer les informations en tant que données indépendantes du contexte (USER ou INTERP).

Une fois le contenu de *mise-à-jour-histo* récupéré correctement sous le contexte USER, il suffit d'exécuter telles quelles les instructions LIER, DELIER et DEPLACER, puisqu'elles ont été stockées en respectant le nom et les paramètres de la fonction à appeler.

LIER et DELIER mettent à jour l'ensemble des objets qui sont solidaires à un moment donné. Cela permet aux démons activés par la fonction DEPLACER de mettre à jour la position de tous les objets solidaires, et pas seulement celle de l'objet dont le nom est précisé dans l'instruction.

4.2.4 Accès aux informations de la base

Dans le cas de l'utilisation de LM-OBJ hors de l'environnement SHARP, l'accès aux données de la base se fait normalement par le nom sous lequel elles ont été créées.

En utilisation sous l'environnement SHARP, les noms des objets et des entités de la base de SHARP ne représentent rien dans le contexte INTERP. Pourtant, les plans d'actions produits ne font référence qu'aux noms connus sous SHARP. Pour identifier la variable sous INTERP, il suffit de rechercher dans les listes de correspondance : *corr-solides*, *corr-composants* , ... où est rangé ce nom. Une fois

trouvé, il fournit automatiquement le nom sous INTERP (puisqu'ils appartiennent à la même paire).

Une fonction a donc été écrite afin de rendre transparent ce problème de la provenance de la variable ; son algorithme est le suivant :

fonction *typ-corres-entité-modèle (nom-SHARP)*
- recherche du nom dans *corr-faces*,
si trouvé alors retour du second élément de la paire et du type "face" et FIN.
- recherche du nom dans *corr-arêtes*,
si trouvé alors retour du second élément de la paire et du type "arête" et FIN.
- recherche du nom dans *corr-points*,
si trouvé alors retour du second élément de la paire et du type "point" et FIN.
- recherche du nom dans *corr-repères*,
si trouvé alors retour du second élément de la paire et du type "repère" et FIN.
- recherche du nom dans *corr-solides*,
si trouvé alors retour du second élément de la paire et du type "solide" et FIN.
- recherche du nom dans *corr-composants*,
si trouvé alors retour du second élément de la paire et du type "composant" et FIN.
- sinon, c'est le nom lui-même qui est à retourner car il correspond au nom sous lequel la variable est connue dans le contexte INTERP.

fin fonction.

Les instructions rangées dans *mise-à-jour-histo* font référence aux noms des objets tels qu'ils sont connus sous SHARP, puisque ces ordres sont destinés à être utilisés sous le contexte USER. La connaissance d'un nom sous SHARP à partir d'un nom sous INTERP est directe : les structures solide, composant, face, arête, point et repère contiennent un champ nommé "modèle" et qui n'est autre que le nom correspondant dans la base de modèles géométriques de SHARP.

4.3 Les mouvements fins

Les mouvements fins sont réalisés à l'aide d'un capteur de forces placé au poignet du robot. Dans un premier temps, nous allons décrire les fonctions LM utilisant le capteur que nous avons dû écrire afin de réaliser plusieurs types de contacts. Nous étudierons ensuite comment les paramètres de ces fonctions sont calculés du côté LISP.

4.3.1 Les fonctions LM de contact

Nous distinguons trois types de contacts :

- "point-plan" ou "plan-point",
- "droite-plan" ou "plan-droite",
- "plan-plan".

Les autres relations telles que "point-point", "point-droite", "droite-point" ou "droite-droite" ne conduisent pas à un contact "réaliste". En effet, à cause des incertitudes il est impossible de réaliser de tels contacts.

• **Contact "point-plan"**

Il est réalisé par la fonction CORF développée par C. GANDON, son algorithme est le suivant :

si seuil > 0 alors
    S := seuil + *eps*
    depl := -0.1
sinon
    S := seuil - *eps*
    depl := 0.1
finsi
tanque abs(lecture-capteur(*numero-capteur*)) < abs(S) faire
    deplacer *rep-dep* de translation(*vect-dir* , depl)
finfaire

Nous avons jugé utile de rajouter dans cette fonction un paramètre *dis-max* qui donne la distance maximale à parcourir avant de déclarer l'échec dans le contact

recherché. En effet, à cause des incertitudes et du fait que les calculs se font sur un plan et non sur une face, il peut arriver que le point soit à coté de la face. Dans ce cas, le contact avec celle-ci ne sera jamais atteint, il faut tout de même arrêter le déplacement et retourner un constat d'échec.

La fonction à utiliser dans le cas d'un contact "point-plan" avec contrôle de la distance parcourue est la suivante :

iret := fcorf (*rep-dep, numéro-capteur, vect-dir, seuil, dis-max, eps*) ;

rep-dep = repère à déplacer.
numéro-capteur = composante du capteur à surveiller (0 = Fx, 1 = Fy , 2 = Fz).
vect-dir = vecteur suivant lequel se fait la translation, il peut prendre les trois valeurs suivantes : axe X, axe Y ou axe Z.
seuil = valeur au delà de laquelle nous considérons que le contact est établi.
> 0 : le déplacement se fait en sens inverse de la direction indiquée dans *vect-dir*,
< 0 : le déplacement se fait dans le sens de la direction indiquée.
dis-max = distance maximale (en millimètres) pouvant être parcourue pour obtenir le contact : au delà c'est l'échec.
eps = marge d'erreur sur le seuil de forces.
En sortie, *iret* vaut un si le contact a été établi ou zéro si la fonction a échoué.

Exemple : Contact "point-plan". L'appel correspondant à la réalisation du contact décrit dans la figure est le suivant : iret := fcorf (R, 2, Vz , 15.0 , 20.0 , 3.0).

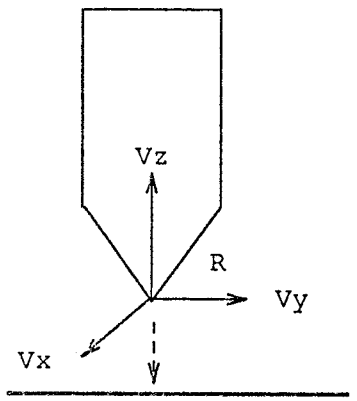


Figure 4.3: Utilisation de fcorf

• **Contact "droite-plan"**

Les fonctions de ce type utilisent à la fois des fonctions permettant d'établir un contact "point-plan" (corf ou fcorf) et une procédure permettant de minimiser les moments autour d'un axe (corm).

La méthode est la suivante :

- établir un contact "point-plan" à l'aide de la fonction FCORF.
- établir le contact "droite-plan" selon l'algorithme :

tantque abs(lecture-capteur(*numéro-capteur*)) > abs(val-limite) faire
rotation autour de *axe* de 1 degré
rétablir contact "point-plan"
finfaire

L'idéal serait d'effectuer les rotations en un repère dont l'origine serait le point de contact "point-plan". Or il n'est pas possible de le connaître dans cette fonction, aussi il est nécessaire, après chaque rotation appliquée au repère robot, de faire un mouvement de translation pour rétablir le contact avec la face.

Les fonctions LM permettant un contact "droite-plan" sont du type :

iret := drteplIJ (*translation, seuil, dis-max*)

avec I $\in$ (x y z), J $\in$ (x y z) et I $\neq$ J.

Ces fonctions permettent un contact "droite-plan" avec contact ponctuel selon la direction I, puis les rotations correctives se font autour de l'axe J.

translation = translation à appliquer au repère robot, le nouveau repère ainsi obtenu permet de faire les rotations.
Les autres paramètres (*seuil, dis-max*) sont identiques à ceux décrits pour *fcorf.*
En sortie, *iret* vaut zéro en cas d'echec et un en cas de réussite.

Exemple : L'appel correspondant à la réalisation du contact décrit dans la figure 4.4 est le suivant : iret := drteplzy (identité , 15.0 , 20.0).

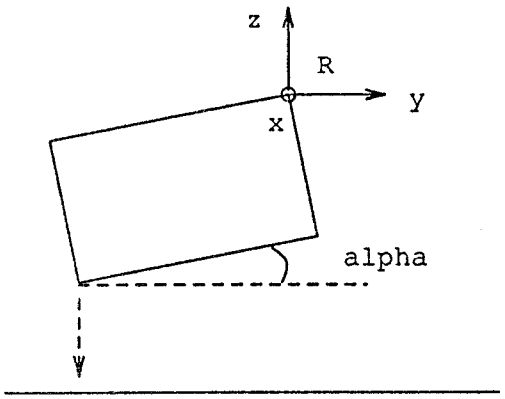


Figure 4.4: Utilisation de drteplIJ

- Contact "plan-plan"

Ces fonctions sont identiques à celles permettant un contact "droite-plan" sauf qu'elles effectuent des rotations supplémentaires autour d'un second axe qui permet de passer d'un contact "droite-plan" à un contact "plan-plan".

Trois fonctions sont disponibles :

contactx : pour un contact guidé par une translation le long de l'axe x. Dans ce cas les rotations ont lieu autour de l'axe y puis de l'axe z.

contacty : pour un contact guidé par une translation le long de l'axe y. Dans ce cas les rotations ont lieu autour de l'axe x puis de l'axe z.

contactz : pour un contact guidé par une translation le long de l'axe z. Dans ce cas les rotations ont lieu autour de l'axe x puis de l'axe y.

Problème d'implantation :

Si le capteur mesure des forces importantes, il peut par la suite ne plus fournir de valeurs justes (ordre de grandeurs différent). Cela peut arriver si une rotation n'est pas effectuée autour d'un point bien placé. Pour éviter qu'un tel inconvénient puisse nuire à l'établissement d'un contact "plan-plan" la démarche utilisée est la suivante : entre les deux rotations, il y a rupture temporaire du contact "droite-plan" établi, le capteur est alors remis à zéro, le contact rompu est rétabli. Les rotations autour du second axe peuvent alors commencer pour arriver au contact "plan-plan".

Les fonctions LM permettant un contact "plan-plan" sont du type :

iret := contactK (*translation, seuil, dis-max*)

avec K ∈ (x y z).

Ces fonctions permettent d'établir un contact "plan-plan" avec une translation selon l'axe K pour recherche du premier contact, et avec des rotations autour des deux autres axes.

Les paramètres sont identiques à ceux décrits pour les fonctions *fcorf* ou *dtreplIJ*.

Exemple : L'appel correspondant à la réalisation du contact décrit dans la figure est le suivant : iret := contactz (R , 15.0 , 20.0).

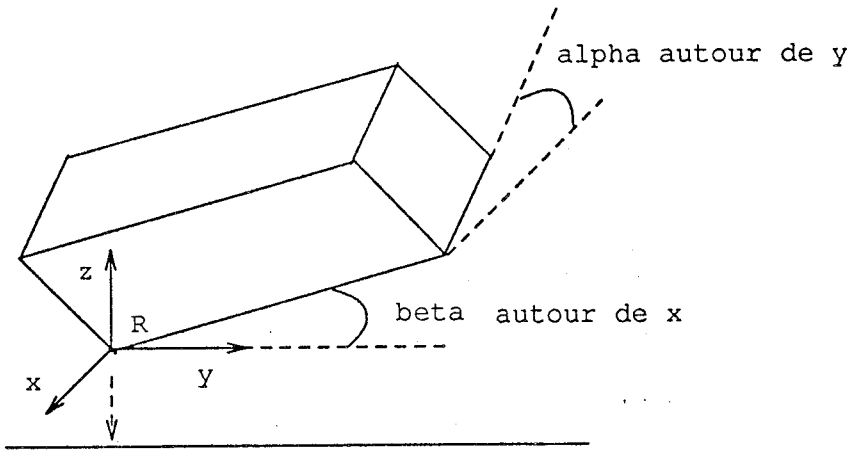


Figure 4.5: Utilisation de contactK

• Mouvements compliants

Ceux qui nous intéressent ici, sont ceux qui ont pour but d'établir un contact, tout en en maintenant d'autres précédemment établis. La démarche est la suivante :

* Rupture temporaire et légère des contacts déjà acquis et ce, à l'aide d'une translation,

* Etablissement du nouveau contact comme décrit dans les trois paragraphes précédents,

* Remise des contacts acquis auparavant par un mouvement de translation.

Remarques :

La rupture des contacts est nécessaire : elle permet le déplacement de l'objet sans forces de frictions importantes qui pourraient faire échouer l'établissement du nouveau contact.

Il faut se rappeler que ces mouvements sont fait pour pallier aux incertitudes et que de façon générale les deux objets sont presque en contact et dans la bonne position avant l'appel à ces fonctions. Aussi, les rotations et les translations sont peu importantes, un mouvement de translation sur un des axes permet toujours, si le contact est possible, de l'établir. Ce ne serait pas le cas si les deux objets étaient éloignés.

Les fonctions disponibles :

* *iret* := ctctdrpl (*numéro-capteur, seuil, dismax, nbctact, tabcapt, tabseuil, tabvap*)

  Cette fonction permet un contact "droite-plan" avec maintien éventuel d'autres contacts.

  Les paramètres *numéro-capteur, seuil, dis-max* sont identiques à ceux décrits précédemment.
  nbctact = nombre de contacts à maintenir durant l'établissement du nouveau contact. Détermine le nombre d'informations rangées dans les tables *tabcapt, tabvap, tabseuil.*
  tabcapt = table d'entiers contenant les numéros de capteurs concernés par les contacts à maintenir.
  tabseuil = table de réels contenant les valeurs des seuils de contact pour chacun des contacts à maintenir.
  tabvap = table de vecteurs contenant la direction avec laquelle les contacts précédents ont été établis.
  Un triplet (*tabcapt(i), tabseuil(i), tabvap(i)*) caractérise le ième contact à maintenir ($0 \leq i \leq$ nbctact).

* *iret* := ctctdrpl (*seuil, dis-max, irot, nbctact, tabcapt, tabseuil, tabvap*)

  Cette fonction permet un contact "droite-plan" avec maintien éventuel d'autres contacts.

irot = indique de quel contact "droite-plan" il s'agit
= 1 : translation sur X, rotation autour de Y.
= 2 : translation sur X, rotation autour de Z.
= 3 : translation sur Y, rotation autour de X.
= 4 : translation sur Y, rotation autour de Z.
= 5 : translation sur Z, rotation autour de X.
= 6 : translation sur Z, rotation autour de Y.

Les autres paramètres sont identiques à ceux décrits précédemment.

* *iret* := ctctplpl (*numéro-capteur, seuil, dis-max, nbctact, tabcapt, tabseuil, tabvap*)

Permet un contact "plan-plan" avec maintien éventuel d'autres contacts.

La description des paramètres est identique à celle donnée précédemment.

4.3.2 Au niveau LISP : recherche des paramètres

• **La méthode**

Plusieurs cas sont à considérer en fonction des paramètres fournis à l'instruction REALISER-C.

o Dans le cas général nous procédons de la façon suivante :

1. Calcul de la transformation géométrique permettant d'amener au contact les deux objets, exactement comme dans le cas de réaliser-s et d'une situation-géométrique. Cette transformation TR1 sert pour la mise à jour de la base de données.
2. Extraction de la situation de contact, d'une relation de type "point-plan", "plan-point", "droite-plan", "plan-droite" ou "plan-plan".
3. Calcul de la translation permettant d'éloigner l'objet le long de la composante principale de la normale du plan de contact et ce, d'une valeur égale à deux fois l'incertitude de position afin d'éviter toute collision. Nous obtenons ainsi une transformation TR2 permettant d'éloigner légèrement l'objet mobile de la position de contact.
4. Composition des transformations TR1 et TR2 afin d'obtenir le mouvement à faire exécuter par le robot pour se trouver proche de la situation de contact.

5. Calcul des paramètres du contact selon le type de contact (dans la situation de proximité entre les deux objets) afin de les transmettre à la fonction LM correspondante.

o Au cas où REALISER-C comporte un paramètre de direction :
La méthode est différente dans le sens où la direction de déplacement est imposée par le paramètre <direction>. Mais toutes les étapes citées précédemment sont identiques.

o Cas où l'instruction comporte un paramètre de maintien de contacts :
Dans tous les cas, si un paramètre <maintenir-c> est précisé, il faut extraire toutes les caractéristiques de ces contacts, et ce de la même façon que s'il s'agissait d'un contact à établir.

D'autre part, si un tel paramètre est précisé alors la méthode générale est modifiée. La transformation TR1 est toujours calculée de la même façon afin de mettre à jour la base (après la réalisation physique du contact). Mais les étapes suivantes sont modifiées : nous calculons directement les paramètres de contacts et faisons appel aux fonctions LM correspondantes sans passer auparavant par un REALISER-S. Ceci s'explique par le fait qu'au premier appel d'un REALISER-C les deux objets peuvent ne pas être à proximité, or il faut absolument pour que la réalisation des contacts soit correct qu'ils soient dans un position de "presque-contact". La situation de contact peut alors avoir des relations géométriques précisant la position d'approche des deux objets. Une fois que le contact est réalisé et que nous souhaitons en réaliser d'autres tout en manitenant celui-ci, les deux objets sont en contact, il ne s'agit plus d'exécuter des grands déplacements avec REALISER-S. Dans ce cas la situation de contact se réduit à une relation de contact entre un plan et une autre entité et le déplacement se fait uniquement à l'aide de petits mouvements et des capteurs.

Remarque :

L'éloignement se fait le long de la composante principale de la normale du plan de contact, car enuite le déplacement LM se fait dans la direction d'un des trois axes X, Y ou Z. Nous pouvons nous permettre d'éloigner l'objet et de retrouver le contact, le long de la composante essentielle de la normale car l'objet mobile est très près du contact avec l'objet fixe et dans une orientation presque parfaite (angles de rotation s'ils existent très faibles).

• **Les paramètres de contact**

contact "point-plan"

* La direction de translation est donnée par la valeur absolue de la composante principale de la normale du plan.

* Le seuil est une valeur fixe en absolu, toutefois son signe détermine le sens du déplacement. Si la composante principale de la normale est positive le seuil sera positif, sinon il sera négatif.

* Le numéro de capteur découle de la direction de translation (axe-x : capteur FX, axe y : capteur Fy, axe z : capteur Fz).

* Calcul de la distance maximale à parcourir avant arrêt pour échec : distance euclidienne entre le point et sa projection dans la direction de translation donnée. Cette valeur est doublée pour être certain de ne pas arrêter alors qu'un contact est possible.

contact "droite-plan"

* Les paramètres direction de translation, seuil, numéro de capteur, distance maximale sont calculés de la même façon que dans un cas "point-plan".

* Le paramètre irot est calculé de la façon suivante :
  – translation sur X : si la composante essentielle du vecteur directeur de la droite est l'axe Z alors les rotations se font autour de l'axe Y (irot=1) sinon elles se font autour de l'axe Z (irot=2).
  – translation sur Y : si la composante essentielle du vecteur directeur de la droite est l'axe Z alors les rotations se font autour de l'axe X (irot=3) sinon elles se font autour de l'axe Z (irot=4).
  – translation sur Z : si la composante essentielle du vecteur directeur de la droite est l'axe Y alors les rotations se font autour de l'axe X (irot=5) sinon elles se font autour de l'axe Y (irot=6).

contact "plan-plan"

Les paramètres sont calculés de la même façon que précédemment.

Le repère de rotation est translaté sur la face de l'objet mobile à mettre au contact. Il faut calculer pour cela la projection de l'origine du repère robot sur la face de contact de l'objet mobile.

maintien de contacts existants

Quels que soient les types de relation de contact à maintenir, il suffit de fournir les seuils de contact, la direction et le numéro du capteur à surveiller, car la rupture et le rétablissement des contacts ne se fera que par translations. Les calculs sont identiques à ceux décrits précédemment.

4.4 Les mouvements de transfert

Le calcul de la transformation permettant de réaliser un mouvement de transfert à partir d'une situation géométrique a été détaillé au chapitre précédent.

Cependant pour des raisons de contraintes physiques sur le robot, il n'est pas toujours possible d'appliquer les rotations en un point précis. Par exemple, nous devons mettre deux plans l'un contre l'autre à l'aide d'une faible rotation (quelques degrés). Le point d'intersection entre les deux plans peut se trouver à "l'infini" et dans ce cas le déplacement ne se fait pas sous LM.

Pour remédier à cet inconvénient, les rotations sont centrées en un point de l'objet, celui le plus proche du point d'intersection calculé. Il est alors nécessaire de calculer la translation permettant de mettre ces deux plans l'un sur l'autre, car si la rotation n'est pas appliquée au point d'intersection elle rend seulement les deux plans parallèles. Cette translation doit alors être composée avec la rotation calculée précédemment afin d'obtenir la transformation permettant de réaliser complètement la situation.

Donc, partout où le point d'intersection entre deux entités virtuelles (plans porteurs, droites) était utilisé il faut employer un point extrême d'un objet, y compris dans les contraintes.

Remarque générale

Dans tous les déplacements, l'instruction de mouvement (déplacement du robot ou action de l'outil) suivante ne pourra être exécutée qu'une fois l'opération en cours terminée, et à condition que celle-ci se soit déroulée correctement.

4.5 Quelques caractéristiques du logiciel

Ce paragraphe a pour objet de fournir des précisions sur la taille du logiciel écrit, ainsi que quelques exemples de temps d'exécution afin de donner une idée des performances.

4.5.1 La taille du code

Comme cela a été décrit précédemment, il faut distinguer deux parties dans le logiciel. Sous l'environnement SUN et COMMON-LISP, le programme se charge d'interpréter les instructions et les situations géométriques ou de contact, et ce afin de calculer les paramètres nécessaires au déplacement ou à l'action sur l'outil terminal du robot. D'autre part, sous l'environnement HP-1000 et LM, les procédures reçoivent les ordres du programme LISP et permettent la commande du robot (déplacements, saisies, lâchers) et du capteur de forces.

La partie du logiciel écrite en COMMON-LISP représente 10000 lignes de code, dont 25 % de commentaires, le tout réparti en 300 fonctions. L'ensemble occupe 300 Koctets sur disque.

La partie du logiciel écrite en LM représente 2000 lignes de code, dont 50 % de commentaires, le tout réparti en 23 fonctions.

4.5.2 Les temps d'exécution

Le temps d'exécution d'une instruction de déplacement dépend de nombreux paramètres tels que :

* Le nombre de relations géométriques ou de contacts dans une situation.
* Le fait que le déplacement nécessite ou pas l'utilisation d'une rotation.
* La proximité ou l'éloignement de la situation à atteindre par rapport à la situation courante ....

Dans le paragraphe suivant, seul le temp d'exécution correspondant à la partie interprétation des instructions en LISP est donné.

Un exemple

Nous présentons dans le tableau ci-dessous les temps d'exécution relevé lors de la réalisation de la seconde version de l'exemple donné en annexe-A :

étape	temps cpu (en millisecondes)
A	380
B	500
C	4400
D	1800
E	1600
F	1500
G	800

Chapitre 5

Conclusion

Ce travail a permis de réaliser l'interpréteur du langage cible de SHARP permettant ainsi d'exécuter physiquement les plans d'actions produits par le planificateur. Les tests ont été effectués sur des objets polyhédriques en métal (voir annexe A). Les objectifs que nous nous étions fixés ont été atteints. L'option GARDE n'a toutefois pas été implanté du fait de la difficulté d'utiliser le capteur dans un mouvement de déplacement.

Le grand intérêt de ce langage est qu'il permet de faire exécuter par le robot, les plans d'actions fournis par SHARP, ce qui n'etait pas possible jusqu'à maintenant. Il offre aussi la possibilité de réaliser des montages par une programmation où l'emploi de capteurs de forces est transparent.

Ce logiciel est utilisé au LIFIA dans le cadre de plusieurs projets :

* Une équipe doit insérer dans ce langage des fonctions permettant de vérifier par des capteurs de vision (caméras) le bon déroulement du montage. Si un problème est détecté en cours d'exécution, l'emploi d'une instruction de type CORRIGER-S permet d'y remédier.
* Il sert de base pour le développement d'un logiciel de vérification de contraintes sur les incertitudes de manipulation et de position.
* Il doit être amélioré par l'utilisation non plus de LM sur un calculateur HP-1000 mais d'un système de commande "CESAR et CLEOPATRE" sur PC.

Deux développements futurs sont à envisager au niveau de l'interpréteur de LM-OBJ :

* Un premier développement nécessaire serait de calculer automatiquement *les seuils de contact* pour les fonctions compliantes en fonction du matériau com-

posant les objets manipulés. A l'heure actuelle, ces seuils ont des valeurs figées arbitrairement du fait que l'on manipule des objets étant tous constitués de la même matière.

* D'autre part le calculateur HP-1000 sur lequel est implanté LM va être remplacé à court terme par un système d'exploitation "CESAR et CLEOPATRE" avec *un autre langage de commande des robots.* Il sera donc nécessaire de réécrire la partie communication entre le SUN et le HP-100 qui deviendra une communication entre SUN et PC. Eventuellement d'autre paramètres seront à fournir aux fonctions compliantes qui seront alors disponibles sur le nouveau système.
  Ce nouveau moyen de commande du robot devrait permettre, par l'utilisation de processeurs parallèles, de disposer de mouvements compliants plus facilement utilisables (en particulier pour une clause GARDE qui n'est pas utilisable actuellement). Les temps d'exécution de ce que nous avons appelé la seconde phase (commande du robot) devraient être réduits, et ce d'autant plus que "CESAR et CLEOPATRE" pourrait être disponible sur SUN.

D'autres développements concernant l'environnement d'utilisation de LM-OBJ sont à envisager afin de rendre le système beaucoup plus convivial lorsqu'il est utilisé hors de l'environnement SHARP :

* La partie *description de la géométrie des objets* n'a pas été implantée, nous avons préféré nous attacher à la partie langage proprement dit. A l'heure actuelle, cette description doit être faite de façon "manuelle", à l'aide de fonctions LISP auxquelles on fournit tous les paramètres numériques nécessaires. On peut aussi décrire les objets en utilisant l'environnement SHARP, initialiser ensuite l'interpréteur de LM-OBJ avec ces objets et en faire une sauvegarde. Celle-ci peut alors servir de bibliothèque d'objets que l'on pourra réutiliser sous LM-OBJ hors de l'environnement SHARP.

* Il pourrait être utile de disposer d'un *module de simulation graphique* permettant d'exécuter les instructions non pas physiquement sur le robot mais de manière visuelle sur l'écran. Ceci permettrait de vérifier la validité d'un programme écrit en LM-OBJ avant de l'exécuter sur l'environnement réel et physique.

Annexe A

Exemple de manipulation planifiée sous SHARP

(REALISER-S (r-robot SUR r0) (VIA (c0 c1 c2 ... cn)))
(SAISIR (r-robot SUR r-prise0))
(REALISER-S (r-robot SUR r1)) *étape 1*
(REALISER-C (point-P1 SUR face-B1) *étape 2*
(SUIVANT (translation :vect vecteur-V1)))
(REALISER-C (arête-A1 SUR face-B1) *étape 3*
(SUIVANT (rotation :axe (make-axe :pt point-P1 :vect vecteur-V2)))
(MAINTENIR (point-P1 SUR face-B1)))
(REALISER-C (face-F1 SUR face-B1) *étape 4*
(SUIVANT (rotation :axe (make-axe :pt point-P1 :vect arête-A1)))
(MAINTENIR (arête-A1 SUR face-B1)))
(REALISER-C (arête-A2 SUR face-B2) *étape 5*
(SUIVANT (translation :vect vecteur-V3))
(MAINTENIR (face-F1 SUR face-B2)))
(REALISER-C (face-F2 SUR face-B2) *étape 6*
(SUIVANT (rotation :axe (make-axe :pt point-P2 :vect arête-A2)))
(MAINTENIR ((face-F1 SUR face-B1)(arête-A2 SUR face-B2))))
(REALISER-C (point-P3 SUR face-B3) *étape 7*
(SUIVANT (translation :vect arête-A1))
(MAINTENIR ((face-F1 SUR face-B1)(face-F2 SUR face-B2))))
(LACHER (r-robot SUR r3))

Voir le schéma des étapes page suivante.

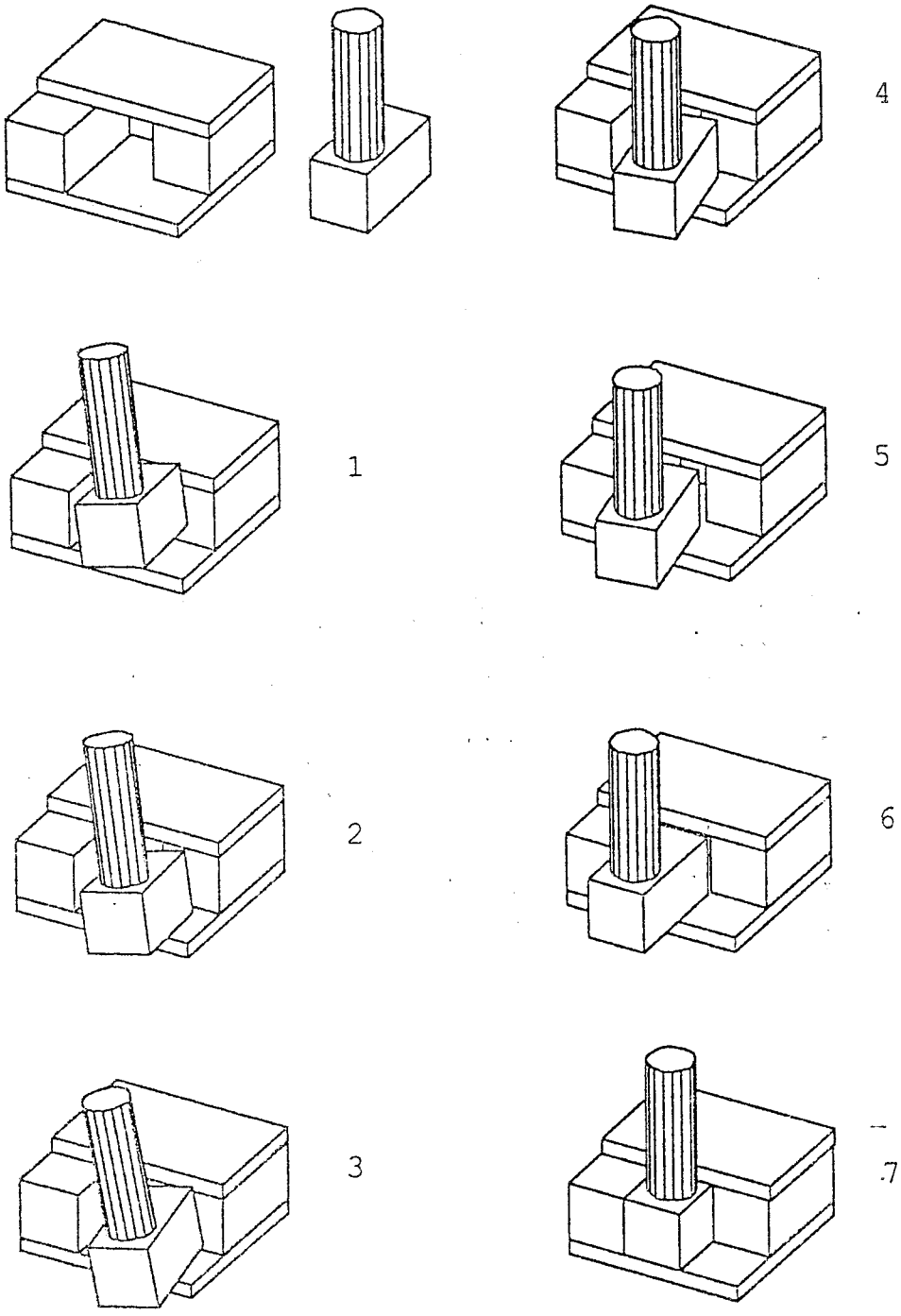


Figure A.1: Montage réalisé

L'exemple ci dessus a été planifié sous SHARP de façon automatique. Grace aux possibilités du langage, ce même exemple aurait pu être écrit plus simplement de la façon suivante :

(REALISER-S (r-robot SUR r0) (VIA (c0 c1 c2 ... cn))) *étape A*
(SAISIR (r-robot SUR r-prise0)) *étape B*
(REALISER-S (r-robot SUR r1)) *étape C*
(REALISER-C (face-F1 SUR face-B1)) *étape D*
(REALISER-C (face-F2 SUR face-B2) *étape E*
(MAINTENIR (face-F1 SUR face-B1)))
(REALISER-C (point-P3 SUR face-B3) *étape F*
(MAINTENIR ((face-F1 SUR face-B1)(face-F2 SUR face-B2))))
(LACHER (r-robot SUR r3)) *étape G*

Annexe B

Syntaxe du langage

Les instructions du langages

(ASSEMBLER <liste-objets>)

(CORRIGER <situation-géométrique> (<direction>) (<amplitude>))

(FIN-GEO)

(INIT-GEO)

(LACHER <situation> { <action> })

(REALISER-C <contact> { <direction> } { <maintenir-c> } {<effort>})

(REALISER-S <situation> { <chemin> } { <garde> })

(RESTAURATION-BASE)

(SAISIR <situation> { nom-d'objet } { <action> })

(SAUVEGARDE-BASE)

(SUPPRIMER-C <contact> { <direction> } { <maintenir-c> })

(VRF-SITUATION <situation>)

Les paramètres de ces instructions

<situation> ::= <situation-géométrique> |
(REPÈRE SUR REPÈRE) |
<coordonnées-articulaires> |
(REPÈRE TRANSFORMATION)

<situation-géométrique> ::= (<relation-géométrique>$^{+}$)

<relation-géométrique> ::= (AXE SUR AXE) |
(<entité-géo-virtuelle> SUR <entité-géo-virtuelle>) |

(<entité-géo-virtuelle> SUR <entité-géo-réelle>) |
(<entité-géo-réelle> SUR <entité-géo-virtuelle>)

<coordonnées-articulaires> ::= (θ_1 θ_2 θ_3 θ_4 θ_5 θ_6)
avec θ_i angle sur l'axe I du robot à six degrés de liberté.

<entité-géo-réelle> ::= NOM-DE-FACE | NOM-D'ARÊTE | NOM-DE-SOMMET

<entité-géo-virtuelle> ::= PLAN | DROITE | POINT

<action> ::= OUVRIR | FERMER

<contact> ::= <relation-géo.-contact> $^+$ { <relation-géométrique> $^+$ }

<relation-géo.-contact> ::= (<entité-géo-réelle> SUR <entité-géo-réelle>)

<direction> ::= (SUIVANT <d>)

<amplitude> ::= (AMPLITUDE <a>)

noindent <maintenir-c> ::= (MAINTENIR <contact>$^+$)

<effort> ::= (EFFORT <condition>)

<condition> ::= (< cond-i > $^+$)

<cond-i> ::= (X > A) avec X = Fx, Fy, Fz, Mx, My OU Mz; et à un réel exprimant une force.

<garde> ::= (GARDE <liste-contacts>)

<liste-contacts> ::= (<relation-géo.-contact> $^+$)

<chemin> ::= (VIA <situation>$^+$)

<liste-objets> ::= (NOM D'OBJETS $^+$)

<d> ::= (Vx Vy Vz) vecteur orienté indiquant une direction de translation |
((Px Py Pz) (Vz Vy Vz)) axe de rotation composé d'un point et d'un vecteur.

<a> ::= valeur réelle indiquant le déplacement en millimètres (translation) ou en radians (rotation)

Note :

liste ::= (<objet>$^+$) indique qu'il peut y avoir une ou plusieurs occurence de <objet> dans la liste.
Le symbole { indique que le paramètre est facultatif.

Annexe C

Connexion, initialisation

C.1 Mise en marche du robot

* Les boutons avant et arrière de l'armoire de commande doivent être positionnés sur MARCHE.

* Eventuellement il faut démarrer le HP1000.

* Entrer sur la machine système du HP1000 avec bifurcation sous l'espace disque "GANDON". La phase d'initialisation du robot est appelée par le programme "robot" avec sélection de l'option 1 : robot SCEMI 1. Là sont chargés des fichiers de données et LM. Puis la puissance est mise sur le robot, il peut alors être contrôlé.

* La première manipulation consiste à faire exécuter un recalage au robot (à l'aide du boitier manuel). Il va en butée sur tous ses axes afin de trouver une position connue. La seconde manipulation consiste à lui faire exécuter un repli afin qu'il se trouve dans une configuration "initiale", début de toute manipulation. Le robot est prêt à être commandé par un programme LM.

C.2 Mise en marche des capteurs

* Bouton de la carte 68000 sur MARCHE.

* Le travail se fait à partir de la console connectée à la carte 68000.

* Le code d'exploitation du capteur doit être téléchargé à partir d'un utilisateur vax car il n'est pas résident en permanence sur le 68000.
  Instructions permettant ce chargement :
  > *TM*
  login vax : *gandon*

password :
vax : *cd capteur*
vax : *ctrl a*
> *LO2 ;X=ctrl u cat cap*
> *GO 1000*

* Les capteurs sont alors prêts à être utilisés (vérifier la connexion du cable gris sur le boitier capteur du robot).

C.3 Connexion SUN-HP1000

Sur SUN :

* A partir de la console maître initialiser le port b à 4800 bauds à l'aide de la commande /etc/ttyb4800.
* Connecter une console sur le port b (fiche 28 du tableau), entrer sous son compte et taper la ligne de commande :

  stty -echo cbreak nohang raw ; mesg no ; sleep 777777

  Le processus est "endormi" sur une connexion ne pouvant pas être interrompue.
* Il faut connecter la ligne venant du HP1000 (numero 6 du tableau) sur le port b "endormi" de la SUN (fiche 28) à l'aide d'un cable "croisé".
* Un processus LISP est alors appelé sur SUN, il commence l'exécution de l'interpréteur de commandes INTERP-GEO ou de SHARP. En parallèle, le programme d'interprétation LM est mis en route. INTERP-GEO peut envoyer les séquences d'initialisation logicielles de la communication (init) et (start).

C.4 Codes instructions, paramètres

Ce paragraphe décrit pour chaque code instruction, la fonction correspondante ainsi que les paramètres d'entrée nécessaires.

0 $\Rightarrow$ initialisation de la communication.
1 $\Rightarrow$ récupération des coordonnées articulaires En sortie : θ_1 θ_2 θ_3 θ_4 θ_5 θ_6 .
4 $\Rightarrow$ ouverture de la pince.

5 ⇒ fermeture de la pince.

6 ⇒ changement de la vitesse de déplacement du bras du robot. En entrée : un réel indiquant la nouvelle valeur à donner à la vitesse ($0.001 \leq vitesse \leq 1$).

8 ⇒ fin de communication : déconnexion.

9 ⇒ demande de l'état de la pince.
En sortie : soit zéro si la pince est ouverte, soit un si elle est fermée.

a ⇒ demande de la valeur du repère robot en coordonnées cartésiennes.
En sortie : une matrice de coordonnées homogènes.

b ⇒ demande d'utilisation en mode manuel.

f ⇒ Demande de la position de l'axe numéro *num-axe*. En entrée : *num-axe*, le numéro de l'axe concerne ($1 \leq num\text{-}axe \leq 6$).
En sortie la position articulaire $\theta_{num-axe}$.

g ⇒ déplacement de l'axe *num-axe* du robot à la position θ degrés.
En entrée : le numéro de l'axe concerné *num-axe* et la valeur de l'angle θ .

l ⇒ appel à la fonction permettant de réaliser un contact point-plan avec éventuellement maintien de certains contacts déjà établis. En entrée les paramètres de la fonction ctctptpl : *numéro-capteur, seuil, dismax, nbcapt, tabcapt, tabseuil, tabvap,*

m⇒ appel à la fonction permettant de réaliser un déplacement d'un repère à un autre via des positions intermédiaires. En entrée : N le nombre total de positions à atteindre et *tabtra* une table contenant les N transformations à appliquer aux N repères de *tabrep.*

n ⇒ Déplacement en mode libre du repère *rep* vers la position $rep * t$. En entrée : *rep* le repère à déplacer et t la transformation géométrique à lui appliquer.

o ⇒ Déplacement en mode cartésien du repère *rep* vers la position $rep * t$.
En entrée : *rep* le repère à déplacer et t la tranformation géométrique à lui appliquer.

p ⇒ demande de contact "plan-plan". En entrée les paramètres à fournir sont ceux de la fonction ctctplpl : *numéro-capteur, seuil, dis-max, nbctact, tabcapt, tabseuil, tabvap.*

q ⇒ demande de vérification du maintien de certains contacts. En entrée : nombre et caractéristiques de ces contacts *nbctact, tabcapt, tabvap, tabseuil.*

r ⇒ demande de contact "droite-plan". En entrée les paramètres à fournir sont ceux de la fonction ctctdrpl : *seuil, dis-max, irot, nbctact, tabcapt, tabseuil tabvap* .

Bibliographie

[Bra 53] L. Brand *"Vector and tensor analysis"* Chapman and Hall editor, 1953.

[Gan 86] C. Gandon *"Introduction de la compliance dans la programmation des robots"*, Thèse de L'Institut National Polytechnique de Grenoble, Octbre 1986.

[GS 88] A.Giraud, D.Sidobre: *"Manipulation au contact et raisonnement géométrique"*. Journées géométrie et robotique LAAS/CNRS Toulouse, Mai 1988.

[Lau 83] C.Laugier: *"Influence du raisonnement géométrique dans le choix d'une prise d'objet"*, Rapport de Recherche IMAG no 414, Décembre 83.

[Lau 87] C. Laugier: *"Raisonnement géométrique et méthodes de décision en robotique. Application à la programmation automatique des robots"*, Thèse d'Etat, INPG, Décembre 1987.

[LB 85] T.Lozano-Perez and R.A. Brooks: *"An approch to automatic robot programming"*. Technical Report, Artificial Intelligence Laboratory, Avril 1985.

[Loz 76] T.Lozano-Perez: *"The design of a mechanical assembly system"*, AI-TR-397, M.I.T., Artificial Intelligence Laboratory, Cambridge, Décembre 1976

[LP 83] C.Laugier, J.Pertin-Troccaz: *"Automatic grasping: a case study in accessibility analysis"*, International Conference on Advanced Software in Robotics, Liège, Mai 1983/ publié dans "Advanced Software in Robotics", édité par A.Danthine et M.Géradin, North Holland, 1984

[LP 85] C.Laugier, J.Pertin-Troccaz: *"S.H.A.R.P. : A system for automatic programming of manipulation robots"*, 3rd International Symposium of Robotics Research, Gouvieux (France), Octobre 1985.

[LW 77] L.I.Liberman, M.A.Wesley : *"AUTOPASS : an automatic programming system for computer controlles mechanical assembly "*. IBM journal of research and development, July 1977.

[Maz 81] E.Mazer: *"Réalisation d'un support expérimental de recherche pour le projet robotique PANDORE. Définition et implantation du langage LM"*, Thèse de 3ème cycle, Institut National Polytechnique de Grenoble, Janvier 1981.

[Maz 82] E.Mazer: *"An algorithm for computing relative positions between two objects from symbolical specifications"*, IMAG, Rapport de Recherche no 297, 1982.

[Maz 82b] E.Mazer: *"LM-GEO : Geometric programming of assembly robots"*, Rapport de recherche IMAG no 196, 1982.

[Maz 87] E. Mazer: *"HANDEY: un modèle de planification pour la programmation automatique des robots"*, Thèse d'Etat, INPG, Décembre 1987.

[Mir 84] J.F.Miribel: *"Conception et implantation d'un système de programmation de robots"*, Thèse de 3ème cycle, LIFIA/IMAG ,Grenoble, France, Octobre 1984.

[MM 85] E.Mazer, J.F.Miribel: *"Le langage LM"* Collection Techniques Avancées de l'Informatique. Janvier 1985.

[Pug 85] P.Puget: *"Problèmes de prise en compte d'incertitudes en Robotique d'assemblage"*, Rapport de DEA, LIFIA/IMAG Laboratory, Juin 1985.

[Pas 89] M.Pasquier: *"Programmation automatique des robots : Système d'évitement de collision"*, Thèse de l'Institut National Polytechnique de Grenoble, LIFIA, Janvier 1989. A paraître.

[Per 86] J.Pertin-Troccaz: *"Modélisation du raisonnement géométrique pour la programmation des robots"*, Thèse de L'Institut National Polytechnique de Grenoble, Avril 1986.

[PW 82] E.Pervin and J.A. Webb: *"Quaternion in computer vision and robotics"*, Technical report, CMU reports, 1982.

[The 88] P.Theveneau : *"Utilisation du raisonnement géométrique pour la planification en robotique d'assemblage : le système PAMELA "*, Thèse de L'Institut National Polytechnique de Grenoble, Novembre 1988.

[SGS 84] B.E. Shimano, C.C. Geschke, and C.H. Splading. *"Val-II a new robot control system for automatic manufacturing"*. In IEEE International Conference on Robotics, Atlanta, Mars 1984.

[SG-all] Steele J.R., Guy L., S.E. Fhalman, R.P. Gabriel, D.A. Moon, D.L. Weinreb : *"COMMON LISP - The langage "*.

[Vyg 75] M.Vygodski : *"Aide mémoire de mathématiques supérieures"*, Editions de Moscou, 1975.

[WKH 72] P.H. Winston, B. Klaus, P. Horn : *"LISP"*, seconde édition, Addison Wesley, Mars 72.