Vérification-correction de programme pour la prise en compte des incertitudes en programmation automatique des robots

Pierre Puget

THESE

présentée par

Pierre PUGET

pour obtenir le titre de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

spécialité informatique

==========

Vérification-Correction de programme pour la prise en compte des incertitudes en programmation automatique des robots

==========

23 fevrier 1989

jury:

Mr FONLUPT
Mr LIEGEOIS
Mr JORRAND
Mme TROCCAZ
Mr CHATILA
Mr CAILLOT

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Georges LESPINARD

Année 1988

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Marc	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FOGGIA Albert	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Jean-Claude	ENSPG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSIEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
RAMEAU Jean-Jacques	ENSEEG
RENAUD Maurice	UFR PGP
ROBERT André	UFR PGP
ROBERT François	ENSIMAG
SABONNADIERE Jean-Claude	ENSIEG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
CHOLLET Jean-Pierre
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
COULOMB Jean- Louis
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Madeleine
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
GHIBAUDO Gérard
HAMAR Sylvaine
HAMAR Roger
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
PASTUREL Alain
PLA Fernand
ROUGER Jean
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S
Directeurs de recherche 1ère Classe

CARRE René
FRUCHART Robert
HOPFINGER Emile
JORRAND Philippe
LANDAU Ioan
VACHAUD Georges
VERJUS Jean-Pierre

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BERNARD Claude
BINDER Gilbert
BONNET Roland
BORNARD Guy
CAILLET Marcel
CALMET Jacques
COURTOIS Bernard
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
GUELIN Pierre
JOUD Jean-Charles
KLEITZ Michel
KOFMAN Walter
KAMARINOS Georges
LEJEUNE Gérard
LE PROVOST Christian
MADAR Roland
MERMET Jean
MICHEL Jean-Marie
MUNIER Jacques
PIAU Monique
SENATEUR Jean-Pierre
SIFAKIS Joseph
SIMON Jean-Paul
SUERY Michel
TEODOSIU Christian
VAUCLIN Michel
WACK Bernard

Personnalités agrées à titre permanent à diriger des travaux de recherche (décision du conseil scientifique)

E.N.S.E.E.G
CHATILLON Christian
HAMMOU Abdelkader
MARTIN GARIN Régina
SARRAZIN Pierre
SIMON Jean-Paul

E.N.S.E.R.G
BOREL Joseph

E.N.S.I.E.G
DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G
ROWE Alain

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

E.F.P.
CHARUEL Robert

C.E.N.G
CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIB Maurice
VINCENDON Marc

Laboratoires extérieurs

C.N.E.T
DEVINE Rodericq
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

Sommaire

Remerciements

Je tiens à remercier Monsieur Jean Fonlupt pour avoir accepter de présider le jury de cette thèse, Messieurs Philippe Jorrand et Alain Liégeois pour l'attention qu'ils ont portée à ce travail, Madame Jocelyne Troccaz, Messieurs François Caillot et Raja Chatila pour avoir bien voulu participer au jury. Je remercie également toutes ces personnes pour l'intérêt qu'elles ont porté à mon travail.

Je remercie également Philippe Jorrand, Jean-Claude Latombe et Christian Laugier pour le cadre de travail dont j'ai pu bénéficier au laboratoire LIFIA pendant toute la durée de ces travaux. Je remercie la société ITMI et plus particulièrement ses dirigeants, Jean-Claude Latombe, Gérard Mézin, Yannick Descotte qui m'ont assuré un soutien financier fidèle et constant, même dans les moments difficiles.

Je remercie enfin chaleureusement toutes les personnes qui m'ont apporté une aide, de quelque façon que ce soit, sans qui ce travail ne serait pas ce qu'il est. En particulier je voudrais remercier Jocelyne Troccaz pour les nombreuses heures de travail que nous avons passées ensemble, pour l'aide qu'elle m'a apportée dans la rédaction et la relecture de ce manuscrit, Jean- Claude Latombe qui est à l'origine de nombreuses idées de ce travail, et les personnes suivantes pour les discussions fructueuses que l'on a pu avoir: Emmanuel Mazer, Christian Laugier, Pascal Théveneau, Michel Pasquier, Jose- Luis Gordillo, Christian Gandon, Pascal Di Giacomo, Fano Ramparany, Christine Bellier, Isabelle Mazon, Pierre Montcuquet, Sylvie Millioni, Marc Lenoir.

Introduction

Le travail effectué dans le cadre de cette thèse concerne la prise en compte des incertitudes géométriques dans le cadre de la programmation automatique de robots. Plus précisément, nous avons étudié une méthode de "vérification/correction" de programme qui permet d'assurer la compatibilité d'un programme de manipulation avec les tolérances imposées par le montage, étant données les incertitudes initiales sur la position des pièces et l'imprécision introduite par les différentes actions du manipulateur et l'utilisation des capteurs.

Dans cette introduction, nous rappellerons brièvement la problématique de la programmation automatique de robots niveau objet en montrant la nécessité d'une méthode de vérification/correction. Ceci sera illustré par un exemple de manipulation. Enfin, un plan du rapport sera présenté.

La problématique de la programmation automatique de robots

Nous nous interessons à la programmation automatique de robots dite de "niveau objet".

Le but d'un système de programmation automatique de robots est de transformer de façon automatique une spécification de haut niveau de la tâche, -sous forme de relations géométriques à établir-, en un programme éxécutable par un robot.

Historiquement, les premières propositions de réalisation de systèmes de programma-

tion automatique sont apparues vers les années 70 peu après l'apparition de langages évolués pour la programmation des robots d'assemblage. Ce qui paraissait comme un problème relativement simple se révéla être d'une très grande difficulté et aucune tentative d'implantation ne se solda par une réalisation de système opérationnel complet. Cependant ces tentatives aidèrent à mieux cerner le problème et eurent pour conséquence importante la mise en place de cadres de travail et la décomposition, dorénavant classique, du problème global en sous-problèmes spécifiques et (relativement) indépendants.

La décomposition du problème global de programmation d'une opération de montage en opérations plus simples a conduit à l'étude séparée des trois "opérations élémentaires" suivantes [Lau87]:

- *détermination des actions de saisie:* il s'agit de calculer les positions de contact entre le préhenseur et l'objet réalisant une "bonne" prise (stabilité et compatibilité avec la tâche) ainsi que les trajectoires d'approche, éventuellement de déproche du préhenseur lors des actions de saisie et de lâcher proprement dites.
- *détermination des trajectoires de transfert:* il s'agit de calculer les trajectoires permettant de déplacer le bras et sa charge d'une position de départ à une position d'arrivée "relativement éloignées" dans un espace peu contraint (c'est-à-dire peu encombré et peu sensible aux incertitudes).
- *détermination des opérations de montage:* il s'agit de calculer une séquence de mouvements et de lecture de capteurs pour assurer la réalisation finale de l'assemblage attendu malgré les fortes contraintes sur les incertitudes imposées par les jeux du montage.

La programmation *séparée* de chacune de ces opérations élémentaires passe par la résolution de deux types de problèmes fondamentaux:

- calcul de trajectoires et positions "sûres" (c'est à dire garantissant l'absence de collisions entre objets).
- prise en compte des incertitudes de modélisation de l'univers et des incertitudes sur les actions du manipulateur.

L'intégration d'un système complet de programmation automatique suppose, non seulement que ces problèmes soient résolus au niveau local des opérations élémentaires mais aussi globalement au niveau du programme complet. Ceci n'est pas évident car les opérations élémentaires présentent un fort degré d'interdépendance. En effet le caractère fondamental d'une action d'un manipulateur est de créer des "effets de bord" qui propagent des interdépendances entre les opérations. On peut distinguer:

- les interdépendances géométriques (par exemple la position de saisie d'une pièce pourra avoir une influence sur la trajectoire de transfert ou sur la trajectoire de montage)
- les interdépendances dues aux incertitudes (par exemple la précision et la stabilité d'une prise détermine la précision des mouvements de montages)

Les interdépendances dues aux incertitudes se manifestent de deux façons:

- une incertitude à un instant donné ne dépend pas seulement de la dernière action ou d'une seule action effectuée auparavant, mais de *multiples* actions antérieures. Le succès final d'une manipulation est donc conditionné a priori par toutes ses actions élémentaires.
- certaines actions élémentaires présentent des "conditions d'applicabilité". Elles ne sont possibles ou ont un résultat garanti que si les incertitudes initiales sont inférieures à un certain seuil. Une stratégie de mouvements fins pour une insertion cylindrique ne pourra par exemple être assurée de succès que si le contact initial de la pige se fait au niveau du chanfrein. Une action imprécise pourra donc remettre en cause l'applicabilité d'une action postérieure.

Notre travail concerne la gestion des interdépendances dues aux incertitudes dans un système de programmation automatique de robots. Nous proposons une approche appelée "vérification/correction" de programme qui opère en deux phases:

1. *vérifier* la validité du programme vis-à-vis des interdépendances entre actions dues aux incertitudes. Un programme sera considéré comme correct si les incertitudes sont inférieures aux jeux (tolérances) imposées par le montage (i.e. elles permettent un succès assuré de la manipulation) et si toutes les actions sont compatibles entre elles et compatibles avec les conditions initiales (i.e. les conditions d'applicabilité de toutes les actions sont satisfaites).
2. dans le cas de non validité du programme, proposer la *correction* de celui-ci, c'est à dire la modification *locale* du programme consistant à ajouter des actions diminuant l'incertitude (des actions de capteurs par exemple).

Exemple

Considérons un exemple d'assemblage (cf fig 0.1) inspiré d'une manipulation effectuée au laboratoire LIFIA dans le cadre du projet ARA. Les problèmes posés par cette manipulation particulière ne sont solubles actuellement par aucun système de programmation automatique. La mise au point a été faite par des programmeurs "humains".

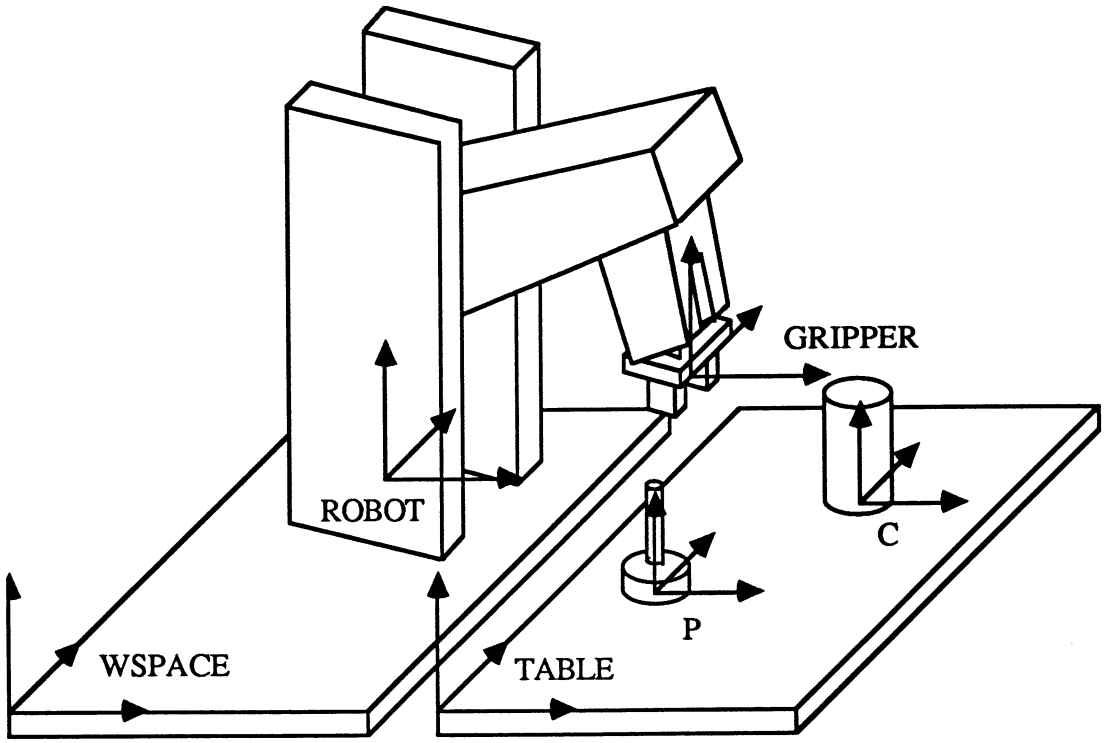


Figure 0.1 : Exemple d'assemblage

Cependant nous prenons cet exemple pour l'intérêt didactique et méthodologique qu'il présente.

Il s'agit de monter un amortisseur, en particulier d'insérer un piston P dans un cylindre C. Les jeux de montage sont faibles (de l'ordre quelques dixièmes de millimètre). La précision du robot est du même ordre. Les pièces sont initialement positionnées "à la main", sans instrumentation particulière. On peut espérer une précision sur leurs position de l'ordre du millimètre.

La décomposition présentée au paragraphe précédent amène à résoudre successivement:

- *le problème de la saisie:* ici la géométrie du piston est très simple et présente une grande symétrie. La prise la plus évidente consiste à mettre en contact deux génératrices diamétralement opposée de la tige avec les deux mors de la pince. La direction d'approche n'est pas très contrainte et on peut se contenter de descendre verticalement vers la position de prise.
- *le problème du transport:* la scène étant très peu contrainte, il suffit de se dégager

verticalement vers le haut, de se déplacer horizontalement jusqu'à la verticale du cylindre et de redescendre pour garantir l'absence de collisions entre les différents objets.

- *le problème des mouvements fins de montage:* les pièces à assembler ne présentant pas de géométrie propice à l'utilisation d'un capteur de force (en particulier, absence de chanfrein): il n'y a pas de stratégie particulière à appliquer, il faut se positionner au dessus de l'orifice du cylindre et descendre verticalement. Cette opération ne sera possible que si le positionnement au dessus du cylindre se fait avec une précision inférieure aux jeux de montage. Ceci représente pour l'opération élémentaire des mouvements fins de montage, ce que nous avons appelé une "condition d'applicabilité".

Les opérations ainsi décomposées se traduisent facilement dans un langage de niveau effecteur tel que LM [MM85]. En ignorant les problèmes de déclaration et d'initialisation de variables, on considère le programme simple suivant:

```
DEPLACER ROBOT A Pprise*TRANSLAT(VZ,10);
DEPLACER ROBOT A Pprise EN MODE CARTESIEN;
FERMER PINCE;
DEPLACER ROBOT DE TRANSLAT(VZ,10);
DEPLACER Ppos A Ctop*TRANSLAT(VZ,10);
DEPLACER Ppos A Ctop*TRANSLAT(VZ,-100) EN MODE CARTESIEN;
OUVRIR PINCE;
DEPLACER ROBOT DE TRANSLAT(VZ,200);
```

Ce programme ne prend pas du tout en compte les incertitudes. S'il peut fonctionner en simulation, il y a très peu de chance qu'il s'éxécute correctement sur un robot réel à cause des incertitudes et des jeux de montage assez faibles.
Un programme destiné à être éxécuté réellement devra comporter également des instructions faisant appel à des capteurs. Nous verrons dans la suite de ce rapport comment notre approche permet de détecter les endroits où les échecs peuvent survenir et la façon dont on pourra le modifier pour le rendre éxécutable.

Plan du rapport

Ce rapport se compose de six chapitres et trois annexes.

Le chapitre 1 présente de façon détaillée le problème de la prise en compte des incertitudes au sein d'un système de programmation automatique de niveau objet. En particulier, il montre l'intérêt primordial d'une approche de vérification/correction de programme dans le contexte imposé par les architectures logicielles et les stratégies de planification qui sont communément adoptées.

Le chapitre 2 présente les travaux principaux qui ont été faits ou qui sont menés actuellement et qui concernent la gestion des incertitudes dans la planification d'actions de robots. Nous mentionnons aussi les travaux qui traitent de la représentation des incertitudes de position des objets dans un contexte éventuellement différent du nôtre.

Notre approche de vérification/correction de programme passe par la description formelle de l'effet de l'éxécution de chaque action élémentaire du robot. Le chapitre 3 présente des méthodes de description connues dans le cadre de l'informatique fondamentale dont nous nous sommes inspirés. Ces méthodes ne sont pas directement applicables à un langage de programmation de robots et nous analysons le problème de leur adaptation.

Notre approche passe également par une représentation explicite des incertitudes de position des objets. Cependant, elle n'est absolument pas dépendante d'un type de représentation particulier. La représentation que nous avons adoptée fait l'objet du chapitre 4.

Le chapitre 5 présente un langage de programmation de niveau effecteur à la fois proche des langages courants tels que LM ou VAL et adapté à la vérification/correction. L'effet de l'éxécution de chaque instruction du langage est précisément décrit dans le formalisme présenté au chapitre 4. Les méthodes de vérification/correction sont ensuite montrées en détail pour ce langage particulier.

L'analyse de la validité de notre approche et les résultats expérimentaux que nous avons obtenus font l'objet du chapitre 6.

Dans l'annexe A, nous faisons quelques rappels sur la logique de Hoare utilisée pour la correction de programme. L'annexe B regroupe le détail des calculs que nous avons utilisé pour la représentation des incertitudes. L'annexe C présente une autre représentation possible des incertitudes. Cette seconde représentation est celle qui

figure dans nos précédentes publications et que nous avions initialement retenue.

Chapitre 1

Intérêt de la Vérification/correction de programme au sein d'un SPANO

1 La programmation des robots

1.1 Généralités

On peut définir le problème général de la programmation des robots d'assemblage de la façon suivante:

Etant donnés:

- *un montage à réaliser (c'est-à-dire un ensemble de pièces mécaniques à déplacer de façon à créer entre elles certaines relations géométriques [GP81])*
- *un ensemble d'effecteurs (robots, bras manipulateurs) capables de se déplacer et d'agir sur l'environnement par l'intermédiaire d'un outil terminal*
- *un ensemble de capteurs*
- *un interpréteur permettant de commander les effecteurs et d'exploiter les données fournies par les capteurs,*

trouver une séquence de commandes pour réaliser le montage

Dans le cadre de ce rapport, nous ne considérerons que des modes de programmation dits "textuels" où les commandes envoyées à l'interpréteur sont sous la forme d'un programme écrit dans un langage de programmation de robots. Un tel langage est caractérisé par le niveau d'abstraction auquel on exprime les actions du robot et les opérations des capteurs. On parle traditionnellement de *niveau de programmation.*

1.2 Niveaux de programmation

De façon courante, ces niveaux de programmation sont classés hiérarchiquement [Vol88] (cf figure 1.1). Cette nomenclature peut paraitre parfois floue et discutable. Il n'est pas de notre propos d'en discuter ici. Pour fixer simplement les idées, on peut dire qu'un langage A sera d'un *plus haut niveau* qu'un langage B si l'éxécution *d'une* instruction de A est équivalente à l'éxécution de *plusieurs* instructions de B. On peut dire aussi qu'une instruction I d'un langage donné est de plus haut niveau qu'un ensemble d'instructions E de ce même langage si l'éxécution de I est équivalente à l'éxécution d'un sous-ensemble de E.

Comme nous verrons par la suite, nous nous intéressons particulièrement à deux niveaux de programmation: le niveau *objet* et le niveau *effecteur.*

1.2.1 Niveau effecteur

La programmation de niveau effecteur est caractérisée par la description explicite et complète de toutes les actions et opérations de capteurs:

- les positions d'arrêt du manipulateur sont décrites par la donnée des valeurs de *tous* les paramètres de position de l'outil terminal,
- les trajectoires de déplacement sont déterministes et ne dépendent pas de l'environnement extérieur: elles se font suivant un mouvement synchronisé des différents axes du robot (déplacements dits "libres"), ou sont décrites de façon exacte (trajectoire prédéfinie, rectiligne, circulaire ou selon une courbe paramétrée)
- les actions de l'outil terminal (ouverture et fermeture de pince dans le cas de l'assemblage) sont également données explicitement,
- l'exploitation des données capteurs (c'est-à-dire l'asservissement d'un mouvement sur une donnée capteur ou une stratégie de mouvements basée sur des mesures faites par un capteur) est également décrite précisément dans le programme.

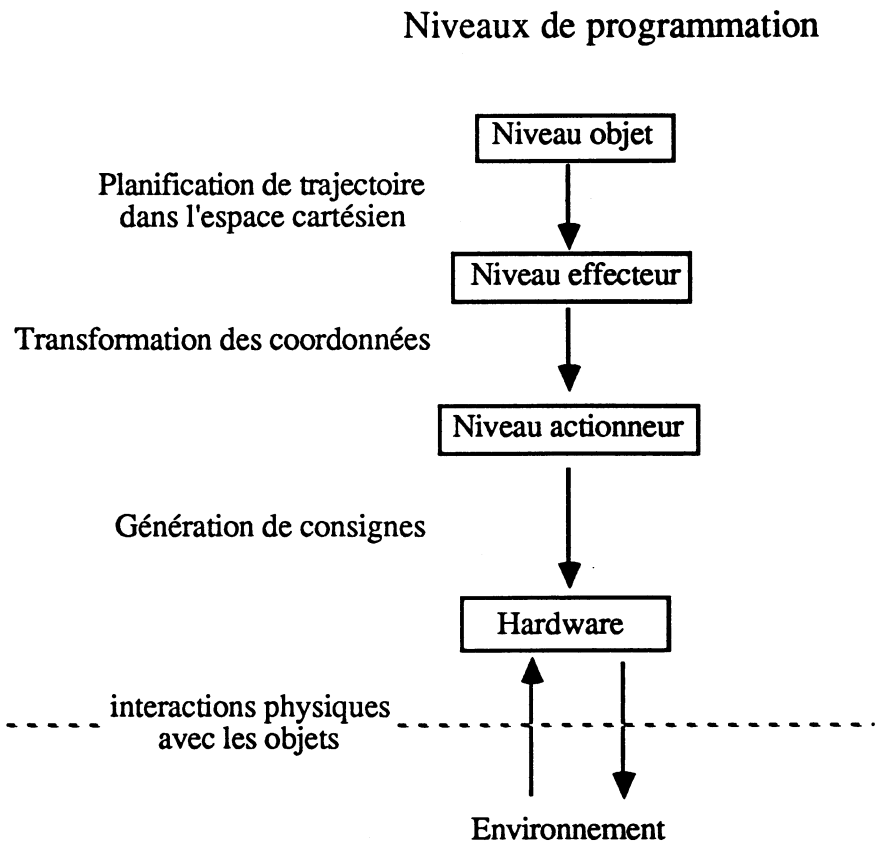


Figure 1.1 : Nomenclature des différents niveaux de programmation

La figure 1.2 est un exemple de programme de niveau effecteur écrit dans le langage LM qui réalise la mise en place de la plaque sur le support de la figure 1.3[1].

Au niveau effecteur, toutes les positions du manipulateur et des objets, ainsi que toutes les trajectoires sont exprimées dans l'espace *cartésien* du robot, indépendamment de sa structure mécanique. Le système prend en charge le calcul des coordonnées articulaires pour les déplacements et positions d'arrêt.

Il résulte de ces caractéristiques que les différentes tâches suivantes sont à la charge du programmeur:

- calcul des trajectoires pour éviter toute collision entre objets,

[1]Cet exemple est tiré de [Per86]

```
PROGRAMME MONTAGE;
CO Le repere ROBOT est situe en bout de pince et les valeurs de
     variables d'etat MX, MY, MZ, FX, FZ sont ramenees en ce point;
CO Declarations;
REPERE SUPPORT, FACE_A_DE_SUPPORT, PLAQUE, FACE_INF_DE_PLAQUE,
          APPROCHE_SUPPORT, TIGE, INIT;
REEL SEUIL_FX, SEUIL_FZ, SEUIL_CONTACT, EPS;
BOOLEEN MIS_EN_PLACE;
EXT PROCEDURE SAISIE (REPERE, REEL);
DEBUT
CO Une partie des initialisations;
SEUIL_FX:=4.; SEUIL_FZ:=2.; SEUIL_CONTACT:=5.; EPS:=10.;
APPROCHE_SUPPORT:=FACE_A_DE_SUPPORT*TRANSLAT(VX,10.);
FIXER VITESSE A 0.8;
INIT:=ROBOT;
CO Saisie et positionnement de la plaque;
SAISIE (PLAQUE,70.);
LIER PLAQUE A ROBOT;
DEPLACER PLAQUE A APPROCHE_SUPPORT*TRANSLAT(VZ,10.);
FIXER VITESSE A 0.05;
MIS_EN_PLACE:=FAUX;
TANTQUE NON MIS_EN_PLACE FAIRE
        DEPLACER PLAQUE A APPROCHE_SUPPORT*TRANSLAT(VZ,-10.)
                    JUSQUA FZ > SEUIL_FZ;
        CO Test sur MX et correction de mouvement;
        SI MX > EPS ALORS DEPLACER PLAQUE DE ROT(VX,PI/2)
                    EN MAINTENANT
                    FZ = SEUIL_CONTACT JUSQUA MX < EPS;
        CO Meme traitement pour MY;
        SI FZ > SEUIL_FZ ALORS MIS_EN_PLACE:=VRAI;
FINFAIRE;
MIS_EN_PLACE:=FAUX;
TANTQUE NON MIS_EN_PLACE FAIRE
        DEPLACER PLAQUE A FACE_A_DE_SUPPORT*TRANSLAT(VX,-10.)
                    EN MAINTENANT FZ=SEUIL_CONTACT,MX=0,MY=0
                    JUSQUA FX > SEUIL_FX;
        CO Test sur MZ et correction de mouvement (cf. MX);
        SI FX > SEUIL_FX ALORS MIS_EN_PLACE:=VRAI;
FINFAIRE;
OUVRIR PINCE SANS ATTENTE;
DELIER PLAQUE DE ROBOT;
DEPLACER ROBOT DE TRANSLAT(VZ,100.);
FIXER VITESSE A 0.8;
DEPLACER ROBOT A INIT;
CO Saisie et insertion de la tige ;
...
FIN;
```

Figure 1.2 : Exemple de programme niveau effecteur

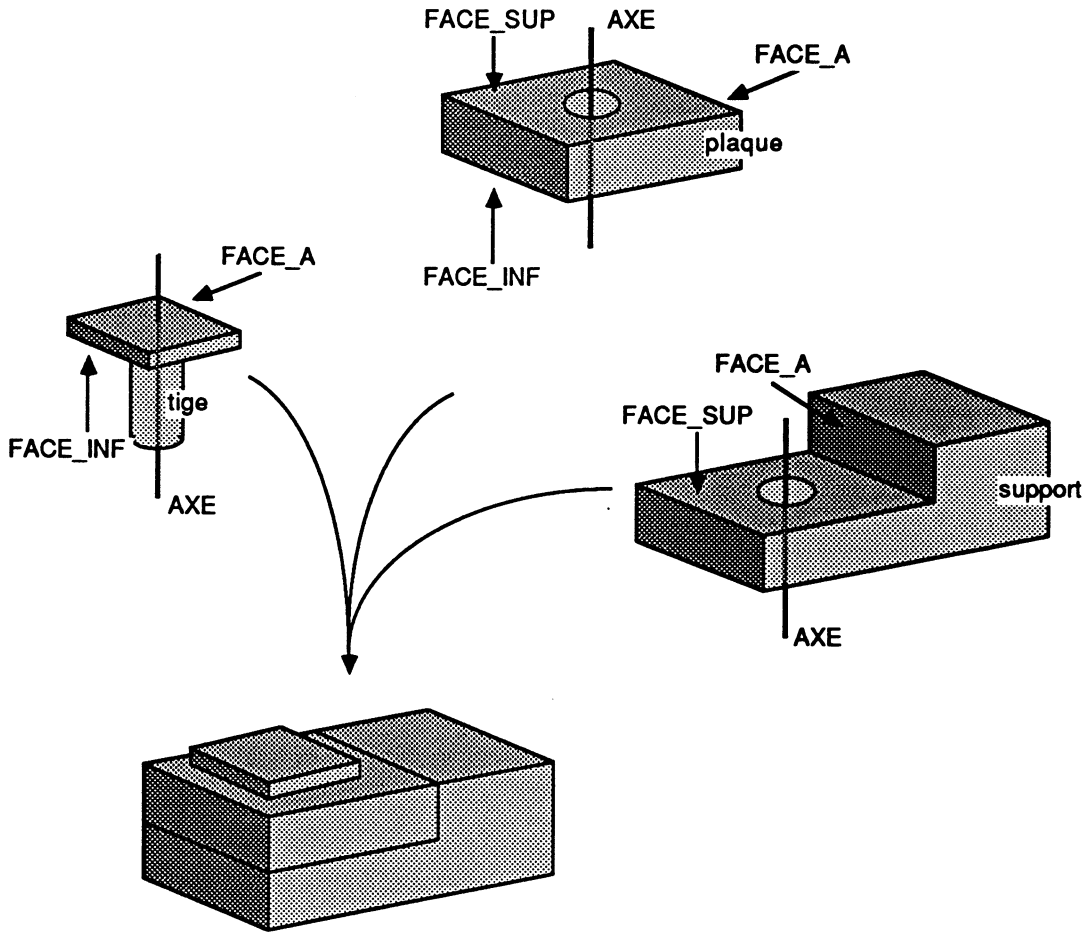


Figure 1.3 : Exemple de montage

- organisation de l'espace de travail et calcul des trajectoires et positions de façon à ne pas dépasser les butées mécaniques du bras,
- recherche de stratégies de mouvements, et utilisation des capteurs pour s'affranchir des incertitudes sur la forme et la position des objets, sur les déplacements du robot qui peuvent remettre en cause le succès de la manipulation.

1.2.2 Niveau objet

La programmation de niveau objet est caractérisée par une description symbolique de la tâche en termes de relations géométriques à réaliser. Par exemple, la figure 1.4 montre un exemple de programme niveau objet. La première instruction seule correspond au programme LM précédent (figure 1.2).

Realiser: AXE de plaque (aligne) AXE de support
FACE_INF de plaque (contre 0) FACE_SUP de support
FACE_A de plaque (contre) FACE_A de support

Realiser: AXE de tige (aligne) AXE de support
FACE_INF de tige (contre 0) FACE_SUP de plaque
FACE_A de tige (contre) FACE_A de support

Figure 1.4 : Exemple de programme niveau objet

Cette description symbolique de la tâche est associée à un modèle de l'environnement du robot et à une description dans ce modèle de la situation initiale du manipulateur et des objets. Ce modèle est de nature essentiellement géométrique. Il inclut une représentation morphologique (description de la forme) et spatiale (description de la position) des objets et du manipulateur. On peut également trouver une représentation des incertitudes et une représentation de certaines propriétés physiques des objets utiles à la planification des mouvements et opérations d'assemblage [Per86].

L'intérêt de la programmation de niveau objet est évident: c'est le système qui doit prendre en charge les différents problèmes qui sont à la charge du programmeur au niveau effecteur.

Dans le cadre de ce chapitre, nous ferons l'abus de langage suivant: par *"programme"* de commande de robots, nous sous-entendrons un *"programme de niveau effecteur"*. Lorsque nous nous placerons au niveau objet, nous mentionnerons explicitement un *"programme de niveau objet"*.

1.3 Caractéristiques d'un "bon" programme

Etant donné un montage, il y a évidemment un grand nombre de programmes (de niveau effecteur) qui sont aptes à réaliser ce montage. Parmi ceux-ci, un "bon" programme doit présenter les caractéristiques suivantes:

1. La première exigence est évidemment que le programme produit garantisse que les relations géométriques caractérisant l'état final soient réalisées après éxécution, pour tout état initial possible.
2. Il ne doit pas y avoir de collision entre le manipulateur et les objets ou entre les objets au cours de la manipulation.
3. Les mouvements commandés doivent se faire dans l'espace de travail du manipulateur (c'est-à-dire avec respect des butées mécaniques)
4. Les forces appliquées sur les objets ne doivent pas occasionner de déformation ou de rupture de ceux-ci.
5. Il ne doit pas se produire de mouvements non contrôlés comme des chutes d'objets ou des glissements intempestifs.
6. Le programme produit doit être robuste, c'est-à-dire peu sensible aux incertitudes intervenant au cours de la manipulation.

Enfin parmi les solutions potentielles satisfaisant les critères ci-dessus, on choisit par exemple les programmes les plus efficaces du point de vue du temps d'éxécution, ou de l'usure du matériel, ou de la consommation de certaines ressources.

1.4 "Savoir-faire" à mettre en œuvre

La production d'un programme satisfaisant aux conditions énoncées ci-dessus impose que puissent être résolus un certain nombre de problèmes de natures sensiblement différentes. C'est ce que nous entendons par l'expression "mise en œuvre de savoir-faire". Pour la programmation de robots, nous pouvons distinguer essentiellement deux savoir-faire.

1.4.1 Résolution des problèmes d'encombrement spatial et de morphologie des objets

Le calcul de positions et trajectoires sûres garantissant la non-collision d'objets et la détermination des relations entre objets (en particulier lors de la saisie et du montage) garantissant le minimum de mouvements non contrôlés se fait en résolvant respectivement des problèmes dits d'encombrement spatial et de morphologie des objets. Plus précisément, ce problème de calcul de positions et trajectoires sûres peut se nuancer pour chaque opération élémentaire (cf [Lau87]):

- la détermination des opérations de saisie se fait essentiellement en considérant la morphologie des objets (pour la position de prise et la trajectoire d'approche) et les caractéristiques physiques des contacts entre objets (pour le calcul de la stabilité des prises potentielles)

- la détermination des trajectoires de transfert est un problème essentiellement d'encombrement spatial des objets.

- la détermination des opérations de montage met en œuvre essentiellement des raisonnements sur la morphologie des objets, sur les contacts physiques entre objets (pour la planification de mouvements asservis par des données de capteur de force).

1.4.2 Prise en compte des incertitudes

Nous entendons par "incertitudes" tout ce qui concerne le manque d'information ou le caractère erroné de l'information que l'on a sur l'environnement du robot. Nous donnerons par la suite une définition beaucoup plus formelle (mais aussi plus restrictive) de ce terme. Ces incertitudes peuvent résulter de l'insuffisance des modèles de l'environnement ou de paramètres de ces modèles ayant des valeurs fausses ou inconnues. Ces incertitudes portent principalement:

- sur l'état de l'univers à un instant donné. Certaines caractéristiques de l'univers peuvent être inconnues (cela arrive *parfois*) ou ces caractéristiques peuvent être fausses (cela arrive *toujours* dans une plus ou moins grande mesure).

- sur l'effet des actions du manipulateur. Elles sont dues à la fois à la commande du robot imparfaitement modélisée (discrétisation des pas codeurs, erreurs d'arrondi et de troncatures dans les calculs) et à des phénomènes physiques non modélisés (élasticité, dilatation, frottements au niveau des objets en interaction avec le robot).

Pour la plupart des manipulations, une faible variation de certains paramètres définissant l'environnement du robot peut modifier radicalement le comportement d'un programme ([Maz87a], cf figure 1.5). Une incertitude trop grande sur certains paramètres "critiques" remet fortement en cause le succès d'un programme. Au niveau de son élaboration, il est nécessaire de prendre en compte ces incertitudes sous peine de produire un programme très peu robuste.

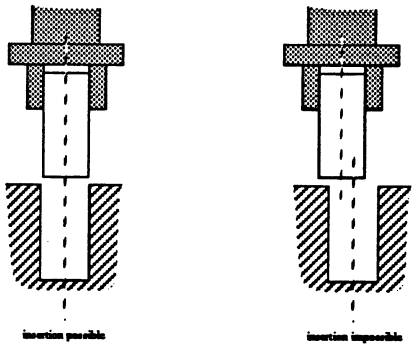


Figure 1.5 : influence d'un faible variation de la position latérale du goujon

1.5 Interdépendances entre actions

Une autre difficulté intervenant dans la programmation des robots est le fort degré d'interdépendance qui existe entre les actions. L'éxécutabilité d'une action du manipulateur et son succès (c'est-à-dire la satisfaction du but pour lequel elle est exécutée) dépendent de l'état de l'environnement avant son éxécution (cf fig 1.5). Or cet état initial résulte des conditions initiales de la manipulation entière et des effets cumulés de *toutes les actions antérieures.* Ainsi l'existence d'une trajectoire de transfert pourra dépendre de la position de prise d'un objet encombrant (par exemple une barre). De même, la précision d'une opération de saisie d'un objet déterminera en partie la précision de sa position au moment du montage. Les interdépendances entre actions concernent essentiellement la géométrie d'une part et les incertitudes de forme et de position d'autre part.

Nous ne nous intéressons pas ici aux interdépendances dues à la géométrie des objets mais uniquement aux interdépendances liées aux incertitudes. Celles-ci présentent les caractéristiques suivantes sur lesquelles nous nous appuyons pour justifier notre approche de vérification/correction de programme:

1. L'utilisation judicieuse de capteurs et de mouvements à compliance permet de réduire les interdépendances liées aux incertitudes. En effet, par ce biais, on peut augmenter très fortement le nombre de situations où une opération réussira, ce qui revient à affaiblir les conditions d'applicabilité et par là réduire la dépendance de l'opération vis-à-vis des opérations antérieures ou de l'état initial.
2. Si on dispose de capteurs adéquats, il est comparativement beaucoup plus "facile" de modifier les incertitudes (il n'est intéressant de les modifier que pour les

réduire) que la géométrie de l'environnement.

3. Même si la résolution du problème des interdépendances doit être traitée globalement, c'est à dire en considérant le programme tout entier, en cas d'incompatibilité d'opérations, il est possible de le corriger en le modifiant localement, sans avoir à replanifier d'opération. Cette modification locale consiste en l'ajout d'actions destinées à réduire l'incertitude (cf point précédent): lecture de capteurs ou mouvements mettant en jeu des contacts entre objets (en particulier les mouvements étudiés par Mason [Mas82,EM86]).

2 Construction d'un système de niveau objet

2.1 Choix d'une architecture logicielle

Nous pensons que le choix de l'architecture pour l'implantation d'un système de niveau objet a une importance fondamentale quant à la nature des problèmes à résoudre et constatons que ce choix est souvent fait *a priori.* Nous présentons ici le type de structure qui est le plus couramment retenu.

Pour pratiquement tous les projets d'implantation de système de programmation de niveau objet, il a été fait le choix de partir d'un système de niveau effecteur déjà existant et de lui ajouter en amont un interpréteur de niveau objet (cf figure **1.6**). Cet interprétreur reçoit en entrée un programme de niveau objet et envoie au système aval un programme de niveau effecteur directement éxécutable.

2.2 Enoncé du problème et discussion

Dans ce contexte fixé a priori, le problème de la réalisation d'un système de programmation de niveau objet peut s'énoncer de la façon suivante:

> *"Construire un interpréteur qui produit de façon off-line un programme de niveau effecteur dont l'éxécution réalisera les relations géométriques décrites par un programme de niveau objet fourni en entrée"*

De cet énoncé, il ressort que le problème de programmation de niveau objet présente les caractéristiques suivantes:

1. *C'est un problème de planification*
   Il s'apparente au problème général de planification d'un agent *opérant en univers*

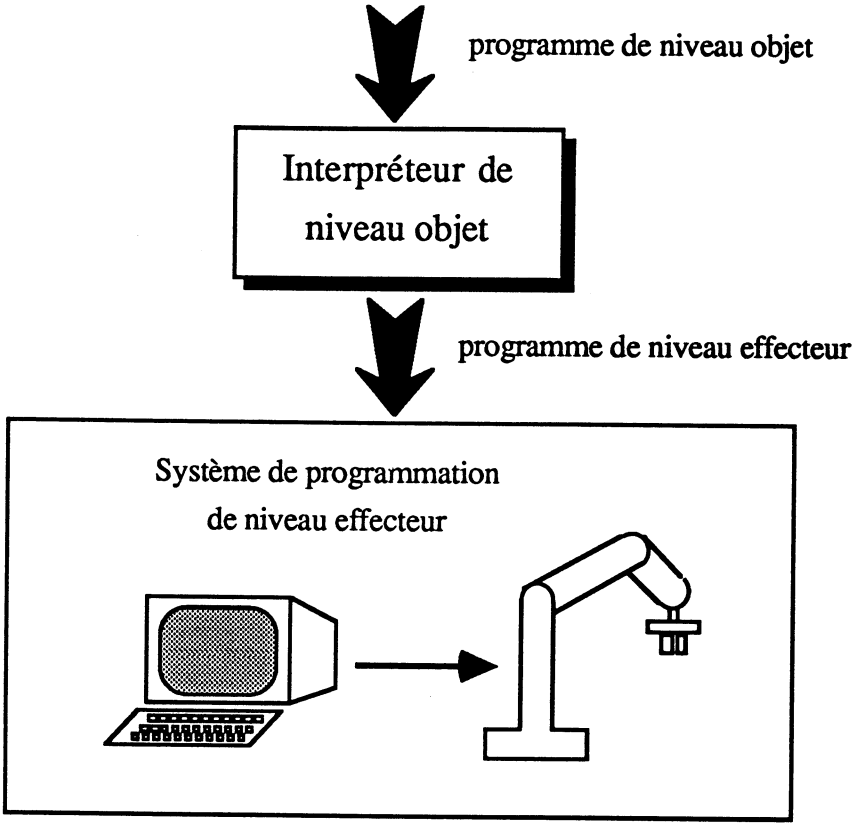


Figure 1.6 : Structure classique d'un système de niveau objet

réel qui est un domaine à part entière de l'intelligence artificielle. Nous verrons en quoi notre approche peut être rattachée à ce domaine.

2. *C'est un problème de calcul off-line*
Du fait de l'architecture particulière des systèmes de programmation de niveau objet (cf paragraphe 2.1) l'interpréteur de niveau objet effectue la planification *totale* des opérations qui sont ensuite éxécutées. Cela implique que le système doit "prévoir" *a priori* toutes les situations possibles pouvant survenir au cours de la manipulation (en particulier faire une prévision des incidents susceptibles de se produire), les moyens à mettre en œuvre pour caractériser ces situations et les actions correctives à mener pour chacune d'entre elles. Pour éviter une explosion combinatoire qu'on est absolument incapable de maitriser, il importe que l'état initial de l'environnement soit "qualitativement bien spécifié", c'est-à-dire que les inconnues sur l'état initial ne portent que sur l'instanciation de

certains paramètres mesurables par des capteurs.

3. *La solution dépend du système aval*
   Du fait également de l'architecture des systèmes de programmation de niveau objet, la recherche de solution dépendra du système aval sur lequel devra s'éxécuter le programme (par exemple les caractéristiques du manipulateur et des capteurs). Du point de vue de l'interpréteur niveau objet, le système aval se comporte comme une boite noire dont on ne connait que les entrées-sorties:

   - les entrées étant les différentes instructions de niveau effecteur.
   - les sorties étant l'effet de l'éxécution de chacune de ces instructions.

   Ce problème particulier de description des "entrées-sorties" est fondamental et sera discuté au chapitre 3.

2.3 La décomposition "classique" en sous-problèmes Saisie-Transfert-Montage

Au sein d'un SPANO il est courant de décomposer le problème global de la planification du montage d'un objet en planifiant séparément les trois opérations élémentaires de saisie, mouvements de transfert et montage.
Cette décomposition se justifie par le fait que les trois sous-problèmes correspondent à des phases successives du montage et qu'ils mettent en œuvre des savoir-faire distincts. De ce fait il seront également traités par des modules spécialisés différents plus efficaces.
Malheureusement, du fait des interdépendances entre actions, la génération d'un programme complet ne peut se faire en juxtaposant simplement les solutions produites pour les opérations de saisie, transfert et montage. En effet, même si la détermination de trajectoires et positions sûres, la prise en compte des incertitudes sont résolues *localement* en planifiant séparément les trois opérations, l'intégration ne peut se faire qu'en résolvant à un niveau *global* certains problèmes nés des des interdépendances entre saisie, transfert et montage.

La planification séparée des différentes opérations est effectuée en faisant certaines hypothèses sur les états initiaux et finaux de chaque action. Au moment de l'intégration de la solution complète, il faut vérifier la cohérence de ces hypothèses entre elles et leur cohérence avec les contraintes imposées par l'environnement du robot. Dans le cas d'incohérence, il faut éventuellement replanifier les opérations.

3 Le principe de Vérification/Correction de programme

Les diverses hypothèses et considérations précédentes concernant les problèmes liés à l'intégration d'un programme complet nous ont amené à développer une approche appelée Vérification-Correction de programme. A partir d'un programme de manipulation consistant en une suite d'instructions, la vérification-correction permet:

1. de vérifier la validité de ce programme vis à vis des incertitudes affectant le manipulateur et les objets,
2. d'apporter une aide à la correction locale du programme dans le cas de non validité (le problème de la correction complètement automatique du programme est encore mal maitrisé).

3.1 Définitions

Un *programme* est considéré comme une suite d'instructions $I_1, I_2, \ldots, In$. Chaque instruction transforme l'environnement du robot d'un état initial vers un état final. Or une action n'est éxécutable et ne donne le résultat qu'on attend que si l'état initial satisfait certaines conditions que nous appelons "conditions d'applicabilité". Ces conditions d'applicabilité peuvent être représentées par des contraintes sur la position des objets dont la signification littérale est du type:

"La position de l'objet A par rapport à l'objet B est P_0*"*

"L'incertitude de position de l'objet C par rapport à l'objet D est inférieure à I_0*"*

Plus formellement, si le modèle de l'univers du robot dépend de p paramètres $w_1, w_2, \ldots w_p$, une contrainte sera un prédicat $C(w_{i_1}, w_{i_2}, \ldots, w_{i_q})$ où les w_{i_j} sont des paramètres du modèle et q un entier inférieur à p.

On considère alors un *programme contraint* qui est un programme pour lequel on associe une contrainte de faisabilité C_i à chaque instruction I_i.
On note:

$$CP = \{(C_1, I_1), (C_2, I_2), \ldots, (C_m, I_m)\}$$

un tel programme contraint. Eventuellement, certaines contraintes pourront être égales au prédicat constant VRAI.

Soit W_0 l'état initial réel de l'environnement avant la manipulation. Soit (W_i) la suite des états intermédiaires, W_i étant l'état résultant de l'éxécution des i premières instructions. Le programme contraint précédent CP sera déclaré correct pour W_0, c'est à dire qu'il sera apte à être éxécuté, si, dans tout état intermédiaire W_i (i variant de 0 à $m-1$), la contrainte d'applicabilité C_{i+1} est vérifiée. Si ce n'est pas le cas, le programme sera déclaré incorrect: un incident sera susceptible de se produire lors de l'éxécution de I_{i+1}.

Or du fait des incertitudes sur la position des objets et sur les mouvements du manipulateur, on ne connait pas les états W_i réels mais seulement les ensembles $\mathcal{W}_i$ des états possibles. L'éxécutabilité du programme est alors assurée si tout état W_i dans $\mathcal{W}_i$ satisfait C_i.

3.2 Phase de vérification

La première phase de notre approche (la vérification) consiste à calculer, à partir des conditions initiales de la manipulation données par $\mathcal{W}_0$, les ensembles $\mathcal{W}_i$ d'états intermédiaires. Pour chacun de ces ensembles, on vérifie que la contrainte C_{i+1} correspondante est satisfaite pour tout élément W_i. Si c'est le cas, le programme est déclaré correct. Le processus mis en œuvre au cours de cette phase est appelé *"propagation descendante des incertitudes"* (voir figure 1.7).

3.3 Phase de correction

Dans le cas où une des contraintes C_j n'est pas satisfaite, une seconde phase de notre méthode consiste à calculer des nouvelles contraintes $C_j^{j-1}, C_j^{j-2} \ldots C_j^1$ portant sur les états antérieurs de telle sorte que si l'une d'elles est satisfaite, alors C_j le sera également (voir figure 1.8). Chaque contrainte C_j^i est choisie pour contraindre les états précédents de façon minimale (une contrainte plus faible étant plus facile à satisfaire). Le processus de calcul mis en œuvre au cours de cette phase est appelé *"propagation ascendante des contraintes"*. Le principe de la correction de programme proprement dite est d'ajouter localement de nouvelles instructions ou "rustines". Le but d'une rustine insérée entre deux instructions I_{i-1} et I_i est de réduire l'ensemble des états possibles $\mathcal{W}_i$ à un nouvel ensemble $\mathcal{W}_i'$ plus petit afin que la condition C_j^i soit satisfaite dans n'importe quel état de $\mathcal{W}_i'$, et qu'alors ainsi tous les états ultérieurs W_j satisfassent la condition C_j.

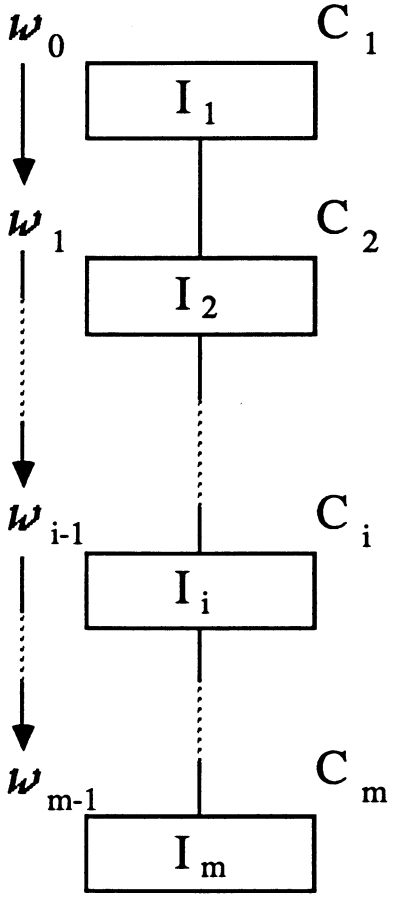


Figure 1.7 : Propagation descendante des incertitudes

Remarque: Cette approche est à rapprocher du traitement des interdépendances en génération de plan par la méthode de régression du système ABSTRIPS [Nil82]

3.4 Niveau de programmation adapté à la Vérification-Correction de programme

La programmation d'un robot pouvant se faire à différents niveaux d'abstraction, on peut légitimement se demander quel est le niveau de langage le plus adapté à la VCP.

Faire la vérification d'un programme avec un langage de haut niveau semble a priori intéressant pour les raisons suivantes:

- Le nombre d'instructions dans un programme écrit dans un langage de haut niveau est moins élevé. On aura donc moins de calculs de propagations ascen-

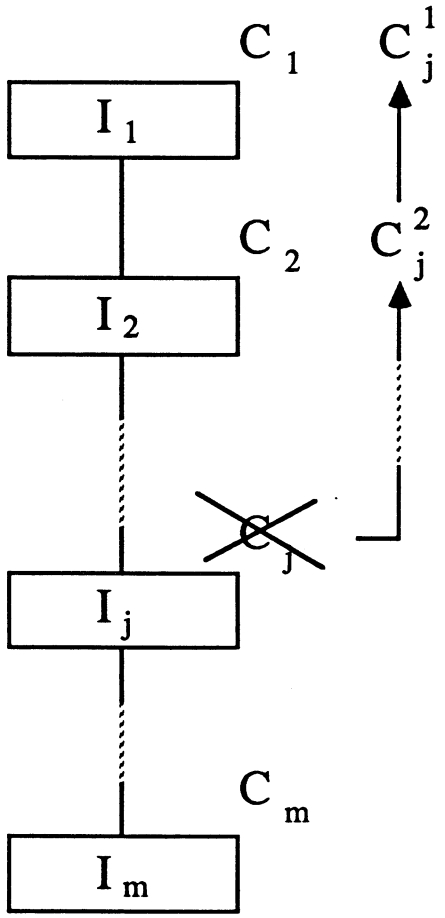


Figure 1.8 : Propagation ascendante des contraintes

dante et descendante à faire.

- La sémantique d'instructions de haut niveau (c'est-à-dire la signification ou l'effet de l'éxécution de ces instructions) est plus riche et moins dépendante du contexte.

Cependant, faire la vérification avec un langage de haut niveau présente les inconvénients suivants:

- Les interdépendances entre actions du robot disparaissent quand on écrit un programme de manipulation avec des langages de plus haut niveau. Le cas extrême est le programme d'entrée écrit au niveau objet où il n'y a aucune interdépendance entre les différentes instructions.
- Dans la pratique, l'implantation de primitives de haut niveau est très complexe

et encore mal maitrisée [Whi87]. La définition de la sémantique d'instructions de haut niveau sous-entend, et par là implique, une grande robustesse du plus bas niveau et reporte sur ce bas niveau une grosse part des problèmes.

En conséquence, nous avons choisi de faire la Vérification/Correction de programme au niveau effecteur qui est le niveau le plus bas auquel on ait accès dans le cadre d'un système de programmation de niveau objet ayant l'architecture présentée au paragraphe 2.1. Cependant, nous verrons au chapitre 3 qu'un langage de programmation ne se prête à la vérification-correction que sous certaines conditions. En particulier, il faut que l'effet de l'éxécution de chaque instruction, autrement dit sa sémantique puisse être décrit de façon satisfaisante. Cela n'est pas tout à fait le cas pour les langages de programmation de niveau effecteur courants. C'est pourquoi nous considérerons un langage adapté, réduit par rapport aux langages existant.

Chapitre 2

Bibliographie

Toute personne utilisant de près ou de loin un robot se trouve confronté au problème des incertitudes: imprécisions des mouvements du manipulateur, imprécision des résultats des mesures effectuées par les capteurs, erreurs de modélisation. Ce problème est un problème fondamental de la robotique et une très nombreuse littérature l'évoque plus ou moins. Nous avons choisi de présenter dans ce chapitre deux classes de travaux antérieurs ou contemporains aux nôtres.

La première section du chapitre présente des travaux où sont mises en œuvre des méthodes de propagation de contraintes pour la planification d'opérations de manipulation. Les deux premiers travaux ont été à la base de notre inspiration et, s'il fallait établir une généalogie, pourraient revendiquer la paternité de notre approche. Le troisième est actuellement en cours au LAAS à Toulouse et propose une approche de vérification/correction de programme comparable à la nôtre.

La seconde section présente un cadre de travail tout à fait différent communément utilisé en vue de la planification de mouvements de montage avec prise en compte des incertitudes n'utilisant pas de propagation de contraintes.

La troisième section regroupe un ensemble de travaux plus disparates: ils traitent

tous de problèmes liés aux incertitudes, parfois dans des contextes différents du nôtre (robotique mobile en particulier) mais utilisent des représentations des incertitudes pouvant rentrer dans le cadre de notre approche.

1 Planification de manipulation par propagation de contraintes

1.1 La prise en compte des incertitudes dans le système AL

Le travail de Taylor présenté dans son mémoire de PhD [Tay76] est une des premières tentatives de réalisation d'un système de programmation de niveau objet. C'est dans le cadre de ce travail que pour la première fois, on fit intervenir les incertitudes dans le processus de planification. L'objet de ce paragraphe est de présenter les aspects de ce travail les plus intéressants pour nous: la modélisation de l'environnement incluant une représentation des incertitudes de position et les méthodes de planification prenant en compte ces incertitudes.

1.1.1 Modélisation de l'environnement

L'intéret majeur de la modélisation adoptée par Taylor est qu'elle permet:

- la traduction d'expressions symboliques exprimant les relations entre les objets[1] sous forme de données utilisables par le planificateur,
- la représentation de transformations dont tous les degrés de liberté ne sont pas parfaitement connus,
- la représentation des incertitudes de position,
- le calcul de propagation d'incertitudes.

La base de cette modélisation est une représentation des objets sous forme de graphe (cf figures **2.1** et **2.2**). Les nœuds correspondent à des sous-objets ou à des entités géométriques. Les différents types de nœuds et les propriétés qui leur sont associées permettent de décrire la *forme* des objets. Les arcs reliant les nœuds du graphe décrivent les relations (en particulier les positions et incertitudes associées) entre constituants de l'objet. Ils permettent de décrire la *structure* et la *position* des objets.

[1] ces expressions symboliques sont similaires à celles utilisées dans un programme de niveau objet (cf figure **1.4**)

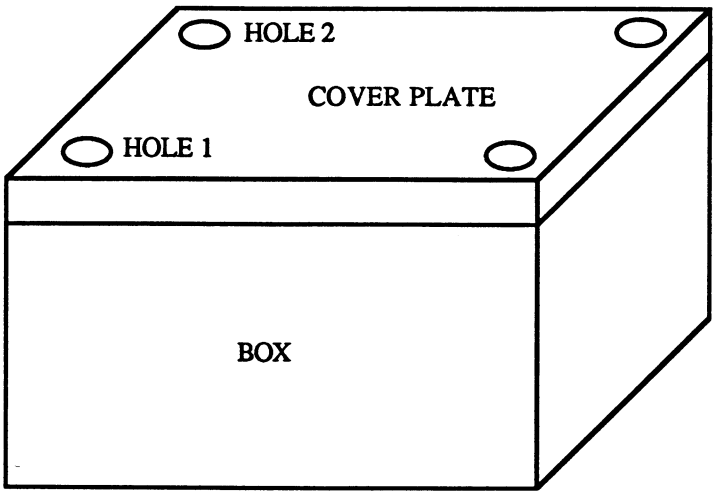


Figure 2.1 : exemple d'objet composé (tiré de [tay76])

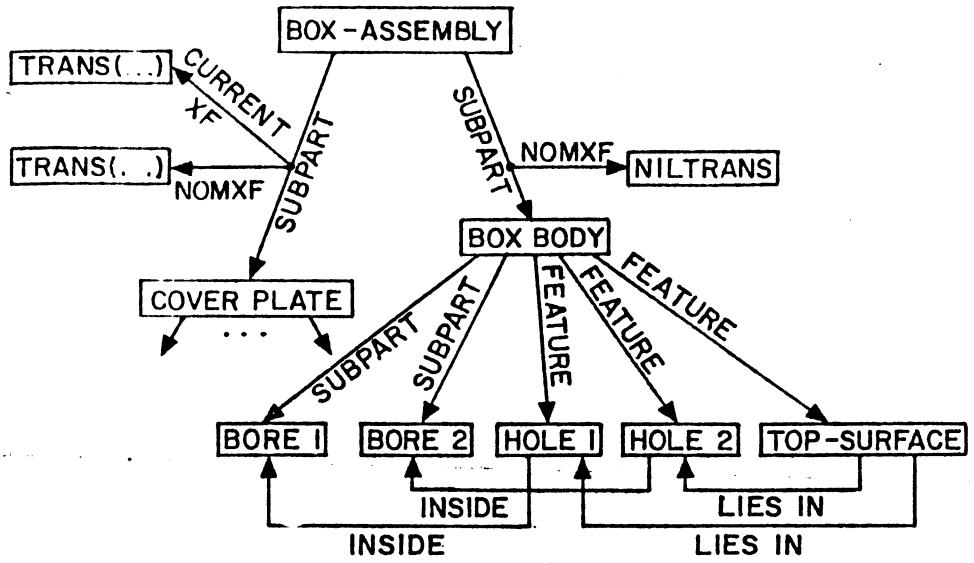


Figure 2.2 : modèle de la boite de la figure précédente

Les positions relatives d'objets ou de parties d'objets sont représentées par des transformations géométriques génériques $T(\lambda_1, \lambda_2, \ldots, \lambda_n)$ dépendant de paramètres $\lambda_1, \lambda_2 \ldots \lambda_n$ qui représentent les degrés de liberté non parfaitement connus. A chaque transformation est associé un ensemble de contraintes du type:

$$f(\lambda_1, \lambda_2, \ldots \lambda_n) > 0$$

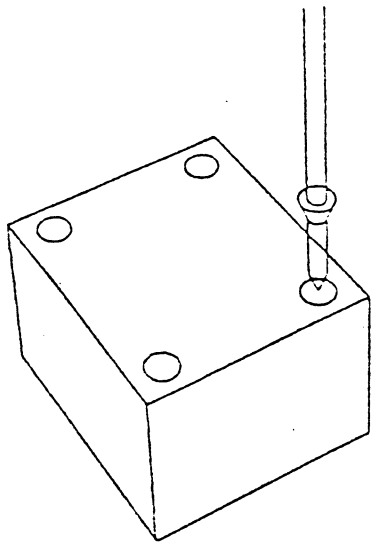


Figure 2.3 : Une boite repose sur une table avec une position et une orientation incertaines. Un robot avec une position et une orientation incertaines tient un tournevis auquel est attachée une vis avec deux degrés de liberté

ou

$$g(\lambda_1, \lambda_2, \ldots \lambda_n) = 0.$$

Ces contraintes permettent de définir le domaine de variation de chacun des paramètres. Dans la pratique, les coefficients λ_i sont considérés comme des *erreurs* sur les degrés de liberté correspondant. La valeur *nominale* d'une transformation, c'est à dire sa valeur non entachée d'erreur est obtenue en annulant chacun des coefficients.

Le modèle est doté de mécanismes de calcul formel qui permettent de passer d'une description symbolique de la position relative d'objets à l'expression $T(\lambda_1, \lambda_2, \ldots, \lambda_n)$ et aux contraintes correspondantes. Ces mécanismes permettent également de faire des calculs de propagation d'erreurs. Ainsi dans l'exemple de la figure 2.3, étant données les positions et incertitudes de la boîte, de la pince du robot, du tournevis et de la vis, le système est capable de calculer la position et l'incertitude associée de la pointe de la vis par rapport à la boîte. Techniquement, les erreurs sont considérées comme "petites" et on fait des approximations linéaires.

1.1.2 Planification des opérations

La méthode générale de planification repose sur une représentation originale de la manipulation. Celle-ci est décomposée en différentes actions: saisie, transport, montage, éventuellement rattrapage d'erreur. Chaque action est représentée par une stratégie prototype sous forme de "squelette de programme". Un squelette est une procédure regroupant tous les mouvements, calculs, tests sur les données de capteurs nécessaires à l'accomplissement de l'action mais dont beaucoup de paramètres restent à spécifier. Pour planifier une manipulation, le système la décompose en une suite d'actions. A chaque action est associé un certain nombre de squelettes possibles. L'applicabilité d'un squelette particulier dépend des caractéristiques géométriques de la tâche et de la valeur de certains paramètres tels que tolérances et incertitudes. Pour une action donnée, le système calcule d'abord les valeurs de certains paramètres spécifiques nécessaires pour la sélection d'un squelette. Parmi les squelettes applicables, il choisit le plus efficace (en général le plus rapide). Enfin il calcule les paramètres non encore instanciés.

Exemple de planification

Prenons comme exemple le cas d'école de l'insertion d'un goujon dans un trou. Le manipulateur doit saisir le goujon et l'extraire d'un support, l'amener au dessus du trou, réaliser l'insertion en se déplaçant verticalement vers le bas.

La manipulation est décomposée comme suit:

- saisie du goujon
- transport jusqu'au sommet du trou
- descente jusqu'à la réalisation d'un contact
- mesure de la distance parcourue verticalement pour déterminer si l'insertion est correcte
- si l'insertion n'est pas faite, recherche locale pour trouver le trou et réaliser l'insertion

La planification est effectuée de la manière suivante:

1. les données nécessaires pour choisir les squelettes applicables sont calculées à partir du modèle géométrique de la tâche, en particulier les positions initiales des objets.

2. les prises possibles sont déterminées. Pour chacune d'elles, on calcule la trajectoire de transport jusqu'au trou.

Comme l'insertion proprement dite se réduit à un mouvement vertical rectiligne, à l'issue de cette seconde phase, le système a déterminé un ensemble de "programmes nominaux" possibles (sans tenir compte des incertitudes).

3. les paires "saisie-approche" sont triées par efficacité décroissante
4. les différentes stratégies sont analysées successivement (en particulier on calcule les incertitudes pertinentes) pour déterminer les raffinements à apporter:
   - si l'incertitude le long de l'axe du goujon est trop grande, on ajoute une opération de "tapping" qui consiste à mesurer la position du goujon dans la pince en recherchant un contact de celui-ci avec le plan de travail à l'extérieur du trou.
   - si l'erreur latérale est trop grande, on ajoute une procédure de recherche locale du trou.
5. en prenant en compte le temps ajouté par les différentes opérations de recherche et de "tapping", on choisit la stratégie optimale.

1.1.3 Discussion

La principale critique que l'on peut faire à l'approche retenue par Taylor est qu'elle permet de faire seulement une analyse "prédictive" des incertitudes. En d'autres termes, elle ne permet que d'estimer par avance les incertitudes susceptibles de se produire. Par contre, elle ne permet pas de caractériser des conditions initiales ou antérieures à une action qui seraient favorables. Le calcul prédictif des incertitudes tel qu'il est mis en œuvre ici permet de produire un programme qui a les qualités suivantes[2]:

- *robustesse*: il sera correctement éxécuté quelles que soient les situations normalement rencontrées,
- *efficacité*: les actions de mesure ("tapping" par exemple) et actions correctrices (recherche du trou) ne seront utilisées que si cela est nécessaire.

Par contre, l'impossiblité de modifier les conditions initiales conduit plus souvent à une situation d'échec: le système est incapable de trouver une stratégie convenable alors

[2]ceci étant vrai *en principe*, sans considérer la réalisation pratique et les méthodes employées

qu'une modification des conditions antérieures permettrait de le faire.

Un autre inconvénient de ce type d'approche vient de l'extrême sensibilité qu'ont parfois les stratégies employées vis-à-vis de petites variations de la géométrie de la tâche (par exemple la présence ou non d'un chanfrein, de l'arrondi d'une arête). Ceci fait qu'un squelette sera souvent adapté à une classe assez réduite de situations géométriques. Ceci rend très difficile la production d'un programme complet à partir de procédures génériques. A moins de disposer d'un nombre extrêmement élevé de squelettes différents (ce qui limite fortement l'intérêt de l'approche), il est souvent difficile de faire correspondre précisément la classe des situations géométriques susceptibles d'être rencontrées avec la classe des situations géométriques auxquelles un squelette est adapté.

1.2 Le système TWAIN

L'approche de Taylor basée sur la manipulation de contraintes sur les positions et les incertitudes a été étendue et améliorée par Brooks [Bro82]. La principale amélioration a été la propagation des contraintes symboliques non seulement "vers l'avant" pour calculer les limites sur les erreurs mais aussi "vers l'arrière" pour restreindre les valeurs initiales de certaines variables et introduire des opérations sensorielles adéquates. L'approche de Brooks est à la base de la structure de contrôle du système TWAIN [LB85].

1.2.1 Présentation générale

Dans ce système, la planification est hiérarchique à deux niveaux. Au niveau supérieur, la description de la tâche est transformée en une suite de squelettes par mise en correspondance de situations géométriques (saisie, transport, lacher, montage). Ces squelettes sont déterminés par un choix dans une librairie (sauf parfois les mouvements fins de montage qui sont calculés ex nihilo). Au niveau inférieur, ces squelettes sont affinés, en commençant par les actions les plus contraintes: mouvements de montage, puis saisie, et enfin transport.

Par rapport au système AL, la planification est ici caractérisée par une gestion élaborée des interdépendances entre actions, permise par l'application d'un principe de moindre engagement.

Chaque squelette de programme est dépendant de certaines quantités physiques:

- des quantités imposées par la tâche
- des quantités dont la valeur est laissée au choix du système
- des quantités qui sont des erreurs et dont on ne connait que des limites.

Des contraintes algébriques portant sur des variables représentant ces quantités sont utilisées pour rendre explicites les interdépendances entre les squelettes. Lors de la planification, chaque module apporte des restrictions sur les variables du squelette. Pour gérer les interdépendances, le système propage ces restrictions sous forme de contraintes à travers les opérations, vers *l'avant* (calcul des contraintes sur les actions postérieures) et vers *l'arrière* (actions antérieures). Les conflits éventuels sont traités par des retours arrières.

1.2.2 Propagation des contraintes

Les contraintes exprimant les restrictions apportées par l'éxécution d'une action sur une autre action sont représentées par des inégalités sur des expressions mettant en jeu des variables formelles. On distingue trois types de variables:

- des variables représentant des paramètres physiques de la tâche. Elles sont imposées par l'environnement.
- des variables dites *de plan.* Elles représentent les paramètres dont la valeur doit être choisie par le système. Représenter un paramètre par une variable formelle permet de raisonner sur des valeurs de ce paramètre tout en reportant le choix de sa valeur exacte. En outre les variables de plan représentent les valeurs *nominales* de paramètres.
- des variables dites *d'incertitude.* Elles représentent des quantités dont on ne connait jamais la valeur. Une variable de ce type représente habituellement *l'erreur* sur un paramètre physique, c'est à dire la différence entre sa valeur réelle et sa valeur nominale. Bien que sa valeur exacte ne soit jamais connue, des limites encadrant ses valeurs possibles sont connues, soit par une connaissance a priori, soit par une série de calculs sur les mouvements et les actions ayant conduit à l'établissement de ce paramètre physique.

1.2.3 Exemple

Pour illustrer ceci, prenons l'exemple classique extrait de [Bro82]. On considère un squelette de programme permettant de faire placer une pièce doit être placée sur une

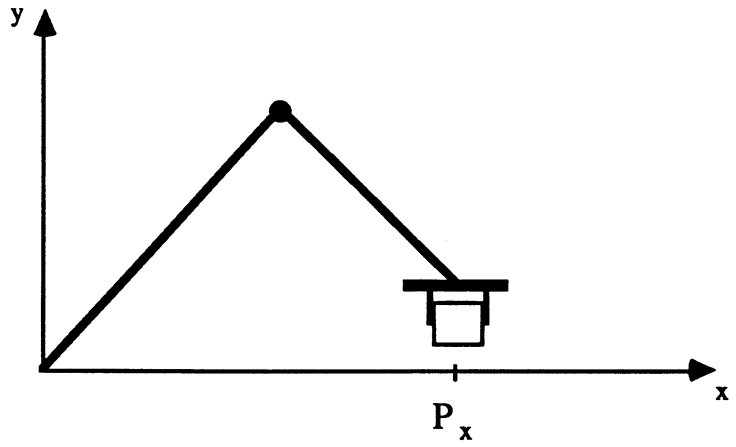


Figure 2.4 : Placement d'un objet par un manipulateur 2D

table à une position p_x par un manipulateur à deux degrés de liberté (fig 2.4).

Du fait de la structure du robot, la précision sur l'axe x est meilleure quand x est grand. Supposons que l'erreur de placement ε soit donnée par:

$$-0.2 + 0.005 * p_x \leq \varepsilon \leq 0.2 - 0.005 * p_x$$

De plus, supposons que son espace de travail est limité par les relations:

$$5 \leq p_x \leq 20$$

Ces limitations imposées par la géométrie de la tâche seront représentées par les contraintes utilisant les variables suivantes:

- *BOX_POSITION*, variable représentant la position réelle de la pièce après placement sur la table,
- *BOX_NOM*, variable de plan représentant la valeur nominale qui sera déterminée par le planificateur,
- *BOX_UNC*, variable d'incertitude représentant l'erreur de placement.

Par définition, les variables sont liées par l'égalité suivante:

$$BOX\_POSITION = BOX\_NOM + BOX\_UNC$$

Les contraintes précédentes s'expriment par les inégalités suivantes:

$$5 \leq BOX\_POSITION \leq 20 \tag{2.1}$$

et

$$-0.2 + 0.005 * BOX\_POSITION \leq BOX\_UNC \leq 0.2 - 0.005 * BOX\_POSITION \tag{2.2}$$

Supposons que pour la suite des opérations il soit nécessaire que la pièce soit positionnée avec une erreur inférieure à ± 0.15. Le système aura propagé vers l'arrière la contrainte suivante:

$$-0.15 \leq BOX\_UNC \leq 0.15 \tag{2.3}$$

Le module de calcul formel associé au système est capable de manipuler les inéquations 2.1, 2.2, et 2.3, pour déduire la contrainte suivante portant sur BOX_NOM:

$$10 \leq BOX\_NOM$$

La satisfaction de cette dernière contrainte implique la satisfaction de 2.3. L'autre possiblité pour le système est d'utiliser un capteur adéquat pour mesurer la quantité $BOX\_POSITION$ avec une précision suffisante.

1.2.4 Discussion

Par rapport au système AL, les principales améliorations portent sur les points suivants:

- nous avons vu aux paragraphes précédents l'intérêt apporté par une propagation des contraintes d'incertitude assez évoluée
- la planification d'une opération par le choix d'un squelette dans une librairie est moins systématique. Cette méthode n'est utilisée que pour les opérations les moins contraintes, et par là les moins sensibles aux incertitudes. Pour les mouvements de montage, on peut partir de primitives de bas niveau.

Dans la littérature publiée par les promoteurs du système, il n'est pas fait mention de détails supplémentaires quant à la mise en œuvre des mécanismes de propagation de contraintes.

La représentation des contraintes par des inégalités algébriques semble intéressante d'un point de vue théorique. En particulier, le formalisme semble tout à fait général puisqu'il permet a priori de traiter le problèmes des interdépendances, qu'elles soient géométriques (dues à la forme des objets) ou dues aux incertitudes (que ce soient les incertitudes de position ou les tolérances sur la forme et les dimensions des pièces). Cependant la mise en œuvre nous semble délicate. En effet avec ce formalisme, toutes les informations intervenant dans la planification doivent être codées sous forme de contraintes algébriques. Pour la moindre manipulation, le nombre de paramètres significatifs est très grand. De plus la représentation de données géométriques par des variables réelles est assez malaisée:

- la représentation de volumes autres que des polyèdres nécessite l'emploi d'expressions non linéaires
- la position d'un objet doit être représentée par au moins six variables. Trois d'entre elles concernent l'orientation de l'objet et, suivant les conventions adoptées interviennent, soit dans des fonctions trigonométriques, soit dans des polynomes de degré supérieur à 2
- les contacts entre objets (point-plan, arête-plan, arête-arête) se traduisent par des contraintes de degré supérieur à 2.

Brooks parle de déduire automatiquement les contraintes d'un modèle géométrique plus classique mais aucune autre précision supplémentaire n'est donnée. De plus nous n'avons pas beaucoup d'informations sur l'efficacité de la méthode de Brooks pour un grand nombre de variables mais nous soupçonnons des performances médiocres et une grande difficulté de mise en œuvre pour des manipulations réelles.

1.3 Une autre approche de vérification/correction de programme

Mazon [Maz87b,Pug88] développe actuellement une approche de vérification-correction de programme qui est comparable à la nôtre. Le contexte est la programmation d'une cellule flexible d'assemblage comprenant un ou plusieurs robots, des capteurs et des

dispositifs péri-robotiques. Le système de programmation automatique procède de la façon suivante:

1. A partir d'une description spatiale et fonctionelle des éléments de la cellule et d'une description symbolique de la tâche, un module engendre de façon hors-ligne un modèle d'éxécution de la tâche.

2. Ce module est ensuite utilisé en ligne pour piloter la cellule.

Lors de la génération du modèle d'éxécution, sont négligées les erreurs de positionnement des pièces et les erreurs sur la forme de celles-ci. Les interdépendances entre actions dûes à la géométrie des pièces sont également négligées. Le modèle d'éxécution doit donc pouvoir être vérifié avant éxécution. Comme pour nous, seules les interdépendances dûes aux incertitudes sont traitées au cours de la phase de vérification. La modélisation des incertitudes de positionnement des objets repose sur la représentation de "petites" transformations par un vecteur de dimension 6 considéré comme une variable aléatoire . Une incertitude est alors définie comme une loi de probabilité gaussienne pour cette variable aléatoire, caractérisée par une matrice de covariance. C'est ce type de représentation des incertitudes que nous avons finalement retenu (cf chapitre 4).

La façon de considérer un modèle d'éxécution comme un programme contraint qui est une séquence alternée d'actions et de conditions est tout à fait similaire à la nôtre.
En ce qui concerne la vérification-correction proprement dite, l'approche de Mazon diffère de la nôtre de la façon suivante: lors de la phase de vérification, quand on calcule un nouvel état E_{i+1}, en fonction de l'état E_i précédent et de l'instruction I_i, on ne calcule pas seulement numériquement le nouvel état, mais on garde également une expression symbolique des incertitudes et positions qui ont été modifiées entre E_i et E_{i+1}. Ces expressions donnent symboliquement les nouvelles valeurs en fonction des précédentes. Cela revient à garder implicitement un historique complet de l'environnement au cours de la manipulation.
Quand une contrainte C_j n'est pas satisfaite, les expressions symboliques des différents termes de C_j en fonction des termes ayant trait à l'état E_{j-1} puis à l'état E_{j-2} ...permet de transformer C_j par reécriture successives de termes, pour faire apparaître des termes décrivant des positions et incertitudes des états précédant E_j.

Il est intéressant de noter que ces deux approches, même si elles se justifient extérieurement de façon très différente conduisent à des résultats très similaires. En effet,

par exemple, C_{j-1} en tant que plus faible précondition de C_j et de l'instruction I_{j-1} (notre méthode) est équivalente à l'expression de C_j dans laquelle on remplace les termes ayant trait à l'état E_{j-1} (la méthode de Mazon). Il en est de même pour les contraintes C_{j-2}, C_{j-3} ... Les deux approches diffèrent plus par leur mise en œuvre: dans la première, on recalcule les expressions dont on a besoin, dans la seconde, on se sert de l'historique de la manipulation que l'on a maintenu.

2 La planification de mouvements par calcul de préimages dans l'espace des configurations

2.1 Présentation de l'approche

Généralement on distingue deux grands types d'approches dans la planification de mouvements avec prise en compte des incertitudes. La première est celle que nous avons vue avec les systèmes AL et TWAIN. L'objet de ce paragraphe est de présenter la seconde, adaptée principalement au calcul de mouvements fins de montage. Cette approche est caractérisée par les points suivants:

- on cherche des "stratégies de mouvements" consistant en une suite de mouvements à compliance,
- les mouvements sont calculés dans l'espace des configurations de la tâche (voir ci-dessous),
- la prise en compte des incertitudes porte sur la commande et, éventuellement, dans une moindre mesure, sur la géométrie de la tâche; cette prise en compte se fait au moment même de la planification.

Le cadre formel de l'approche fut introduit par Lozano-Pérez, Mason et Taylor [LMT84]. Tous les calculs géométriques se font dans l'espace des configurations de la tâche. L'espace des configurations noté C-espace est l'espace engendré par les paramètres de position d'un objet mobile dans un environnement donné (cf figure **2.5**). Ceci permet de ramener le problème du déplacement d'un solide dans l'espace cartésien à celui du déplacement d'un point dans le C-espace. Les actions du robot sont des déplacements à compliance. La loi de commande, de type "generalized damper" (commande en vitesse) relie la cinématique du robot aux forces exercées par celui-ci. L'expression des lois de Coulomb dans le C-espace permet de prendre en compte les phénomènes de friction et de prévoir les glissements ou blocages éventuels. Etant donnés une vitesse commandée v et une zone d'espace à atteindre G, on définit la préimage de G selon v

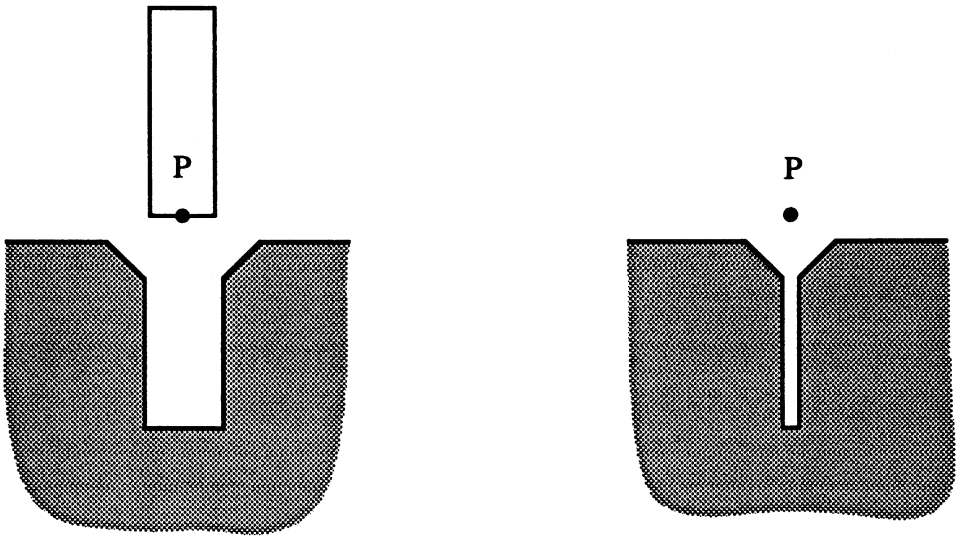


Figure **2.5** : le schéma de droite représente l'espace des configurations associé à la situation géométrique du schéma de gauche

comme la zone d'espace de laquelle le robot est sûr d'atteindre G s'il se déplace suivant v, malgré les incertitudes affectant la commande du robot et sa position initiale. Le calcul d'une stratégie de mouvements de montage se fait en choisissant judicieusement des directions de vittesse successives et en calculant récursivement les préimages du but jusqu'à ce que la position initiale du robot soit contenue dans une préimage (figure **2.6**).

Dans le cadre de ce formalisme, de nombreux travaux ont suivi. Mason [Mas84] consolide le formalisme et prouve la correction et la complétude des algorithmes proposés dans [LMT84]. Erdmann [Erd84,Erd85,Erd86] approfondit l'étude en mettant en évidence deux sous-problèmes distincts: l'atteignabilité et la reconnaissabilité du but. Ce second sous-problème est de déterminer un prédicat évaluable par le manipulateur qui permette de décider si oui ou non on a atteint le but. Il introduit la notion de backprojection, plus faible que celle de pré-image et qui évacue le problème de reconnaissabilité. Il étudie également le problème de la complexité et de l'implantation des algorithmes.

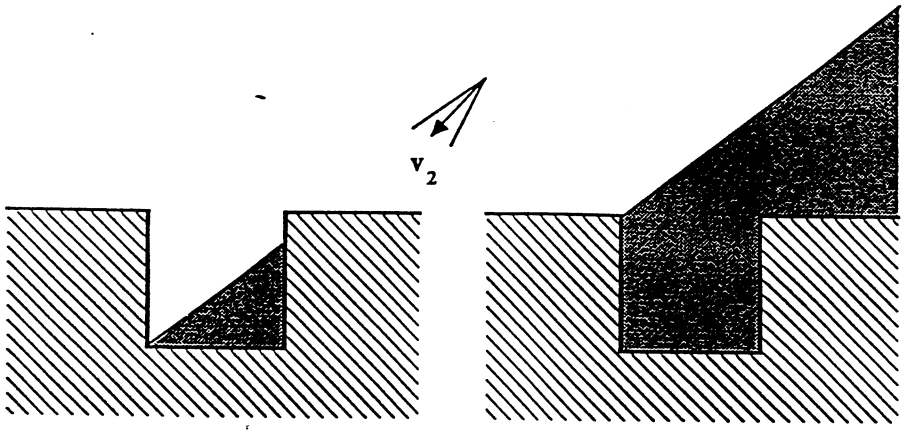


Figure 2.6 : Exemple de préimage

Buckley [Buc87] implante un planificateur utilisant ce formalisme. Il introduit la notion de *"forward projection"* ou "projection vers l'avant" pour vérifier la correction d'une stratégie donnée. Il propose un partitionnement du C-espace pour rendre la recherche de mouvements plus facile. Il étend également le formalisme à une loi de commande de type *"generalized spring"* (commande en position).

Donald [Don86b,Don86a,Don87] reprend également l'approche pour construire des stratégies appelées *"Error Detection and Recovery"* (EDR). De telles stratégies sont plus générales que les stratégies précédentes dites "garanties", dans le sens où elles permettent l'échec, pour peu qu'il soit reconnaissable. Donald introduit la notion d'espace des configurations généralisé (produit cartésien du C-espace et de l'espace formé des paramètres caractérisant la géométrie de la tâche susceptibles d'être entachés d'incertitude). La planification peut se faire également dans l'espace des phases (produit cartésien du C-espace et de l'espace des vitesses).

Actuellement des recherches sont effectuées, tant pour épurer davantage le formalisme, que proposer des algorithmes [Lat88]. Shekhar et Khatib [SK87] proposent une loi de contrôle avec un centre de compliance variable qui peut donner de plus grandes préimages mais dans un espace des configurations de dimension plus élevée.

2.2 Discussion

Le point le plus remarquable de l'approche par calcul de préimages réside dans son espace de recherche, à savoir le C-espace. Plus précisément ce sont les propriétés géométriques qui sont exploitées et, de là, découlent toutes les qualités et limites des

méthodes développées.

La planification de stratégies de mouvements se fait en raisonnant sur des trajectoires et se résout par des arguments purement géométriques. Cela est permis par la nature des actions du manipulateur qui consistent en des mouvements à compliance. La forme des trajectoires engendrées dépend de la géométrie de la tâche mais aussi de la mécanique de la manipulation. Mais grâce à, d'une part la loi de commande liant les forces exercées à la vitesse du manipulateur (generalized damper) ou à sa position (generalized spring) et, d'autre part la modélisation des contacts par les lois de Coulomb, l'aspect mécanique de la manipulation peut être codé par la géométrie des trajectoires.

De plus, les incertitudes sur la position et la vitesse du manipulateur sont relativement facilement prises en compte puisqu'elles s'expriment comme donnant des caractéristiques particulières aux trajectoires (elles peuvent être contenues dans une préimage faible et elles sont sûres d'être contenues dans une préimage forte).

Cependant faisant pendant à toutes ces qualités, des limites de cette approche peuvent apparaître.

Tout d'abord, d'après la définition la plus simple et la plus standard du C-espace, on ne peut planifier, avec ce type d'approche, que les mouvements d'un seul objet mobile dans un environnement de géométrie connue. Or, dans une manipulation interviennent en général de nombreux objets susceptibles d'être déplacés (dont le manipulateur lui-même). Le calcul de mouvements pour plusieurs objets ne peut se faire qu'en augmentant la dimension du C-espace d'un nombre égal à la somme des degrés de liberté de chacun des objets supplémentaires. La complexité des algorithmes utilisant le C-espace croît au moins exponentiellement avec la dimension de celui-ci. On peut remarquer également que les algorithmes proposés ne sont valables que dans le cas de degrés de liberté en translation, cas où la géométrie du C-espace est relativement simple. L'introduction de degrés de liberté en rotation accroit énormément la complexité du C-espace.

De plus, dans un cadre plus général de planification d'actions (et non pas seulement de mouvements), cette approche est adaptée à des actions dont l'effet s'exprime à partir de fonctions du C-espace sur lui-même. Ainsi, par exemple, définir l'effet d'un

déplacement de type generalized damper est équivalent à exhiber la fonction du C-espace sur lui même qui à un point de départ quelconque associe un point d'arrivée. Il n'est pas du tout évident que des opérations classiques dans le cadre de la manipulation comme la saisie puisse s'exprimer de cette façon. Mason [MS85] a étudié des "modèles d'action" pour la saisie. L'espace de travail qu'il propose n'est pas le C-espace mais ce qu'il appelle un "espace d'opérations" [MB86]: il est caractéristique de l'objet saisi et du préhenseur (tant par leur géométrie que par leur coefficient de friction) et a pour axes des paramètres définissant l'action. D'un point de vue expérimental, à notre connaissance, il n'a pas été fait de manipulation utilisant le calcul des préimages ou backprojections associé à une commande réelle de type generalized damper ou generalized spring.

Enfin, la nature du C-espace est telle qu'on ne peut facilement prendre en compte des incertitudes autres que celles sur la position de l'objet manipulé. En particulier, nous avons vu que pour prendre en compte les incertitudes sur la géométrie de la tâche, il fallait augmenter la dimension du C-espace d'une unité par paramètre sujet à incertitudes.

Il est courant dans la littérature de comparer, voire d'opposer, d'une part les approches par raffinements de squelettes et la nôtre et, d'autre part l'approche par calcul de préimages. Pour résumer en quelques mots, on peut dire que dans le premier cas, les interdépendances entre actions sont traitées à l'aide de prédicats portant sur des paramètres du modèle de l'environnement, dans le second cas, elles sont traitées géométriquement grâce au choix d'un espace de travail particulièrement adapté. Nous pensons pour notre part que ces approches sont complémentaires. On peut dire que les approches type raffinement de squelettes sont aptes à prendre en compte les incertitudes de position de nombreux objets mais rendent difficiles la représentation et la manipulation de données géométriques. Les approches type par calcul dans le C-espace, par contre permettent une représentation simple de la géométrie de la tâche quand un seul objet est manipulé à la fois. Nous pensons que la façon la plus efficace de planifier une manipulation est de planifier localement des actions particulières comme les mouvements de montage ou la saisie en se plaçant dans un espace de travail adapté type C-espace et de gérer à un niveau global les interdépendances par propagation de contraintes: c'est ce type de planification qui est proposée dans le système SHARP [Per86,Lau87]

3 Représentation des incertitudes

Plusieurs auteurs ont étudié le problème de la représentation des incertitudes pour la robotique, mais en vue d'une application différente de la nôtre. Nous faisons ici un survol de quelques travaux intéressants.

Smith et Cheesemann [SC86,SCS86,SCS87] ont étudié spécifiquement le problème de représentation applicable dans le cadre de la robotique d'assemblage ou de la robotique mobile. Les objets sont ici représentés par des repères et les positions par des transformations. Une position entachée d'incertitude (notée AT) est représentée par un vecteur aléatoire gaussien (de dimension 6 dans le cas d'un espace tridimensionnel avec rotations) défini par une valeur moyenne et une matrice 6×6 de covariance. Les calculs de composition, d'inversion d'AT sont à base de calcul matriciel sur les covariances et en faisant des approximations au premier ordre. Dans [SCS87], pour tenir compte de l'aspect relatif de la définition de la position d'objets, le modèle est structuré sous forme de graphe quelconque. Le problème du maintien de cohérence du modèle lors de l'ajout d'information (par exemple quand on crée un cycle dans le graphe, le produit des transformations sur un cycle doit être égal à l'identité) est partiellement résolu pour certaines configurations de graphe. Certains cas plus difficiles sont résolus en effectuant une transformation du graphe basée sur une analogie avec des réseaux électriques. Dans [SC86,SCS86], la position de tous les repères est ramenée à un repère de référence unique, ce qui facilite le maintien de la cohérence du modèle. La perte d'information qui en résulte n'est pas analysée. Toutes les méthodes de maintien de cohérence du modèle font appel au filtre de Kalman étendu.

Durrant-Whyte [Dur87a,Dur87b] a étudié le problème de la représentation des incertitudes dans le cadre de l'intégration multi-capteurs. Le modèle qu'il propose est également à base de graphe de repères géométriques liés par des transformations incertaines. Les méthodes statistiques à la base de sa représentation sont étudiées de façon soignée. Elles sont appliquées aux problèmes de composition et d'inversion de transformations et de mise à jour de la position d'un objet à partir de mesures partielles. Le problème du maintien de cohérence du modèle est particulièrement bien étudié: Durrant-Whyte introduit des méthodes d'intégration de mesures qui garantissent la conservation "d'invariants topologiques" (par exemple distances, angles, parallélisme).

De nombreux autres travaux considèrent également le problème de représentation des incertitudes. Dans le contexte de la robotique mobile, [CL85,Ram88,AF88], utilisent une représentation probabiliste des incertitudes. Dans le même cadre, [Bro84,Bro85]

utilisent une représentation ensembliste. Toutes ces représentations, fondées sur un graphe de repères géométriques sont compatibles avec notre approche de correction de programme. Par contre, [Mor88] propose une approche originale incompatible avec notre approche: l'espace des configuration du robot mobile est discrétisé de façon uniforme et on associe à chaque cellule un coefficient de probabilité d'occupation par un obstacle, formant ainsi une "grille d'incertitude". Un modèle de l'espace libre du robot est construit dynamiquement à partir de mesures par des capteurs embarqués. La grille d'incertitude permet de prendre en compte les incertitudes importantes entachant les mesures des capteurs. Cette approche est intéressante quand on travaille dans l'espace des configurations, mais elle n'est pas adaptée au cas où on manipule plusieurs objets. Dans le cas d'un C-espace de dimension élevée, le nombre de cellules doit être également excessivement grand.

Chapitre 3

Sémantique des langages et Vérification-Correction de programme

Pour pouvoir mettre en œuvre l'approche de Vérification/Correction de programme que l'on a présenté au chapitre 1, il est nécessaire de pouvoir décrire parfaitement l'effet de l'éxécution de chaque instruction du langage dans lequel est écrit le programme. Plus précisément, on doit être en mesure de résoudre les deux problèmes suivants:

- étant donné un ensemble d'états initiaux W_i possibles et une instruction quelconque I, calculer l'ensemble W_j de tous les états possibles résultant de l'éxécution de I.
- étant donnée une condition C que doit satisfaire l'état de l'univers et une instruction I, trouver l'ensemble maximal d'états initiaux tel que C soit satisfaite après éxécution de I.

En informatique théorique, ces deux problèmes ont été abordés dans le cadre de l'étude de la sémantique des langages de programmation. Cependant, les études n'ont concerné que des langages de programmation "classique"[1]. Pour les langages de programmation de robots, ce problème n'a jamais été explicitement abordé à notre connaissance, si ce n'est dans [Zie85] et le formalisme utilisé dans ce travail n'est pas adapté à la

[1]Nous sous-entendrons par programmation "classique", la programmation dans un langage procédural des années 60-70 tels que FORTRAN, ALGOL ou PASCAL, la plupart des langages de programmation de robots étant inspiré de ces langages.

vérification-correction de programme. Après avoir exposé rapidement ce qui est fait dans le cadre de la programmation classique, nous analyserons les spécificités des langages de programmation de robots de niveau effecteur. Nous proposerons enfin deux types de description, inspirées de méthodes décrites auparavant, l'une étant adaptée à la vérification, l'autre à la correction.

1 Sémantique d'un langage de programmation

D'après le dictionnaire encyclopédique Larousse, la sémantique, d'un point de vue linguistique est *"l'étude du sens des mots et des énoncés"*. Pour un langage de programmation, définir sa sémantique consiste à donner un sens à ses formules. Cela peut se faire de façon peu formelle en langue naturelle. Cependant l'émergence d'un grand nombre de langages, utilisant parfois des notations communes avec des sens différents (par exemple les utilisations diverses du signe = en FORTRAN, C ou PASCAL), a entrainé un besoin de description de la sémantique plus formelle et plus précise. Cette description se fait grâce à un ensemble de techniques qui permettent de passer de la syntaxe du langage à des objets mathématiques dont le sens est "évident" (ou sous-entendu comme tel).

Outre son utilité lors de l'implantation d'un compilateur standard, cette description est une aide précieuse pour l'écriture de programmes corrects. Un programme est écrit dans le but de calculer une certaine fonction f à l'aide d'une séquence *finie* de calculs. Il transforme un ensemble de données D en un ensemble de résultats R. Intuitivement, un programme est correct s'il satisfait les deux conditions:

- le programme doit pouvoir s'éxécuter pour toute donnée de D, c'est-à-dire qu'il ne doit pas y avoir d'instruction dont l'éxécution conduise à une erreur (par exemple division par zéro, ou indice de tableau hors limites).
- pour toute donnée initiale $d \in D$, le programme doit se terminer (ne pas "boucler") et rendre le résultat escompté $f(d)$.

Nous sommes plus concernés par le premier point, l'éxécutabilité du programme, le second étant davantage du ressort du planificateur qui aura produit le programme que nous avons à vérifier. Nous nous intéressons aux méthodes utilisées en programmation classique pour en déduire des méthodes valables dans le cadre de la programmation des robots.

Il existe plusieurs méthodes de description de la sémantique de langages informatiques [Liv78]. Nous nous intéressons aux deux types de méthodes suivants: les

méthodes dites opérationnelle (ou interprétative) et axiomatique. Les premières sont adaptées au calcul des ensembles successifs d'états possibles $\mathcal{W}_i$ lors de la phase de vérification. Les dernières sont sont adaptées au calcul des contraintes sur les états antérieurs lors de la phase de correction.

1.1 Sémantique opérationnelle

Le principe de cette approche est de définir un automate ou "machine abstraite", caractérisée par un certain nombre d'états possibles et un certain nombre d'opérations de base. La machine est définie en spécifiant les transitions entre états sous l'effet des différentes opérations. L'idée est que, bien que la machine puisse être complètement irréaliste d'un point de vue pratique, elle soit suffisamment simple pour que l'éxécution des opérations de base ait un effet totalement dépourvu d'ambiguité. La description de la sémantique du langage de programmation consiste alors à établir une correspondance entre les instructions du langage et les opérations de base de la machine. Pour déterminer l'effet de l'éxécution d'un programme, il suffit de transcrire le programme en une suite d'opérations sur la machine abstraite et de l'éxécuter formellement pas à pas. La sémantique des langages PL/1 et Algol 68 a été définie de cette façon [LW71,vWea75].
Cette méthode est bien adaptée au calcul de l'effet de l'éxécution d'un programme. La machine abstraite est un modèle de "l'état mémoire" du calculateur (valeur actuelle de chacun des identificateurs du programme). Dans le cas de la programmation d'un robot, l'équivalent de cet état mémoire est, outre la valeur des identificateurs (c'est à dire implicitement le modèle théorique que le robot a de son environnement), l'environnement réel lui-même (position et forme des différents objets). En vue de la vérification de programme, nous donnerons une description de type "opérationnel" de l'effet de chaque instruction (cf chapitre 5).

1.2 Sémantique axiomatique

Le principe de cette approche est de définir ce que fait une instruction ou un groupe d'instructions du langage en précisant comment cette instruction ou ce groupe transforme un prédicat caractérisant un sous-ensemble ou l'ensemble des objets manipulés par le programme. Cela se fait en associant à chaque instruction un "axiome" qui établit:

- soit quelles conditions sont vraies après éxécution de l'instruction, en fonction de ce qui est vrai avant.

- soit quelles conditions doivent être vraies avant éxécution pour qu'une condition donnée soit vraie après.

Plus précisément, un axiome est un énoncé noté E{P}S où E et S sont des conditions, P une instruction (ou une séquence d'instructions) dont la signification est *"si E est vraie avant éxécution de P, alors si P se termine, S est vraie après éxécution de P"*. On appellera E *précondition* et S *postcondition.*

Exemple:

Axiome de l'affectation:
Etant donnée une instruction d'affectation $x := expr$ et une postcondition S, l'énoncé E$\{x := expr\}$S est valide, où E est obtenue à partir de S par substitution à toutes les occurences de la variable x de l'expression *expr*.

Application: Cet axiome permet d'établir l'énoncé suivant:

$$(y + 2 = 2)\{x := y + 2\}(x = 2)$$

La donnée d'un axiome pour chaque instruction du langage définit sa sémantique. A l'aide de *règles d'inférence*, on peut déduire des théorèmes qui seront de nouveaux énoncés E{P}S.

Exemples de règle:

1. *règle de la précondition:*
   si E{P}S
   et E' $\Rightarrow$ E
   alors E' {P} S
2. *règle de la composition de deux instructions* (règle du ;):
   si E {P} F
   et F {Q} S
   alors E {P;Q} S

Supposons qu'on ait un programme P quelconque dont les données érifient une condition E, et dont on voudrait que les résultats satisfassent une condition S. Si on peut établir l'énoncé E{P}S à l'aide des différents axiomes et règles, on dira qu'on aura fait la

preuve de correction partielle[2] de P par rapport aux conditions E et S. C'est là l'utilité principale de cette approche pour nous: elle fournit des outils formels permettant de prouver la correction de programmes.

Une façon très efficace de formuler un axiome E{P}S est de donner ce qu'on appelle la *plus faible précondition* E_0 associée à une instruction P et une postcondition S. Cette condition est caractérisée par le fait que *toutes* les conditions E telles que E{P}S sont telles que E implique E_0 (réciproque de la règle de la précondition).

Cette propriété, qui sera étudiée plus en détail par la suite, sera utilisée lors de la phase de *correction* introduite au paragraphe 3.3 du chapitre 1. En effet, dans le formalisme de la sémantique axiomatique, les conditions C_j^i produites lors de la propagation ascendante sont en fait des préconditions telles que:

$$C_j^i\{I_i;\ldots;I_j\}C_j$$

L'axiome de l'affectation, tel qu'il a été donné ci-dessus, permet de calculer une précondition E en fonction d'une postcondition S donnée.

Cette approche axiomatique de la sémantique a été introduite principalement par Hoare [Hoa69]. L'alphabet du système formel défini par les axiomes et les règles d'inférence est composé de symboles logiques ($\wedge, \vee, \neg, \Rightarrow, \ldots$) et d'éléments du langage. Les conditions sont des prédicats du premier ordre. En plus des axiomes définissant la sémantique du langage, on utilise comme axiomes les théorèmes logiques du calcul des prédicats, et des axiomes précisant les propriétés du domaine de discours (entiers, réels, booléens, ...).

L'adaptation à un langage de programmation de robots nécessite de préciser les propriétés du domaine sur lequel on travaille (essentiellement des données géométriques précisant la position des objets et les incertitudes associées) et les types de conditions qu'il est intéressant de manipuler.

[2]Il y a ici un petit problème de vocabulaire: ce qui s'appelle en algorithmique "preuve de correction partielle" correspond à ce que nous appelons vérification, alors que nous entendons par "correction" (l'action de corriger) la phase de modification du programme

2 Particularités d'un langage de programmation de robots

Les approches opérationnelle et axiomatique ont été développées dans le cadre de langages de programmation classique. Leur adaptation à un langage de programmation de robots nécessite l'analyse des spécificités et des différences de ces deux types de langage.

La différence la plus immédiate vient évidemment des champs d'application. Les langages de programmation classiques sont bâtis implicitement sur le concept de machine de Von Neumann [Bac78] où l'information (au sens large), traitée par le programme est codée sous forme d'un certain nombre de bits résidant dans des registres mémoire. Toutes les opérations effectuées par le programme sont une combinaison plus ou moins complexe d'opérations de transfert de données entre la mémoire et un processeur, et d'opérations booléennes au sein du processeur. Même si la description microscopique physique du fonctionnement d'un calculateur est éminemment complexe, la description du comportement macroscopique par les notions de registre mémoire, d'opérations booléennes et à un plus haut niveau par les notions de variables, de types et d'opérations arithmétiques se formalise par des notions mathématiques relativement simples.

Un langage de programmation de robots est un langage informatique à part entière et certaines opérations qu'il décrit portent également sur des objets bien formalisés mathématiquement. Cependant, ce qui en fait sa raison d'être, ce sont les opérations de commande du bras manipulateur qui, elles, agissent sur *l'environnement physique* du robot. La description de cet environnement et des opérations n'est plus à la base un problème de mathématiques mais de physique, essentiellement de mécanique. La description des phénomènes microscopiques qui entrent en jeu lors d'une manipulation est extrêmement complexe mais celle du comportement macroscopique l'est également. En particulier, on se trouve confronté au problème des incertitudes dans la modélisation des états de l'univers et dans l'effet des opérations.

La différence au niveau des champs d'application des deux types de langage se retrouve de manière parallèle dans la différence des instructions. Outre les instructions commune avec les langages classiques (contrôle algorithmique, affectation, ...) les langages de programmation de robots possèdent des types de données et des instructions spécifiques [Loz82] que nous allons examiner dans ce chapitre. Nous prendrons comme

cas particulier le langage LM, enrichi, par rapport à sa version commerciale, des instructions de mouvement à compliance définies dans [Gan86]. Nous allons décrire en langage naturel l'effet des différentes actions et montrer les difficultés que l'on a *a priori* pour décrire la sémantique d'un langage de niveau effecteur tel que LM. Rappelons les grandes caractéristiques de ce langage:

- *Manipulation de données géométriques:* LM possède des types adaptés à la représentation et à la manipulation de données géométriques. Ce sont les types *REPERE*, *TRANSFORMATION*, *VECTEUR*, et les opérations associées. Il existe également des instructions *LIER* et *DELIER* qui permettent d'établir des liens rigides entre repères.
- *spécification des mouvements:* les instructions de mouvement permettent de décrire sa géométrie et sa cinématique ainsi que l'interaction du mouvement avec les capteurs.
- *action de l'outil terminal:* dans le cadre de la manipulation, nous ne considérerons que les instructions d'ouverture et de fermeture de pince.
- *perception et communication avec l'environnement:* En langage LM, la lecture de données fournies par des capteurs et la communication avec l'environnement se fait de plusieurs façons:
  - il existe des "fonctions capteur de force" qui rendent comme résultat les composantes du torseur de force appliqué sur l'extrémité du robot
  - il existe une interface normalisée appelée ESVOIE qui permet d'activer des routines externes à l'interpréteur LM et de lire des données. Cela peut servir à communiquer avec un capteur muni de son propre processeur de traitement, en particulier un capteur de vision.

2.1 Structures de données et opérations spécialisées

Au sein du langage LM, la représentation d'un objet solide se fait en lui associant un repère cartésien. La position de cet objet dans un référentiel est représentée par la position du repère associé. Elle s'exprime par la transformation qui amène le référentiel sur le repère lié à l'objet. Cette transformation se décompose en une translation et un rotation. Cela conduit à définir au sein du langage LM des variables de type *REPERE* et de type *TRANSFORMATION*. Un repère particulier appelé *STATION* sert d'unique référence à tous les repères. La valeur d'une variable de type *REPERE*

est la transformation géométrique définissant sa position par rapport à *STATION*. A ces deux types s'ajoutent le type *VECTEUR* qui permet de définir facilement des transformations. Les fonctions *TRANSLAT* et *ROT* permettent de définir des transformations.

A ces types de données spécialisés, sont associées de nombreuses fonctions qui permettent entre autres de calculer:

- la composition, l'inversion de transformations,
- étant donnés deux repères, la transformation faisant passer de l'un à l'autre,
- le vecteur de translation et les paramètres de la rotation d'une transformation donnée,
- l'image d'un vecteur et l'image d'un repère par une transformation,
- le produit d'un réel par un vecteur, l'addition, le produit scalaire et le produit vectoriel de deux vecteurs.

2.1.1 Liaisons de repères

Le langage LM, ainsi que la plupart des langages de niveau effecteur évolués possède une commande *LIER* permettant de rendre solidaires plusieurs repères. Ainsi l'instruction:
`LIER REP_1 A REP_2`
aura pour conséquence qu'une modification ultérieure de la valeur de l'un des repères REP_1 ou REP_2 (par affectation ou lors d'un déplacement) sera automatiquement appliquée à l'autre repère de telle sorte que la transformation faisant passer de l'un à l'autre reste constante.
La commande *DELIER* permet de rendre indépendants des repères auparavant liés.
L'effet de l'instruction *LIER* est illutré par l'exemple suivant tiré de [MM85] (voir figure **3**.1):

```
PROGRAMME TLIER;
REPERE REP1, REP2;
DEBUT
REP1 := STATION * TRANSLAT(VX,100.0);
REP2 := REP1 * TRANSLAT(VY,200.0);
CO REP1 et REP2 sont initialises, et sont pour le moment
        independants l'un de l'autre;
```

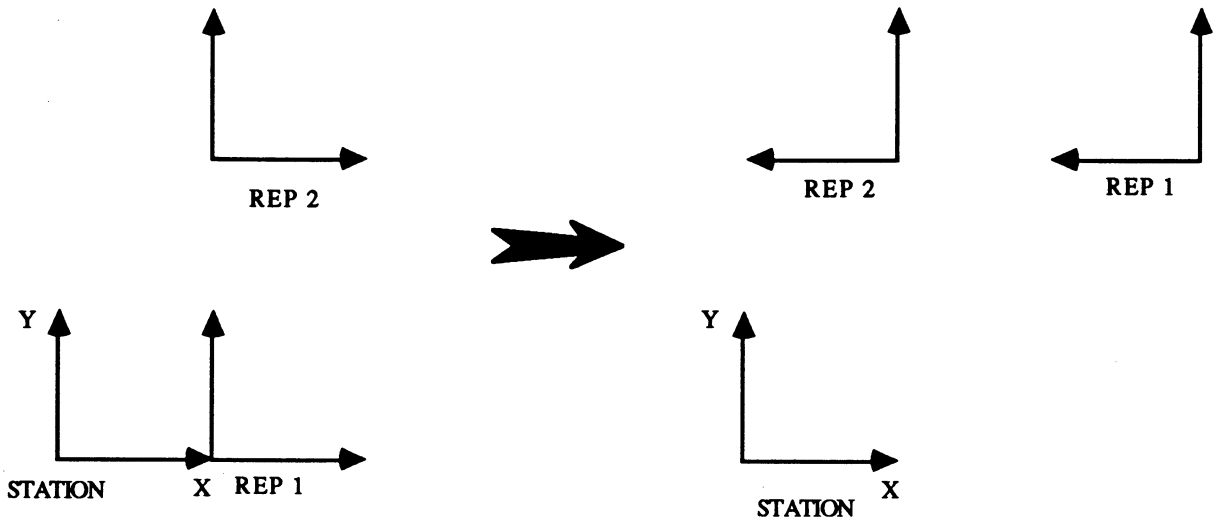


Figure 3.1 : effet de l'instruction LIER

```
LIER REP1 A REP2;
ECRIRE "VALEUR DE REP1", REP1;
ECRIRE "VALEUR DE REP2", REP2;
CO cette liaison ne modifie ni la position de REP1, ni
        celle de REP2;
REP2 := REP2 * ROT(VZ,PI/2.0);
CO la valeur de REP2 est bien sur modifiee, mais celle
        de REP1 l'est aussi puisqu'il lui est lie;
ECRIRE "NOUVELLE VALEUR DE REP1", REP1;
ECRIRE "NOUVELLE VALEUR DE REP2", REP2;
FIN;
```

La trace d'éxécution de ce programme est la suivante:

```
VALEUR DE REP1
   1.  0.  0.  100.
   0.  1.  0.    0.
   0.  0.  1.    0.

VALEUR DE REP2
   1.  0.  0.  100.
```

```
   0.  1.  0.  200.
   0.  0.  1.    0.

NOUVELLE VALEUR DE REP1
   0. -1.  0.  300.
   1.  0.  0.  200.
   0.  0.  1.    0.

NOUVELE VALEUR DE REP2
   0. -1.  0.  100.
   1.  0.  0.  200.
   0.  0.  1.    0.
```

2.1.2 Cas d'utilisation de la liaison

De façon pratique, on peut distinguer trois utilisations privilégiées de la liaison de repères que nous allons détailler ci-dessous.

Définition d'un objet par plusieurs repères

Il est souvent intéressant d'attacher plusieurs repères à un même objet, ce qui permet de décrire cet objet avec toutes les positions caractéristiques qui lui sont associées. Par exemple, à l'objet de la figure **3.2**, on associe un repère POS_A servant à définir sa position, et deux repères $TROU\_1$ et $TROU\_2$ qui servent à définir la position des deux trous cylindriques. Pour que la valeur de chacun des repères appartenant à un même objet soit mise à jour automatiquement, lors d'un déplacement par exemple, on lie tous ces repères.

Création et destruction de structures rigides

Au cours d'une manipulation, on crée souvent des structures rigides en liant physiquement plusieurs objets. Ces structures peuvent être temporaires: c'est le cas lors de la saisie où on établit une liaison rigide entre l'objet et le préhenseur. Elles peuvent être également définitives comme par exemple lorsqu'on assemble rigidement deux objets. Pendant l'existence de ces structures rigides, il est intéressant de lier les repères associés aux différents objets participant à la structure.

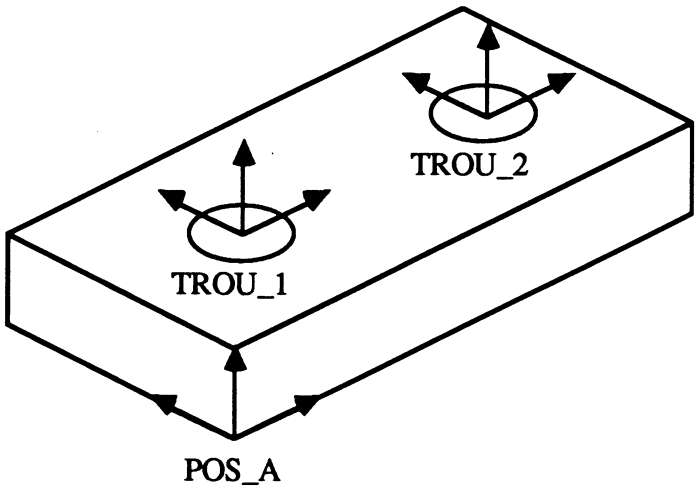


Figure 3.2 : Objet défini par plusieurs repères

Calibration

Un problème de calibration assez fréquent se pose lorsque les objets participant à la manipulation sont définis par rapport à des repères de référence différents. Le but de la calibration est alors d'établir la correspondance entre les différents repères de référence. Prenons l'exemple de la figure 3.3. Le robot doit prendre des pièces sur une table distincte de son propre support. La position du robot est définie par rapport au repère $WSPACE$ solidaire du support du robot. La position des objets est définie par rapport au repère $TABLE$. Un procédure de calibration possible est de mesurer la position de $TABLE$ par rapport à $WSPACE$. La position de tous les objets par rapport à $WSPACE$ sera alors connue. Pour cela, il faut successivement:

- lier les repères associés aux objets au repère $TABLE$,
- mesurer et mettre à jour par une instruction d'affectation la valeur de $TABLE$,
- délier les objets du repère $TABLE$.

2.1.3 Discussion

L'examen des cas d'utilisation de la liaison de repères montre une différence dans la dépendance effective des repères entre eux:

- dans les deux premiers cas d'utilisation, la liaison de repère traduit une solidarité physique: d'un point de vue mécanique, les repères liés appartiennent au même

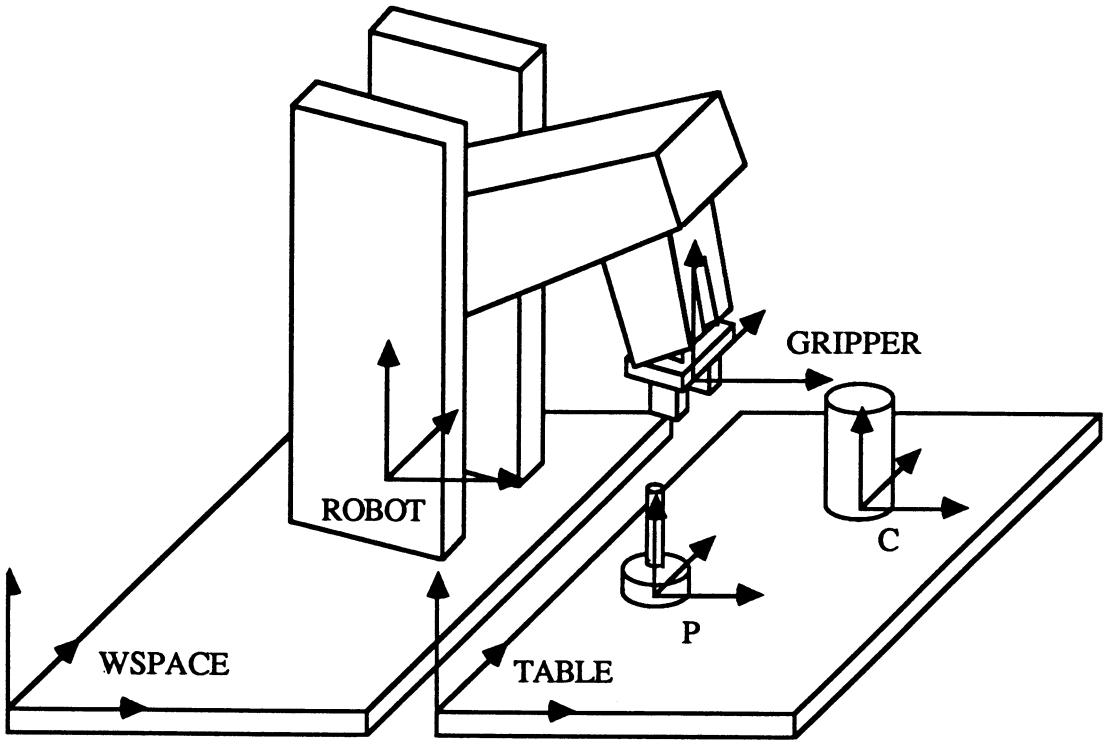


Figure 3.3 : Utilisation de la liaison de repères pour la calibration

solide. Il est normal que la modification de l'un quelconque des repères se propage sur chacun des autres. En outre, il n'y a pas de repère qui joue de rôle particulier.

- dans la troisième utilisation, par contre, la liaison traduit une solidarité plus symbolique: elle n'existe que parce que la position des objets est définie par rapport à la table. Les liaisons sont unilatérales et le repère *TABLE* joue un rôle privilégié. Si on le modifie, les autres doivent l'être également, mais si on modifie un des repères attaché à l'un des objets, il n'y a pas lieu de modifier *TABLE* ni chacun des repères attachés aux autres objets.

Le fait que dans LM, la valeur d'un repère ne puisse faire référence qu'au repère *STATION* et que l'instruction *LIER* établisse un lien bilatéral entre les repères fait que le recours à la liaison pour une utilisation proche du troisième cas est un artifice qui n'est pas très propre d'un point de vue sémantique. Nous estimons qu'il serait préférable que la valeur d'un repère fasse référence à un autre repère qui puisse être différent de *STATION* rendant marginal l'emploi de l'instruction *LIER*. C'est ce que nous faisons dans le langage que nous avons retenu (chapitre 5).

De plus, dans LM, la gestion simple des liens entre repères induit une ambiguité dans l'effet des instructions *LIER* et *DELIER*. Ainsi par exemple si on considère les deux séquences suivantes:

```
LIER REP_1 A REP_2;
LIER REP_2 A REP_3;
LIER REP_1 A REP_3;
DELIER REP_2 DE REP_3;
```

et

```
LIER REP_1 A REP_2;
LIER REP_2 A REP_3;
DELIER REP_2 DE REP_3;
```

La dernière instruction de chaque séquence est la même, mais elle n'a pas le même effet. Dans la première séquence, elle n'a aucun effet puisque les deux repères REP_2 et REP_3 restent liés par l'intermédiaire de REP_1. Dans la seconde séquence, elle réalise effectivement une destruction de lien.

Le dernier point à souligner est que l'existence de liens entre repères modifie fortement la sémantique des instructions d'affectation et de déplacement puisque ces instructions ne modifient pas seulement les valeurs des identificateurs qu'elles ont en paramètres mais aussi la valeur de *tous les identificateurs* qui leur sont liés, directement ou indirectement.

2.2 Instructions de mouvements

2.2.1 Effet sur les positions nominales

Les instructions permettant de commander les déplacements du robot sont parmi les plus importantes. Elles permettent de modifier les positions du manipulateur et celles des objets tenus dans la pince. Une instruction générale de déplacement peut s'écrire:

deplacer *OBJET* **selon** *TRAJECTOIRE-GEN*

Où:

- *OBJET* représente l'objet que le robot doit déplacer. Il est sous-entendu que l'objet est déplaçable, c'est-à-dire préalablement lié au robot par l'intermédiaire du préhenseur.
- *TRAJECTOIRE-GEN* (pour "trajectoire-généralisée") spécifie le mode de déplacement: forme de la trajectoire et position d'arrivée.

Les principales façons de spécifier un déplacement sont les suivantes:

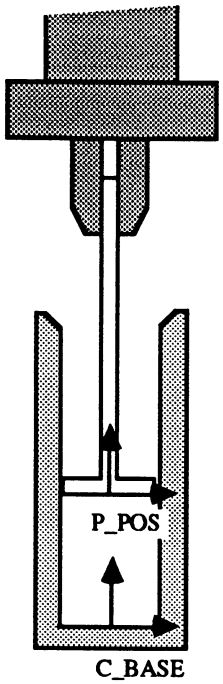
1. la plus simple consiste à donner une position d'arrivée explicite, assortie éventuellement d'une trajectoire (ligne droite, circulaire, trajectoire prédéfinie ...). Si on ne donne pas de trajectoire, le manipulateur en utilise une par défaut, fonction uniquement des positions de départ et d'arrivée (déplacement dit "en mode libre") (cf figure **3**.4).
2. la position d'arrivée peut être implicite: dans ce cas, on donne un ordre de terminaison du déplacement, en général une condition d'arrêt sur des données capteurs (figures **3**.5 et **3**.6). La trajectoire peut elle-même être donnée explicitement (mouvements gardés, cas de la figure **3**.5) ou implicitement. Dans ce cas, la trajectoire est asservie sur des données de capteurs [Gan86] (mouvements à compliance, figure **3**.6).

L'intérêt des spécifications de mouvements implicites réside dans le fait que ces mouvements sont moins dépendants de la géométrie de la tâche au moment de l'éxécution, en particulier, ils sont moins sensibles aux incertitudes: la trajectoire effectivement réalisée dépend des données fournies par les capteurs pendant le mouvement et non pas uniquement d'un modèle a priori de la tâche. Cependant, leur effet est plus difficilement calculable.

Les trois instructions de déplacement donnés en exemple dans les figures **3**.4 à **3**.6 ont théoriquement le même effet, à savoir le déplacement du piston à l'intérieur du cylindre. Dans le cas des deux dernières, cet effet ne peut-être calculé que si on connait la géométrie particulière de la tâche. Si on ne dispose que de la position des objets sous forme de repères et de transformations, on ne peut prédire l'état de l'univers après éxécution de ces instructions.

2.2.2 Effet sur les incertitudes de position

La précision d'un manipulateur n'est jamais parfaite et lors de déplacements, la position d'arrivée est toujours entachée d'incertitudes. Celles-ci dépendent du robot et de sa commande mais aussi du mode de déplacement.

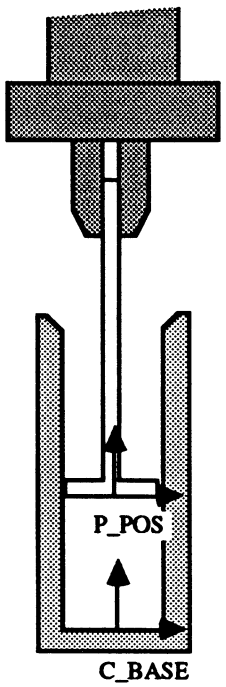


deplacer P_POS a C_BASE en cartesien

Figure 3.4 : Instruction de déplacement avec position d'arrivée et trajectoire explicites

Position d'arrivée explicite

L'incertitude de position de l'objet après déplacement est celle du manipulateur composée avec l'incertitude de position de l'objet dans la pince. L'incertitude de position finale du manipulateur est caractéristique de celui-ci et dépend éventuellement de la position d'arrivée (par exemple pour un robot à axes rotoïdes, elle est plus importante sur les frontières extérieures de l'espace de travail). Généralement, on la considère constante sur l'espace de travail.
Il faut distinguer le cas de déplacements où la position d'arrivée est donnée dans l'absolu, c'est-à-dire par rapport au repère de référence de l'environnement (cas du *DEPLACER A* de LM) et le cas où elle est donnée relativement à la position initiale du manipulateur (cas du *DEPLACER DE*). Dans ce second cas, il apparait que l'incertitude de position du manipulateur est égale à l'incertitude de position initiale incrémentée d'une incertitude de déplacement.



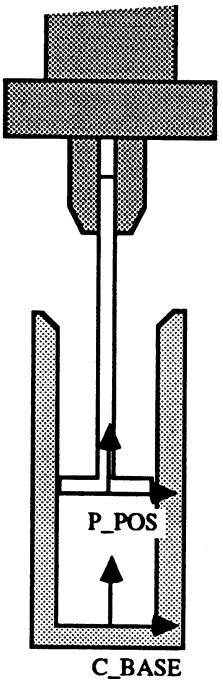
deplacer P_POS suivant (VZ,-1) jusqua FZ > 5.0

Figure 3.5 : Instruction de déplacement avec position d'arrivée implicite, trajectoire explicite

Position d'arrivée implicite

Le but d'une instruction de mouvement gardé ou à compliance est d'établir un contact entre l'objet tenu dans la pince et un objet cible de l'environnement. A l'issue du mouvement, l'incertitude de position du robot est la même que pour n'importe quel autre déplacement. De plus, et c'est là l'intérêt de ce type de mouvement, l'existence d'un contact garantit une incertitude nulle pour certains degrés de liberté sur la position de l'objet saisi par rapport à la cible. Cela peut permettre de diminuer:

- l'incertitude de position de la cible si on connait bien la position de l'objet saisi par rapport au préhenseur
- l'incertitude de position de l'objet dans le préhenseur si on connait bien la position de la cible.




```
deplacer P_POS vers (VZ, -1)
      en maintenant FX=0, FY=0, MX=0, MY=0,
      jusqua FZ  >  5.0
```

Figure 3.6 : Instruction de déplacement avec position d'arrivée implicite, trajectoire implicite

2.3 Action de l'outil terminal

2.3.1 Effet sur les positions nominales

Dans le cadre de la manipulation, l'outil terminal est un préhenseur[3] qui permet de saisir et de lâcher des objets. En langage LM, les deux instructions permettant d'actionner l'outil terminal sont *"FERMER PINCE"* et *"OUVRIR PINCE"*. Théoriquement, la position du manipulateur est calculée pour que les actions de saisie et de lâcher n'occasionnent pas de déplacements de l'objet car ces déplacements sont difficiles à prévoir et à contrôler. Le seul effet nominal de ces instructions est d'établir ou de détruire des liaisons bilatérales entre l'objet et le préhenseur.

Comme leurs noms le laisse supposer, ces instructions ne permettent pas de commander explicitement une action de prise ou de lâcher mais seulement de donner une géométrie

[3]On ne considerera ici que le type de préhenseur le plus répandu qui est une pince à deux mors parallèles

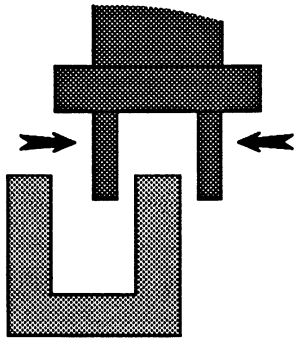


Figure **3.7** : Un effet de la fermeture de pince

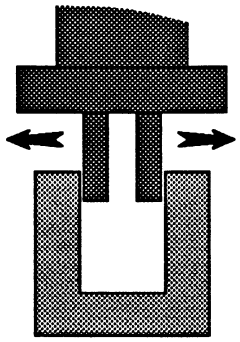


Figure **3.8** : Un effet de l'ouverture de la pince

particulière à la pince (mors serrés ou écartés). En général, une instruction *FERMER PINCE* correspond à une opération de saisie, et *OUVRIR PINCE* à un lâcher. Ceci est dû à la géométrie des objets habituellement manipulés mais n'est pas systématique. Ceci entraine une certaine ambiguité dans la sémantique des instructions *"FERMER PINCE"* et *"OUVRIR PINCE"*.

Dans les figures **3.7** et **3.8**, on peut voir l'éxécution de deux séquences d'instructions différentes ayant exactement le même effet (saisie du cylindre), alors que ne connaissant pas la géométrie locale de la tâche, on aurait tendance à penser que la première correspond à une saisie et la seconde à un lâcher.

2.3.2 Effet sur les incertitudes

Saisie

A cause des erreurs de position du préhenseur et de l'objet, une action de saisie s'accompagne de déplacements parasites de l'objet. Ces déplacements ont pour effet de modifier l'incertitude de position de l'objet par rapport à son support et par rapport au préhenseur. L'établissement d'un contact plan entre le préhenseur et l'objet rend nulle l'incertitude de position relative de l'objet par rapport au préhenseur suivant un degré de liberté en translation (normale au plan des mors) et deux degrés de liberté en rotation (d'axes dans le plan des mors).

Lâcher

De la même façon, à cause des erreurs de position du préhenseur et du support sur lequel on lâche l'objet, une action de lâcher s'accompagne de mouvements parasites qui ont pour effet de modifier les incertitudes de position de l'objet par rapport au préhenseur et au support. Suivant la nature du contact entre l'objet et le support, l'incertitude de position de l'objet par rapport au support sera nulle suivant certains degrés de liberté.

2.4 Perception de l'environnement

La perception de l'environnement par l'intermédiaire de capteurs est fondamentale en manipulation pour pouvoir adapter le comportement du robot aux situations différentes susceptibles de se produire. Malheureusement, il existe une très grande variété de capteurs et surtout une variété encore plus grande de traitements des données brutes fournies par un capteur pour extraire l'information utile au robot. En conséquence, hormis éventuellement les fonctions rendant les composantes du torseur de force appliqué à l'extrémité du robot, il n'existe pas de fonction standard permettant une perception de l'environnement. En particulier, il n'existe pas de fonction rendant une information standard issue d'un capteur de vision. Dans la pratique, le programmeur développe parallèlement à la manipulation, un programme externe de traitement vision qui lui fournira des résultats spécifiques dont il aura besoin et avec lequel il communiquera par l'interface normalisée ESVOIE. La sémantique des opérations de communication avec l'environnement ne peut donc absolument pas être définie.

2.5 Conclusion

Cette analyse des principales instructions caractéristiques d'un langage de programmation de robots nous amènent à formuler deux conclusions.

La première concerne les méthodes utilisées en informatique classique. Celles-ci en effet ne sont pas directement transposables pour les langages de programmation de robots. La raison principale est qu'un programme classique peut être décrit par l'évolution d'un "état mémoire" formé d'un certatin nombre de variables ou identificateurs de types booléens, entiers, ou flottants. L'équivalent de cet état mémoire pour un robot est son environnement caractérisé par les objets qui le composent, leur forme et surtout leur position. Pour décrire l'effet de l'éxécution des instructions d'un langage de programmation de robots, il faudra auparavant développer un formalisme adapté à la description des états de l'environnement et de leurs transitions. Il faut remarquer également que la programmation de robots se compose essentiellement "d'effets de bord". Une grande partie des instructions est formellement équivalente à une affectation. Par exemple un déplacement du robot à une position A revient à affecter la valeur A à la transformation définissant la position du robot. Cette analogie permet d'utiliser l'axiome de l'affectation de la logique de Hoare vue précédemment. Cependant, les effets de bord concernant une manipulation sont plus complexes: il se propagent à tous les objets liés. Ce phénomène se traduira par une complexité accrue dans le calcul de plus faibles préconditions.

La seconde conclusion concerne les langages de programmation de robots. Certaines instructions d'un langage de niveau effecteur courant comme l'instruction "$FERMER\ PINCE$" de LM n'ont pas un effet interprétable en termes de modification de la position ou de modification de l'incertitude de position ou de modification des liaisons entre objets. Or pour faire la vérification-correction d'un programme, il est nécessaire pourtant que l'effet de chaque instruction de ce programme puisse être décrit de cette façon. Au chapitre 5, nous présenterons un langage particulier adapté à la vérification-correction de programme.

Chapitre 4

Représentation de l'univers

Comme tout programme informatique, un programme de commande de robot décrit formellement des *opérations* devant être effectuées sur des *données*. Pour nous, les données sont les différents états de l'univers réel auquel est "confronté" le robot, les opérations sont les différentes actions effectuées par le manipulateur. Pour pouvoir raisonner sur ces programmes, il est donc nécessaire d'avoir des représentations adaptées pour décrire les différents états de l'univers et l'effet des actions du robot sur cet univers.

En ce qui concerne la modélisation des états de l'univers, on peut trouver dans [Per86] une analyse détaillée des besoins en modélisation pour la programmation automatique des robots. Cependant, par rapport à cette analyse, nous nous concentrerons sur la modélisation dans le contexte de la "vérification/correction". Ce chapitre se compose de quatre parties:

- la première rappelle le cahier des charges pour une modélisation adaptée à la vérification/correction.
- la seconde présente formellement les notions de base que nous manipulons: objets, positions et incertitudes de position.
- la troisième présente les outils permettant de rendre compte de l'aspect "relationnel" de l'environnement: calcul des positions et incertitudes relatives, liaisons entre objets.

- la quatrième présente les outils permettant de décrire les effets des actions du robot.

1 Cahier des charges

D'une façon générale, pour décrire un système physique quelconque, il est nécessaire de choisir un nombre k de *grandeurs physiques* caractérisant chaque état du système. Ces grandeurs physiques définissent k *variables d'état* $(x_1, x_2, \ldots, x_k)$. Le choix de ces variables résulte en général d'un compromis entre complexité du modèle et "qualité" de la description des phénomènes auxquels on s'intéresse.

En ce qui nous concerne nous sommes intéressés principalement par la description de la position d'objets et de l'incertitude qui leur est associée.

Ces objets sont des objets rigides ou des structures articulées formées d'objets rigides élémentaires. Les positions et incertitudes d'un objet doivent pouvoir être définies relativement à un autre objet. De plus, on doit pouvoir être capable de calculer à tout instant les positions et incertitudes relatives de deux objets quelconques.

L'existence de liaisons physiques entre deux objets (par exemple un contact ou une fixation) est très importante quant à la forme et l'évolution des positions et incertitudes relatives:

- si les deux objets sont liés, tout déplacement de l'un deux se traduira par un déplacement de l'autre tel que leur position relative reste constante.
- la forme géométrique du contact (par exemple plan-plan ou arête-plan) conditionnera fortement l'incertitude de position entre les deux objets.

Notre représentation devra pouvoir rendre compte de ces phénomènes.

Il faut remarquer qu'au stade actuel de notre étude, l'aspect géométrique de l'univers peut être négligé. Cependant on pourrait très bien en tenir compte dans notre perspective de vérification/correction. Un avantage serait de pouvoir tenir compte des incertitudes de forme et des tolérances des pièces. Une extension naturelle de notre représentation serait de représenter les objets sous leur aspect géométrique - et non plus sous forme de repère - et de décrire les positions relatives des objets à partir de relations ou positions relatives d'entités géométriques.

L'aspect temporel de l'univers nous intéresse dans la mesure où cet univers évolue sous l'effet des actions du manipulateur. Cette évolution sera évoquée par la suite dans le

chapitre concernant la représentation des actions. En dehors des actions commandées, l'univers sera considéré comme statique.

2 Notions de base *objet, position, erreur, incertitude*

2.1 La notion d'objet

Les objets physiques que le robot aura à manipuler seront considérés comme des solides simples ou des structures articulées constituées d'un nombre fini (généralement quelques unités) de solides liés entre eux. Un cube est un exemple de solide simple, un bras manipulateur est un exemple de structure articulée. On fera l'hypothèse que les solides sont indéformables: on négligera toute plàsticité ou élasticité. Ces hypothèses ne sont pas très restrictives quant au type d'objets généralement rencontrés lors de manipulations.

Ces hypothèses nous amènent à pouvoir représenter chaque solide élémentaire par un repère géométrique. Le solide en tant que distribution de matière sera défini en donnant la position de chacun de ses points dans ce repère. Une structure articulée constituée de N solides élémentaires sera définie à partir de N repères.

2.2 La notion de position

2.2.1 Définition

La position d'un solide Σ_1, associé à un repère $\Re_1$, est définie par rapport à un repère de référence $\Re_0$. $\Re_0$ peut être un repère "fictif"qui n'a pas de réalité physique (cas du repère "STATION" du langage LM) ou un repère attaché à un objet Σ_0. Cette position de Σ_1 dans $\Re_0$ (on dit aussi *par rapport* à $\Re_0$) notée T_{01} est par définition l'isométrie affine (ou transformation géométrique, composition d'une rotation et d'une translation) transformant $\Re_0$ en $\Re_1$ (cf figure 4.1). Cette définition se justifie par le fait qu'elle détermine de façon unique la position de tout point du solide Σ_1 dans le repère $\Re_0$. En effet, si M est un point de Σ_1, défini dans $\Re_1$, $T_{01}(M)$ (image de M par T_{01}) donnera la position de M dans $\Re_0$.

Remarque Par la suite, on fera très souvent l'abus de langage consistant à identifier repères et objets.

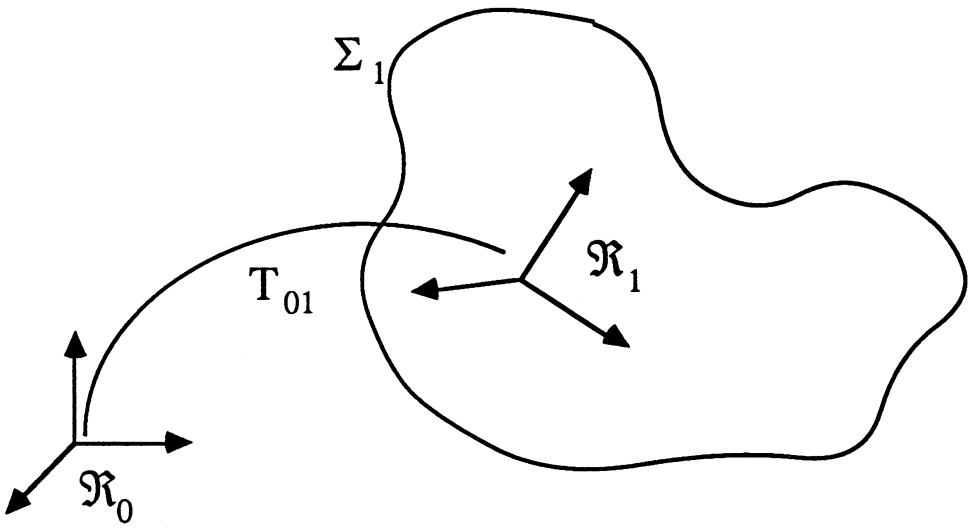


Figure 4.1 : définition de la position

2.2.2 Notations

Le repère transformé d'un repère $\Re$ par une transformation T sera noté $\Re * T$. Ces notations sont celles utilisées en particulier dans le langage LM. Le fait qu'une transformation T_{01} soit la position de Σ_1 dans $\Re_0$ sera noté par l'égalité suivante:

$$\Re_1 = \Re_0 * T_{01} \tag{4.4}$$

Cette convention de notation implique implicitement que la transformation T_{01} *soit exprimée dans le repère* $\Re_0$. Dans l'égalité 4.4, la transformation T_{01} peut elle-même être la composition de plusieurs transformations T_1, T_2 ... T_n. La composition de transformations sera abusivement notée par un signe $*$. L'équation 4.4 s'écrit alors:

$$\Re_1 = \Re_0 * T_2 * T_3 * \cdots * T_n$$

Cette égalité sous-entend que pour calculer $\Re_1$, il faudra que:

- T_2 soit exprimée dans $\Re_0$
- T_3 soit exprimée dans $\Re_0 * T_2$

- T_4 soit exprimée dans $\Re_0 * T_2 * T_3 \ldots$
- T_n soit exprimée dans $\Re_0 * T_2 * T_3 * \cdots * T_{n-1}$

2.2.3 Propriétés

En utilisant les propriétés du groupe des transformations, on peut établir les propriétés suivantes:

Propriété 4.1 *Etant donnés deux objets représentés par deux repères* $\Re_1$ *et* $\Re_2$, T_{12} *la position de* $\Re_2$ *par rapport à* $\Re_1$, *la position de* $\Re_2$ *par rapport à* $\Re_1$ *est* T_{12}^{-1} *(transformation inverse de* T_{12}*)*

Propriété 4.2 *Etant donnés trois objets représentés par trois repères* $\Re_1$, $\Re_2$ *et* $\Re_3$, T_{12} *et* T_{23} *respectivement les positions de* $\Re_2$ *par rapport à* $\Re_1$, *et de* $\Re_3$ *par rapport à* $\Re_2$, *la position de* $\Re_3$ *par rapport à* $\Re_1$ *est* T_{23} *donnée par:*

$$T_{23} = T_{12} * T_{23}$$

preuve 1 L'égalité:

$$\Re_2 = \Re_1 * T_{12}$$

devient, en multipliant à droite chaque membre par T_{12}^{-1}:

$$\Re_1 = \Re_2 * T_{12}^{-1}$$

d'où le premier résultat. □

preuve 2 $\Re_1$, $\Re_2$, $\Re_3$ sont liés par les deux équations suivantes:

$$\Re_2 = \Re_1 * T_{12} \tag{4.5}$$

$$\Re_3 = \Re_2 * T_{23} \tag{4.6}$$

En remplaçant dans 4.6 $\Re_2$ par sa valeur (donnée par 4.5), on obtient:

$$\Re_3 = \Re_1 * (T_{12} * T_{23})$$

Ce qui donne le résultat. □

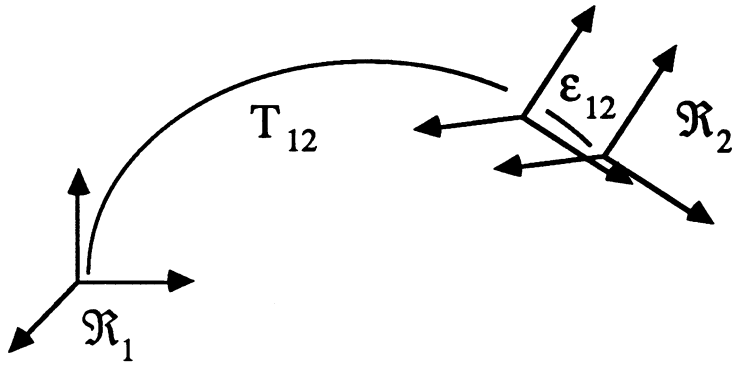


Figure 4.2 : définition de l'erreur de position

2.3 La notion d'erreur de position

2.3.1 Définition

Une transformation T_{12} représentant la position d'un repère $\Re_2$ dans un repère $\Re_1$ représente en fait la position nominale ou théorique de $\Re_2$. Il existe une différence entre cette position théorique et la position réelle T'_{12}. On appelle *erreur de position* cette différence. On pose par définition que l'erreur de position ε_{12} de $\Re_2$ par rapport à $\Re_1$ est la transformation géométrique entre position théorique et position réelle (cf figure 4.2).

La position réelle de $\Re_2$ par rapport à $\Re_1$ est maintenant $T'_{12} = T_{12} * \varepsilon_{12}$, soit:

$$\Re_2 = \Re_1 * T_{12} * \varepsilon_{12}$$

Une position réelle *exacte* correspond à une erreur égale à la transformation identité élément neutre du groupe pour la composition. Il est important de noter aussi que par définition ε_{12} est exprimée dans le repère $\Re_2$

2.3.2 Remarque

Du fait de la non commutativité du produit de transformations, l'ordre des termes est très important. La transformation ε'_{12} telle que:

$$T'_{12} = \varepsilon'_{12} * T_{12}$$

soit également

$$\Re_2 = \Re_1 * \varepsilon'_{12} * T_{12}$$

correspond aussi à un écart entre position réelle et position théorique mais *n'est pas* égale à ε_{12}. En effet, on a:

$$T'_{12} = T_{12} * \varepsilon_{12} \tag{4.7}$$

soit:

$$\varepsilon_{12} = T_{12}^{-1} * T'_{12}$$

D'autre part,

$$T'_{12} = \varepsilon'_{12} * T_{12}$$

soit:

$$\varepsilon'_{12} = T'_{12} * T_{12}^{-1} \tag{4.8}$$

ε_{12} et ε'_{12} ne sont donc pas égales. Il est intéressant cependant de comprendre les liens entre ε_{12} et ε'_{12} à l'aide de la proposition suivante:

Propriété 4.3 *ε'_{12} est la transformation géométrique faisant passer de $\Re_{2théorique}$ à $\Re_{2réel}$* exprimée dans le repère $\Re_1$*; ε_{12} est cette même transformation exprimée dans $\Re_{2théorique}$ (d'après la convention d'écriture utilisée dans l'équation de définition 2.3.1).*

preuve D'après les formules de changement de repère, l'expression de ε_{12} dans $\Re_1$ est égale à $T_{12} * \varepsilon_{12} * T_{12}^{-1}$.
D'autre part, en remplçant dans l'équation 4.8, l'expression de T'_{12} donnée par 4.7, on obtient:

$$\begin{aligned} \varepsilon'_{12} &= T'_{12} * T_{12}^{-1} \\ &= T_{12} * \varepsilon_{12} * T_{12}^{-1} \end{aligned}$$

ce qui donne le résultat. □

2.4 La notion d'incertitude de position

2.4.1 Généralités. Définition

Une erreur de position a cette particularité que sa valeur exacte ne peut jamais être connue. On n'a d'une erreur qu'une information partielle. C'est cette information ou

plutot manque d'information sur une erreur de position qu'on appelle incertitude de position. On peut représenter les incertitudes de position de plusieurs façons. On peut distinguer deux approches principales [MP88]:

- une approche ensembliste où une incertitude est considérée comme l'ensemble des erreurs possibles,
- une approche probabiliste où une erreur est considérée comme une variable aléatoire multidimensionnelle où l'incertitude s'identifie alors à une loi de probabilité [Maz87b,MP88,SCS87,Dur87a].

Chronologiquement, nous avions d'abord opté pour une représentation de type ensembliste [Pug85,PP86,TP87,Pug88] pour la vérification de programme. Cependant, pour différentes raisons, en particulier la simplicité relative des calculs associés au choix d'une représentation probabiliste avec des lois gaussiennes, nous avons révisé ce choix. Au chapitre 6, sont comparées plus en détail les deux approches et est analysé également le problème du choix d'une représentation. Nous présentons ici la mise en œuvre d'un représentation probabiliste avec des lois gaussiennes. L'annexe C présente la mise en œuvre d'une représentation ensembliste.

Une erreur, qui est une transformation peut être représentée par deux vecteurs de R^3, le premier représentant l'erreur en translation, le second l'erreur en rotation[1] (le vecteur de rotation de la transformation: son module est égal à l'angle et sa direction donne l'axe de la rotation). Dans toute la suite du rapport, nous noterons une erreur par la lettre ε, son vecteur de translation par la lettre τ, son vecteur de rotation par la lettre ω. Nous noterons aussi par ν le vecteur normé de même direction et de même sens que ω. Il représente l'axe de la rotation associée à ε. On associera à une erreur ε un vecteur de dimension 6 noté $\vec{\varepsilon}$ dont les trois premières coordonnées sont celles de τ et les trois suivantes celles de ω:

$$\vec{\varepsilon} = \begin{pmatrix} \tau \\ \omega \end{pmatrix}$$

Cette représentation permet alors de considérer $\vec{\varepsilon}$ comme une variable aléatoire réelle de dimension 6. L'incertitude sur $\vec{\varepsilon}$ est donnée sous forme de loi de probabilité sur R^6.

[1] cette représentation est celle adoptée dans [Maz87b]. Dans d'autres travaux, la rotation est représentée par trois angles d'Euler. Dans ce dernier cas, les calculs semblent plus complexes car mettent en œuvre des calculs trigonométriques, en particulier pour les linéarisations.

Nous avons choisi de ne considérer que des lois de probabilité gaussiennes centrées. Ces lois sont caractérisées par la donnée d'une matrice de covariance Δ définie par:

$$\forall i,j \in [1,6],\ \Delta_{ij} = E(\varepsilon_i \varepsilon_j)$$

où E est l'espérance mathématique et ε_i est la $i^{ème}$ coordonnée de $\vec{\varepsilon}$.

2.4.2 Propriétés

Les propriétés générales liées aux matrices de covariance nous permettent de déduire certaines propriétés sur les incertitudes.

Si les coordonnées de ε sont toutes indépendantes, Δ est diagonale. Si l'erreur en translation et l'erreur en rotation sont indépendantes, Δ est formée de deux blocs diagonaux Δ_τ et Δ_ω:

$$\Delta = \begin{pmatrix} \Delta_\tau & 0 \\ 0 & \Delta_\omega \end{pmatrix} .$$

Cette indépedance ou non des coordonnées dépend évidemment du repère dans lequel est exprimée une incertitude. Dans nos exemples ultérieurs nous prendrons toujours des repères pour que les matrices de covariances soient diagonales. Il faut remarquer de plus qu'une matrice de covariance étant symétrique est diagonalisable sur une base orthogonale: il sera donc toujours possible de trouver un repère sur lequel une incertitude sera représentée par une matrice diagonale.

Les ensembles P_n tels que la probabilité $P(\varepsilon \in P_n)$ soit égale à n sont des ellipsoïdes de R^6 d'axes principaux les vecteurs propres de Δ (Δ matrice symétrique réelle positive est diagonalisable sur une base orthonormée et ses valeurs propres sont positives). De plus si Δ admet une ou plusieurs valeurs propres nulles, cela signifie que l'incertitude de position est nulle sur les degrés de liberté correspondant aux vecteurs propres associés.

Enfin, nous citons les deux propriétés suivantes qui seront très importantes pour les calculs sur les incertitudes que nous ferons par la suite.

Propriété 4.4 *La loi de probabilité associée à l'image de ε par une application linéaire de R^6 associée à un matrice M est aussi une loi de probabilité gaussienne centrée de matrice de covariance Δ' telle que:*

$$\Delta' = M \Delta M^t$$

Où M^t est la matrice transposée de M.

Propriété 4.5 *Si deux variables aléatoires gaussiennes $\vec{\varepsilon_1}$ et $\vec{\varepsilon_2}$ sont associées respectivement à deux matrices de covariance Δ_1 et Δ_2, la somme des variables aléatoires $\vec{\varepsilon}$ égale à $\vec{\varepsilon_1} + \vec{\varepsilon_2}$ est une variable aléatoire suivant également une loi de probabilité gaussienne et sa matrice de covariance est $\Delta_1 + \Delta_2$.*

2.4.3 Exemples

Solide sur un plan $z = cste$

Considérons un cube reposant sur un plan "horizontal" d'équation $z = cste$ (voir figure 4.3). Sa position en translation sur le plan est parfaitement définie dans la direction z grâce au contact entre le plan et une face du cube. L'erreur de position en translation a sa troisième coordonnée toujours nulle. Quant à son orientation, les seules rotations permises par le contact sont des rotations suivant l'axe z. L'erreur de position en rotation a ses secondes et troisièmes coordonnées toujours nulles. La matrice de covariance représentant l'incertitude de position du cube sur le plan sera donc de la forme:

$$\Delta = \begin{pmatrix} \Delta_{11} & \Delta_{12} & 0 & & & \\ \Delta_{21} & \Delta_{22} & 0 & & 0 & \\ 0 & 0 & 0 & & & \\ & & & 0 & 0 & 0 \\ & 0 & & 0 & 0 & 0 \\ & & & 0 & 0 & \Delta_{66} \end{pmatrix}$$

Solide dans l'espace

Si on considère maintenant un solide libre dans l'espace (voir figure 4.4), l'incertitude de position en translation n'a plus de direction privilégiée. L'incertitude de position

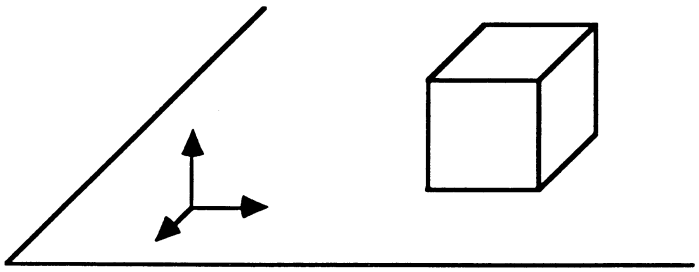


Figure 4.3 : cube sur un plan horizontal

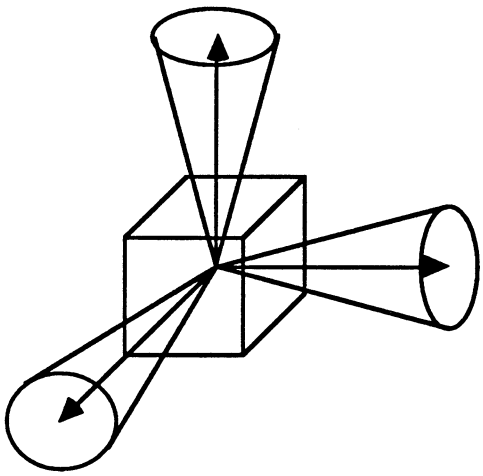


Figure 4.4 : solide dans l'espace

en rotation également. La matrice de covariance représentant l'incertitude de position d'un objet libre dans l'espace est donc une matrice de covariance a priori quelconque.

2.4.4 Relation d'ordre sur les incertitudes

Quand on fait la vérification d'un programme, la satisfaction d'une condition d'applicabilité exprime le fait qu'une incertitude relative entre deux repères est "plus petite" qu'un certain seuil. Ceci a un sens si on a défini au préalable une relation d'ordre sur les incertitudes.

Etant donnée une matrice de covariance Δ, un veteur unitaire x de R^6, la quantité

$x^t \Delta x$ représente une dispersion dans la direction x. Si on considère alors deux matrices de covariance Δ_1 et Δ_2, on dira que l'incertitude représentée par Δ_1 est plus petite ou inférieure à l'incertitude représentée par la matrice de covariance Δ_2 si pour tout vecteur x de R^6, on a:

$$x^t \Delta_1 x \leq x^t \Delta_2 x$$

Ceci est équivalent à:

$$x^t (\Delta_2 - \Delta_1) x \geq 0 \tag{4.9}$$

$(\Delta_2 - \Delta_1)$ est une matrice symétrique. La condition 4.9 exprime que $(\Delta_2 - \Delta_1)$ est positive. C'est équivalent à dire que les valeurs propres en sont toutes positives ou nulles.

En résumé, si Δ_1 et Δ_2 sont deux matrices de covariance, on dit que l'incertitude représentée par Δ_1 est inférieure à l'incertitude représentée par Δ_2 si et seulement si toutes les valeurs propres de $(\Delta_2 - \Delta_1)$ sont positives ou nulles. Ceci est noté:

$$\Delta_1 \subseteq \Delta_2$$

3 Structure de l'univers

On appelle "univers" ou "environnement" l'ensemble formé du robot et des objets participant directement ou indirectement à la manipulation. Nous avons souligné précédemment que l'information la plus intéressante pour nous était les positions des objets et les incertitudes associées. Cependant une position n'est pas une grandeur intrinsèque d'un objet mais s'exprime par rapport à un autre objet ou par rapport à une entité fictive de référence. Dans le cadre d'une manipulation, il est parfois intéressant de modifier la référence définissant la position d'un objet. Il est également intéressant de pouvoir calculer les positions relatives et les incertitudes associées de deux objets quelconques. Pour ces raisons, il est important de prévoir une structure du modèle de l'environnement appropriée.

3.1 Graphe d'état

3.1.1 Définitions

Pour représenter un état de l'univers, on utilise une structure de graphe orienté telle que celle de la figure 4.5. Un nœud du graphe représente un repère (c'est-à-dire un objet élémentaire). L'existence d'un arc d'origine $\Re_i$ et d'extrémité $\Re_j$ signifie que la

position de $\Re_j$ est définie par rapport à $\Re_i$. A cet arc sont associées la position et l'incertitude de $\Re_j$ par rapport à $\Re_i$.

3.1.2 Propriétés

Connexité

Il existe un repère particulier de référence "absolue" dont la position n'est définie par rapport à aucun autre repère. Ce repère appelé $WSPACE$ dans la suite du rapport est l'équivalent du repère $STATION$ de LM. C'est en particulier par rapport à ce repère qu'est définie la position du robot. De plus, on ne considère que des états de l'univers où la position de chacun des objets (hormis $WSPACE$) est définie. Il en découle que tout nœud, sauf $WSPACE$ est l'extrémité d'au moins un arc. En conséquence le graphe d'état sera toujours connexe.

Caractère orienté du graphe

Etant donnés deux repères $\Re_1$ et $\Re_2$, la position relative de $\Re_1$ par rapport à $\Re_2$ n'est pas égale à la position de $\Re_2$ par rapport à $\Re_1$. Il en est de même pour les incertitudes. C'est pour cela qu'il est nécessaire de considérer un graphe *orienté*. Cependant, nous verrons plus loin comment on peut, étant données la position et l'incertitude de $\Re_2$ par rapport à $\Re_1$, calculer la position et l'incertitude de $\Re_1$ par rapport à $\Re_2$. Ceci permet de déterminer un arc symétrique. Le graphe d'état peut alors être considéré implicitement comme un graphe orienté symétrique. Cette propriété associée à la propriété de connexité permet d'affirmer qu'il est toujours possible, étant donnés deux nœuds $\Re_i$ et $\Re_j$ quelconques du graphe, de trouver un chemin d'origine $\Re_i$ et d'extrémité $\Re_j$.

Exemples:

Dans l'état initial de la manipulation prise en exemple dans l'introduction de ce rapport (figure 0.1), la structure de l'univers est représentée par le graphe de la figure 4.5.

Dans cet état, la position de la table et celle du robot sont définies par rapport au repère de référence de la scène. Les deux objets sont définis par rapport à la table. La position de la pince est définie par rapport au robot.

Après la prise du piston P par le robot, la position de P n'est plus définie par rapport à la table mais par rapport à la pince. C'est pourquoi l'univers sera alors représenté

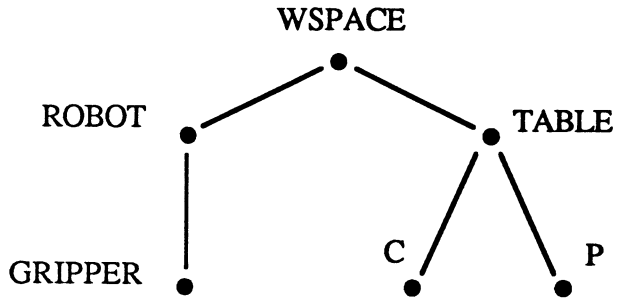


Figure 4.5 : graphe de l'état initial de la manipulation

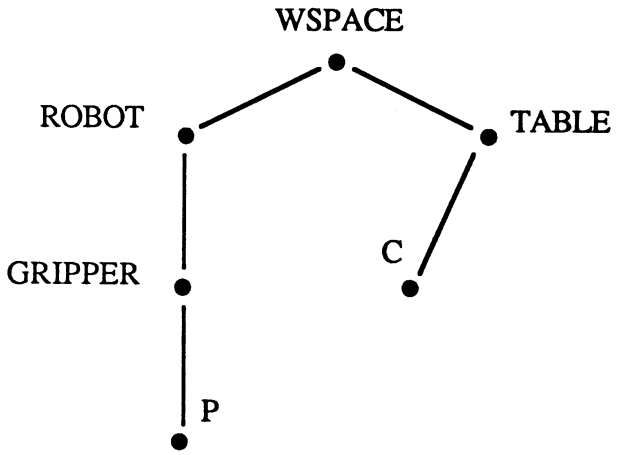


Figure 4.6 : graphe de l'état après prise

par le graphe de la figure 4.6. Les mécanismes de mise à jour et d'évolution du modèle seront vus par la suite.

3.2 Calcul d'un chemin dans le graphe

Etant donné un état de l'univers représenté par un graphe G, pour calculer la position et l'incertitude associée d'un repère $\Re_j$ par rapport à un autre repère $\Re_i$, on doit successivement:

1. trouver un chemin reliant $\Re_i$ à $\Re_j$, noté $Way(\Re_i, \Re_j)$
2. "composer" positions et incertitudes le long de ce chemin.

En ce qui concerne le premier point, il n'y a pas de difficulté particulière et on utilisera des algorithmes classiques de théorie des graphes. Grâce à la propriété de connexité du graphe, on est assuré de trouver un chemin.

Le second point est plus spécifique à notre étude. Evaluer une incertitude le long d'un chemin ne se fait pas simplement en additionnant des coûts, mais en multipliant des transformations. La non-commutativité du produit de transformations entraine des difficultés dans le calcul. Par contre l'associativité permettra d'utiliser un algorithme récursif. Les deux opérations fondamentales qui permettent de calculer la position relative et l'incertitude associée de deux repères quelconques reliés par un chemin sont les suivantes:

- *inversion d'un chemin:* étant donnés deux repères $\Re_i$ et $\Re_j$, leur position et incertitude relatives T_{ij} et I_{ij}, calculer la position et l'incertitude inverses T_{ji} et I_{ji}.
- *combinaison de deux chemins successifs:* étant donnés trois repères $\Re_i$, $\Re_j$, $\Re_k$, reliés par deux chemins successifs, calculer la position et l'incertitude associée T_{ik} et I_{ik} en fonction des positions et incertitudes T_{ij}, I_{ij}, T_{jk}, I_{jk}.

3.2.1 Inversion d'un chemin

L'inversion d'un chemin se fait à partir de la propriété suivante:

Propriété 4.6 *Etant donné un chemin $Way(\Re_i, \Re_j)$ associé à une transformation T_{ij} et une incertitude I_{ij}, la position T_{ji} associée au chemin inverse $Way(\Re_j, \Re_i)$ est la transformation inverse de T_{ij}, soit:*

$$T_{ji} = T_{ij}^{-1}$$

L'erreur ε_{ji} sur la position inverse en fonction de l'erreur ε_{ij}, sur la position directe est donnée par:

$$\varepsilon_{ji} = T_{ji}^{-1} * \varepsilon_{ij}^{-1} * T_{ji}$$

preuve Par hypothèse, $\Re_i$ et $\Re_j$ sont liés par l'équation suivante:

$$\Re_j = \Re_i * T_{ij} * \varepsilon ij$$

soit,

$$\begin{aligned}\Re_i &= \Re_j * \varepsilon_{ij}^{-1} * T_{ij}^{-1} \\ &= \Re_j * T_{ji} * (T_{ji}^{-1} * \varepsilon_{ij}^{-1} * T_{ji})\end{aligned}$$

Le résultat est obtenu en identifiant le terme de position et le terme d'erreur. □

Interprétation du terme d'erreur: Par convention d'écriture, ε_{ij} est exprimée dans $\Re_j$. ε_{ij}^{-1} est exprimée dans ce même repère. Le terme $(T_{ji}^{-1} * \varepsilon_{ij}^{-1} * T_{ji})$ est cette même transformation exprimée dans $\Re_i$ d'après les formules de changement de repère. On peut donc interpréter l'expression de ε_{ji} en disant que ε_{ji} est égale à la transformation inverse de ε_{ij}, exprimée dans le repère $\Re_i$.

Calcul de la matrice de covariance correspondante

Pour calculer la matrice de covariance associée à l'erreur $\varepsilon_{ji} = T_{ji}^{-1} * \varepsilon_{ij}^{-1} * T_{ji}$, on applique successivement les deux résultats suivants dont les démonstrations sont données en annexe B:

Propriété 4.7 *Soit $\vec{\varepsilon}$ une variable aléatoire gaussienne centrée de matrice de covariance Δ associée à une erreur ε, l'erreur aléatoire $\vec{\varepsilon'}$ associée à l'erreur ε^{-1} suit* approximativement au premier ordre *une loi de probabilité gaussienne de matrice de covariance égale aussi à Δ.*

Propriété 4.8 *Soit T une transformation de translation t et de rotation r, $\vec{\varepsilon}$ une variable aléatoire gaussienne centrée de matrice de covariance Δ associée à une erreur ε, la variable aléatoire $\vec{\varepsilon'}$ associée à l'erreur $\varepsilon' = T^{-1} * \varepsilon * T$ suit* approximativement au premier ordre *une loi de probabilité gaussienne de matrice de covariance $\Delta' = M \Delta M^t$, M étant une matrice 6×6 formée des blocs:*

$$M = \begin{pmatrix} r^{-1} & r^{-1}\mathcal{M}_t \\ 0 & r^{-1} \end{pmatrix}$$

Avec

$$\mathcal{M}_t = \begin{pmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{pmatrix}$$

t_x, t_y, t_z étant les coordonnées de t.

Le terme "approximativement au premier ordre" signifie que l'on a recours à une linéarisation de ε'. Cette linéarisation n'est valable que pour des angles de rotation suffisamment petits (d'écart type inférieur à 0.1 rad). Pour la propriété 4.7, on montre que $\vec{\varepsilon'}$ est égale au premier ordre à $-\vec{\varepsilon}$. Pour la propriété 4.8, on montre que $\vec{\varepsilon'}$ est égale au premier ordre à $(r^{-1}(\omega \wedge t + \tau) \quad r^{-1}\omega)^t$ soit $M\varepsilon$. Les résultats découlent ensuite de l'application de la propriété 4.4. Le détail des calculs conduisant à ces résultats ainsi que l'évaluation des erreurs dûes à la linéarisation sont présentés à l'annexe B.

En conclusion, pour calculer l'inversion d'un chemin, si on appelle M_{ij} la matrice M associée à la transformation T_{ij}, Δ_{ji} sera égale à:

$$\Delta_{ji} = M_{ij} * \Delta_{ij} * M_{ij}^t \tag{4.10}$$

3.2.2 Combinaison de deux chemins

La combinaison de deux chemins se fait à partir de la propriété suivante:

Propriété 4.9 *Etant donnés deux chemins successifs, $Way(\Re_i, \Re_j)$ et $Way(\Re_j, \Re_k)$, associés respectivement aux positions T_{ij}, T_{jk} et aux incertitudes I_{ij}, I_{jk}, la position T_{ik} du repère $\Re_k$ par rapport à $\Re_i$ associée au chemin $Way(\Re_i, \Re_k)$ est donnée par:*

$$T_{ik} = T_{ij} * T_{jk}$$

L'erreur ε_{ik} sur la position de $\Re_k$ par rapport à $\Re_i$ en fonction des erreurs ε_{ij} et ε_{jk} est donnée par:

$$\varepsilon_{ik} = (T_{jk}^{-1} * \varepsilon_{ij} * T_{jk}) * \varepsilon_{jk}$$

preuve Par hypothèse, $\Re_i$, $\Re_j$ et $\Re_k$ sont liés par les équations suivantes:

$$\Re_j = \Re_i * T_{ij} * \varepsilon_{ij} \tag{4.11}$$

$$\Re_k = \Re_j * T_{jk} * \varepsilon_{jk} \tag{4.12}$$

En remplaçant dans 4.11 $\Re_j$ par sa valeur, on obtient:

$$\begin{aligned} \Re_k &= \Re_i * T_{ij} * \varepsilon_{ij} * T_{jk} * \varepsilon_{jk} \\ &= \Re_i * (T_{ij} * T_{jk}) * (T_{jk}^{-1} * \varepsilon_{ij} * T_{jk}) * \varepsilon_{jk} \end{aligned}$$

Le résultat est obtenu en identifiant le terme de position et le terme d'erreur. □

Interprétation du terme d'erreur: La transformation ε_{ij} est exprimée dans le repère $\Re_j$, le terme $(T_{jk}^{-1} * \varepsilon_{ij} * T_{jk})$ est l'expression de ε_{ij} dans le repère $\Re_k$. Le terme d'erreur total ε_{ik} peut donc s'interpréter comme le produit de:

- l'erreur ε_{ij} dont l'expression est ramenée dans $\Re_k$
- l'erreur ε_{jk}

Calcul de la matrice de covariance correspondante

Pour calculer la matrice de covariance associée à l'erreur $\varepsilon_{ik} = T_{jk}^{-1} * \varepsilon_{ij} * T_{jk} * \varepsilon_{jk}$, on applique successivement les résultats de la propriété 4.8 et de la propriété suivante:

Propriété 4.10 *Soient $\vec{\varepsilon_1}$ et $\vec{\varepsilon_2}$ deux variables aléatoires gaussiennes centrées de matrices de covariance respectives Δ_1 et Δ_2 associées à deux erreurs ε_1 et ε_2. La variable aléatoire $\vec{\varepsilon'}$ associée à l'erreur $\varepsilon_1 * \varepsilon_2$ suit* approximativement au premier ordre *une loi de probabilité gaussienne centrée de matrice de covariance $\Delta_1 + \Delta_2$.*

Cette propriété découle du fait que $\varepsilon' = \varepsilon_1 * \varepsilon_2$ représentée par un vecteur de dimension 6 tel que nous l'avons défini est égal au premier ordre à la somme des 6-vecteurs ε_1 et ε_2. Le détail des calculs ainsi que l'évaluation de l'erreur commise sont présentés à l'annexe B.

En conclusion, l'incertitude associée à la composition des deux chemins $Way(\Re_i, \Re_j)$ et $Way(\Re_j, \Re_k)$, est égale à (en reprenant les notations précédentes):

$$\Delta_{ik} = M_{jk} \Delta_{ij} M_{jk}^t + \Delta_{jk} \tag{4.13}$$

3.2.3 Calcul de la position relative de deux repères quelconques

Etant donnés deux repères $\Re_i$ et $\Re_j$, le calcul de la position de $\Re_j$ par rapport à $\Re_i$ et de l'incertitude associée se fait en deux étapes à l'aide des opérations précédentes:

1. **Détermination d'un chemin dans le graphe symétrisé.** Le calcul d'arcs symétriques se fait en appliquant l'inversion d'un chemin aux arcs du graphe. Supposons que ce chemin soit la suite alternée de repère et d'arcs suivante:

$$\Re_i Arc(\Re_i, \Re_1)\Re_1 Arc(\Re_1, \Re_2)\Re_2 \cdots \Re_n Arc(\Re_n, \Re_j)\Re_j$$

2. **Combinaison des (n+1) arcs composant le chemin.** On utilise le résultat sur la combinaison de deux chemins en décomposant récursivement, le chemin total est en deux sous-chemins dont l'un est un arc (dont on connaît la position et l'incertitude correspondantes):

premier pas: $T_{ij} = T_{i1} * T_{1j}$
$\Delta_{ij} = M_{1j}\Delta_{i1}M_{1j}^t + \Delta_{1j}$
second pas: $T_{1j} = T_{12} * T_{2j}$
$\Delta_{1j} = M_{2j}\Delta_{12}M_{2j}^t + \Delta_{12}$
...
dernier pas: $T_{(n-1)j} = T_{(n-1)n} * T_{nj}$
$\Delta_{(n-1)j} = M_{nj}\Delta_{(n-1)n}M_{nj}^t + \Delta_{nj}$

Pour un chemin comportant (n+1) arcs, le calcul se fait en n décompositions.

3.3 Problèmes causés par l'existence de cycles

Pour calculer la position relative et l'incertitude associée de deux repères quelconques, nous avons fait l'hypothèse (justifiée) qu'il existait un chemin entre ces deux repères. Cependant, s'il existe plusieurs chemins différents entre ces deux repères, à cause principalement des incertitudes, il s'avère généralement que:

1. les positions nominales relatives de ces deux repères, calculées en composant les transformations le long de chacun des chemins n'ont pas les mêmes valeurs,
2. de même, les incertitudes calculées le long de chacun des chemins n'ont pas non plus les mêmes valeurs.

L'existence ou non de plusieurs chemins entre deux repères dépend de la forme du graphe d'état. Du fait de la symétrisation possible du graphe, l'existence de plusieurs chemins entre deux repères est équivalent à l'existence d'un cycle. Il y a un cycle à partir du moment où un repère différent de $WSPACE$ a sa position définie par rapport à plusieurs autres repères, directement ou indirectement (cas de la figure 4.7). Si chaque objet différent de $WSPACE$ a sa position définie par rapport à un et un seul repère, le graphe a une structure d'arbre de racine $WSPACE$.

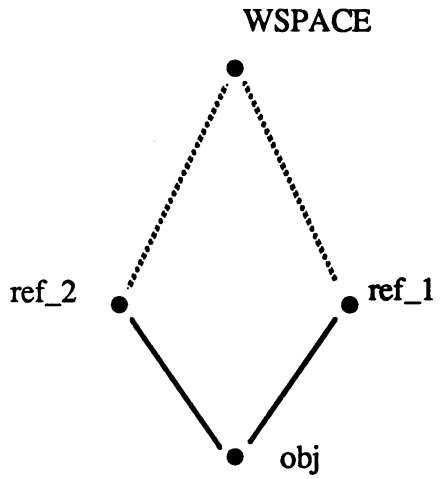


Figure 4.7 : *obj* est défini par rapport à deux références *ref_1* et *ref_2*

3.3.1 Création potentielle de cycles au cours de la manipulation

Au cours d'une manipulation, il arrive que la position d'objets puisse être définie par rapport plusieurs autres repères pouvant créer alors un cycle dans le graphe. Ceci arrive en particulier dans les cas suivants.

Cas de références multiples

Il peut arriver que la position d'un objet $\Re_i$ soit définie par rapport à un objet $\Re_j$ sur certains degrés de liberté, par rapport à un autre objet $\Re_k$ sur d'autres degrés de liberté. Cela est le cas par exemple lors de l'établissement de contacts entre des objets, en particulier lors de la saisie ou de la dépose. Ainsi lorsqu'on saisit un cube reposant sur un plan par un pince à deux mors parallèles (cf figure 4.8), le cube se déplace sur le plan de telle sorte que deux faces viennent en contact avec les mors de la pince. Juste après la saisie, la position en translation suivant z et l'orientation suivant x et y sont déterminées par rapport au plan horizontal. La position en translation suivant y et l'orientation suivant z sont déterminées par rapport à la pince du robot.

Cas d'une mesure

Lorsqu'on mesure la position d'un objet attaché à un repère $\Re_i$ à l'aide d'un capteur de vision par exemple, on calcule la transformation entre un repère $\Re_m$ et le repère $\Re_i$. $\Re_m$ peut être un autre objet de la scène (cas de la mesure de la position relative

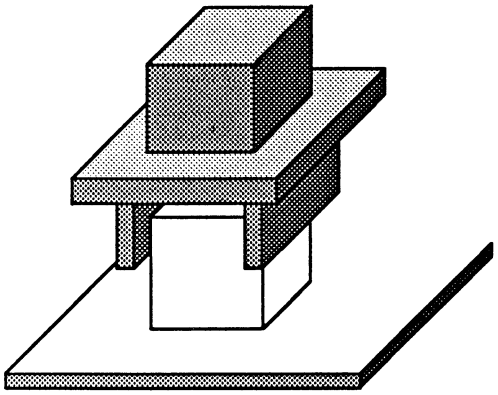


Figure 4.8 : Saisie d'un cube

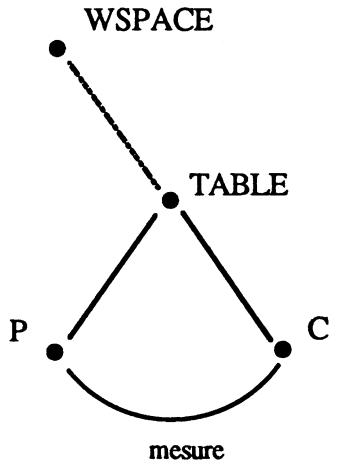


Figure 4.9 : Mesure de la position de C par rapport à P

de deux objets) ou peut être un repère lié au capteur (cas de la mesure d'une position absolue). Si auparavant la référence de position de $\Re_i$ est un repère $\Re_j$ différent de $\Re_m$, on crée un cycle. Par exemple, dans la figure 4.9, les positions du piston P et du cylindre C sont définies par rapport à $TABLE$. Si on mesure la position de C par rapport à P, on crée un cycle.

3.3.2 Problèmes engendrés par l'existence de cycles

Le fait que des valeurs de position et d'incertitudes relatives dépendent du chemin sur lequel elles sont calculées induit les problèmes suivants:

1. Lors de la propagation descendante des incertitudes, lorsqu'on a à calculer la position et l'incertitude relatives de deux repères $\Re_1$ et $\Re_2$, il faut successivement:
   - calculer *tous* les chemins reliant $\Re_1$ à $\Re_2$, les positions relatives et les incertitudes associées,
   - calculer une seule valeur de position et une seule incertitude (les "meilleures" possibles) compatibles avec tous les résultats précédents
2. La propagation arrière des contraintes est beaucoup plus difficile. La propagation arrière des contraintes se fait par reecriture de termes dans une expression algébrique (voir chapitre 5): un terme $WAY(\Re_i, \Re_j) : U$ est remplacé par une expression donnant sa valeur après modification d'une partie d'un chemin liant $\Re_i$ à $\Re_j$. S'il n'existe qu'un seul chemin entre $\Re_i$ et $\Re_j$, il est facile de calculer la modification de $WAY(\Re_i, \Re_j) : U$ apportée par la modification d'une partie du chemin. S'il existe plusieurs chemins, la modification apportée sur $WAY(\Re_i, \Re_j)$ est beaucoup plus complexe à calculer.

3.3.3 Conclusion

A cause des problèmes évoqués ci-dessus, nous avons jugé préférable d'éliminer tout cycle dans le graphe représentant l'environnement, et de ne manipuler que des structures arborescentes. Ceci est rendu possible par les trois restrictions suivantes:

- le graphe décrivant la situation initiale de l'environnement doit être arborescent. Tout repère autre que $WSPACE$ doit avoir un repère de référence et un seul.
- on évite de décrire les effets des actions en introduisant des références multiples (cas des actions de prise et de lâcher d'objet). C'est ce qui est fait dans les descriptions d'actions faites au chapitre 5.
- quand on mesure la position relative de deux repères $\Re_i$ et $\Re_m$, on ne garde pas l'arc supplémentaire ainsi créé mais on reporte l'information sur tous les arcs du chemin $WAY(\Re_i, \Re_m)$ (modification des positions nominales et réduction de l'incertitude). Les méthodes utilisées pour cette opération font l'objet du paragraphe 4.1.

4 Opérateurs associés au modèle

Les différentes opérations vues jusqu'à maintenant permettent de calculer les positions et incertitudes relatives de deux objets quelconques de l'environnement. Ce sont les

opérations de base de notre modèle et sont absolument nécessaires. Les opérations que nous allons présenter dans ce paragraphe servent à décrire les effets de l'éxécution des différentes actions du robot. La liste que nous présentons ici n'est absolument pas exhaustive: nous présentons seulement les opérations dont nous avons besoin pour décrire les actions principales du robot présentées au chapitre 5. Nous tenons à préciser que le choix des opérateurs, apparemment très formels, a été guidé par la réalité de la manipulation: d'une part la forme des résultats fournis par un capteur de vision, d'autre part la "forme" qu'imposent les contacts aux incertitudes.

4.1 Intégration du résultat fourni par un capteur de vision

Un capteur de vision est un capteur apte à mesurer la position relative de deux objets. Nous supposerons donc que le résultat fourni par un tel capteur se présente sous la forme d'une transformation nominale entre un repère $\Re_i$ et un repère $\Re_j$ et une incertitude associée.Deux cas se présentent:

- Dans le modèle, $\Re_j$ est déjà défini par rapport à $\Re_i$. On a alors deux valeurs a priori différentes de positions et d'incertitudes pour l'arc $Arc(\Re_i, \Re_j)$: celle du modéle et celle donnée par la mesure. Il s'agit alors de retenir une valeur "optimale" de position et d'incertitude compatible avec les deux précédentes.
- $\Re_i$ et $\Re_j$ sont reliés par un chemin formé de plusieurs arcs. Pour ne pas créer de cycle dans le graphe d'état, on reporte l'information sur tous les arcs composant le chemin.

Pour traiter ces deux cas, nous utilisons le fitre de Kalman employé en traitement du signal et depuis peu dans le cadre de la robotique, en particulier pour l'intégration de mesures multi-capteurs (voir [AF88,Ram88] pour une présentation plus détaillée du filtre et son utilisation). Le filtre de Kalman permet de trouver un estimateur optimal a_N^* du vecteur d'état a d'un système à partir de N mesures $y_1, y_2, \ldots y_N$ satisfaisant aux équations de mesures suivantes:

$$y_i = M_i a + \omega_i$$

où M_i sont des matrices connues et ω_i un bruit de covariance donnée. Le calcul se fait récursivement à partir d'une estimation initiale $\hat{a}_0$ et une matrice de covariance initiale Δ_0. Considérant a_N^* comme une variable aléatoire, le filtre permet d'en calculer une matrice de covariance. Bien que le premier cas puisse être considéré comme un cas particulier du second, pour des raisons didactiques, nous verrons successivement l'application au cas 1 ("fusion de deux mesures de position") et au cas 2 ("mise à jour des arcs d'un chemin composé")

4.1.1 Application à la fusion de deux mesures de position

Supposons que l'on connaisse la position relative T_1 de deux objets et leur incertitude sous la forme d'une matrice de covariance Δ_1, et qu'on effectue une mesure supplémentaire donnant le résultat T_2 avec une incertitude Δ_2. On considère le 6-vecteur X_1 représentant la position T_1 relative comme le vecteur d'état intéressant. La mesure donne directement une valeur de ce vecteur d'état (les matrices M_i sont ici l'identité). Si ainsi X_1, Δ_1 sont l'estimation initiale et la covariance initiale, X_2 la valeur de la mesure et Δ_2 sa covariance, les équations du filtre donnent pour nouvelle valeur de la position:

$$X^* = X_1 + \Delta_1(\Delta_1 + \Delta_2)^{-1}(X_2 - X_1) \tag{4.14}$$

Et pour nouvelle valeur de la covariance:

$$\Delta^* = (I - \Delta_1(\Delta_1 + \Delta_2)^{-1})\Delta_1$$

Malgré les apparences, ces équations sont symétriques en Δ_1, Δ_2, X_1 et X_2: on ne privilégie ni l'ancienne valeur, ni la nouvelle mesure.

En résumé, pour calculer la nouvelle valeur T^* à partir de deux valeurs T_1 et T_2 et de deux covariances Δ_1 et Δ_2, si on note *vec* la fonction qui associe à une transformation T le 6-vecteur $(\tau \ \omega)^t$ correspondant, alors T^* sera donnée par:

$$T^* = vec^{-1}(vec(T_1) + \Delta_1(\Delta_1 + \Delta_2)^{-1}(vec(T_2) - vec(T_1)))$$

Propriétés du filtre

1. La covariance obtenue est "inférieure" à chacune des covariances initiales.
2. L'estimateur a_N^* fourni par le filtre de Kalman a pour propriété de minimiser le critère des moindres carrés suivant:

$$C = \frac{1}{2}(\hat{a}_0 - a)^t\Delta_0^{-1}(\hat{a}_0 - a) + \sum_{i=1}^{N}(y_i - M_i a)^t W_i^{-1}(y_i - M_i a)$$

   Dans notre cas particulier, X^* minimise:

$$C = \frac{1}{2}(X - X_1)^t\Delta_1^{-1}(X - X_1) + \frac{1}{2}(X - X_2)^t\Delta_2^{-1}(X - X_2)$$

   Ce critère représente la somme des carrés des sommes des erreurs de mesures pondérées par leurs covariances: les mesures les plus précises (donc avec une

covariance plus petite) sont affectées d'un poids plus fort.
Il est facile ici de vérifier que la valeur $\hat{X}$ qui minimise C est bien égale à la valeur X^* donnée par l'égalité 4.14. En effet, $\hat{X}$ annule la dérivée de C par rapport à X qui vaut:

$$\frac{\partial C}{\partial X} = \Delta_1^{-1}(X - X_1) + \Delta_2^{-1}(X - X_2)$$

D'où:

$$\hat{X} = (\Delta_1^{-1} + \Delta_2^{-1})^{-1}(\Delta_1^{-1}X_1 + \Delta_2^{-1}X_2)$$

On en tire:

$$\hat{\Delta} = E(\hat{X}^t\hat{X}) = (\Delta_1^{-1} + \Delta_2^{-1})$$

Les égalités respectives de $\hat{X}$ et X^*, $\hat{\Delta}$ et Δ^* se vérifient en utilisant les égalités suivantes:

$$\begin{aligned}(\Delta_1^{-1} + \Delta_2^{-1})^{-1} &= \Delta_1(\Delta_1 + \Delta_2)^{-1}\Delta_2 \\ (\Delta_1^{-1} + \Delta_2^{-1})^{-1} &= \Delta_2(\Delta_1 + \Delta_2)^{-1}\Delta_1\end{aligned}$$

Ces égalités s'établissent elles-mêmes en vérifiant que les quantités $\Delta_2(\Delta_1 + \Delta_2)^{-1}\Delta_1(\Delta_1^{-1} + \Delta_2^{-1})$ et $\Delta_2(\Delta_1 + \Delta_2)^{-1}\Delta_1(\Delta_1^{-1} + \Delta_2^{-1})$ sont égales à l'identité. Même si les deux résultats sont équivalents, le résultat donné par le filtre est plus intéressant d'un point de vue numérique: il ne nécessite qu'une inversion de matrice au lieu de trois.

4.1.2 Application à la mise à jour des arcs d'un chemin composé

Supposons que l'on mesure la position relative de deux repères $\Re_1$ et $\Re_3$ reliés par un chemin formé de deux arcs $ARC(\Re_1, \Re_2)$ et $ARC(\Re_2, \Re_3)$ (figure 4.10). Soient T_{12}, T_{23} les transformations nominales associées à ces deux arcs, Δ_{12} et Δ_{23} les incertitudes correspondantes. Soient $\overline{T}_{13}$, $\overline{\Delta}_{13}$ respectivement la transformation relative mesurée entre $\Re_1$ et $\Re_3$, et l'incertitude correspondante. En général, $\overline{T}_{13}$ n'est pas égale à la composition $T_{13} = T_{12} * T_{23}$. Pour supprimer l'arc supplémentaire formé par la mesure, il faut "reporter" le résultat de cette mesure sur chacun des arcs T_{12} et T_{23}. Comme la composition de transformations n'est pas linéaire, on ne peut pas procéder directement. Il faut linéariser en passant par des "petites" transformations correctrices. Soient δ_{12}, δ_{23} et δ_{13} les transformations correctrices à apporter respectivement à T_{12}, T_{23} et $\overline{T}_{13}$ telles que:

$$(T_{12} * \delta_{12}) * (T_{23} * \delta_{23}) = (\overline{T}_{13} * \delta_{13}) \qquad (4.15)$$

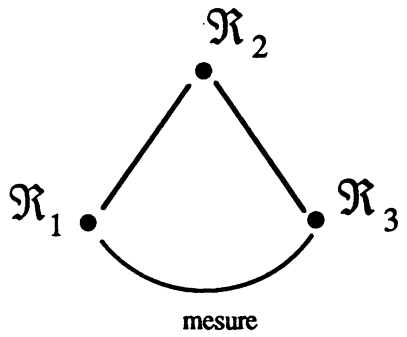


Figure 4.10 : Le chemin allant de $\mathfrak{R}_1$ à $\mathfrak{R}_3$ est formé de deux arcs

Cette équation s'écrit aussi:

$$(T_{12} * \delta_{12} * T_{12}^{-1}) * (T_{13} * \delta_{23} * T_{13}^{-1}) = (\overline{T}_{13} * T_{13}^{-1}) * (T_{13} * \delta_{13} * T_{13}^{-1}) \qquad (4.16)$$

$\delta'_{12} = (T_{12} * \delta_{12} * T_{12}^{-1})$ est une "petite" transformation et représente la correction δ_{12} rapportée dans le repère $\mathfrak{R}_1$, $\delta'_{13} = (T_{13} * \delta_{23} * T_{13}^{-1})$ est la correction δ_{13} rapportée aussi dans $\mathfrak{R}_1$. Le premier terme du second membre $(\overline{T}_{13} * T_{13}^{-1}$ est la "différence" entre la valeur mesurée $\overline{T}_{13}$ de la transformation entre $\mathfrak{R}_1$ et $\mathfrak{R}_3$ et la valeur obtenue par composition de T_{12} et T_{23}.

Soient X_1 et X_2 les 6-vecteurs représentant respectivement δ'_{12} et δ'_{23}. On considère X_1 et X_2 comme des variables aléatoires gaussiennes de moyenne nulle et de covariances respectives:

$$\Delta'_{12} = M_{21}\Delta_{12}M_{21}^t \qquad (4.17)$$
$$\Delta'_{23} = M_{31}\Delta_{23}M_{31}^t \qquad (4.18)$$

M_{21} et M_{31} sont les matrices 6×6 associées respectivement aux transformations T_{21} et T_{31} telles qu'elles ont été définies dans la proposition 4.8.

Soit A_3 le 6-vecteur représentant la transformation $(\overline{T}_{13} * T_{13}^{-1})$. L'expression vectorielle du second membre de l'égalité 4.16 doit alors être considérée comme une variable aléatoire X_3 de moyenne A_3 et de covariance $\Delta'_{13} = M_{31}\overline{\Delta}_{13}M_{31}^t$. On peut linéariser l'équation 4.16 et l'exprimer sous forme de 6-vecteurs. Notre problème se ramène alors

à trouver des estimateurs X_1^*, X_2^*, X_3^* tels que:

$$X_1^* + X_2^* = X_3^*$$

Pour cela on utilise le filtre de Kalman.

Calcul de X_1^*

On considère que X_1 a pour valeur estimée initiale 0 assortie d'une covariance Δ_1 et pour mesure $(A_3 - 0)$ qui est la valeur moyenne de $(X_3 - X_2)$ assortie d'une covariance $(\Delta'_{23} + \Delta'_{13})$.
Les équations du filtre donnent:

$$X_1^* = \Delta'_{12}(\Delta'_{12} + \Delta'_{23} + \Delta'_{13})^{-1} A_3$$

$$\Delta'^*_{12} = (I - \Delta'_{12}(\Delta'_{12} + \Delta'_{23} + \Delta'_{13})^{-1})\Delta'_{12}$$

Cette valeur de X_1^* nous donne une valeur de δ'_{12} correspondante, d'où celle de δ_{12} et la nouvelle valeur de T_{12}. La valeur de Δ'^*_{12} nous permet de calculer une nouvelle valeur de Δ_{12} en inversant l'équation 4.17.

Calcul de X_2^*

On procède de même. On trouve ici:

$$X_2^* = \Delta'_{23}(\Delta'_{12} + \Delta'_{23} + \Delta_{13})'^{-1} A_3$$

$$\Delta'^*_{23} = (I - \Delta'_{23}(\Delta'_{12} + \Delta'_{23} + \Delta'_{13})^{-1})\Delta'_{23}$$

Résumé et cas général

L'approche précédente se généralise au cas où on mesure la position relative de deux repères $\Re_1$ et $\Re_2$ reliés par un chemin formé de N arcs. Si on note:

- $\overline{T}_{12}$ la transformation mesurée, et $\overline{\Delta}_{12}$ la covariance associée,
- $\Delta'_{kl} = M_{l1} * \Delta_{kl} * M_{l1}^t$ pour tout arc $ARC(\Re_k, \Re_l)$ du chemin, avec les notations habituelles,
- comme précédemment *vec* la fonction qui à une transformation associe le 6-vecteur correspondant

alors la nouvelle valeur T^*_{ij} d'une transformation T_{ij} d'un arc du chemin est:

$$T^*_{ij} = T_{ij} * T_{1i}^{-1} * vec^{-1}[\Delta'_{ij}(\sum_{WAY(\Re_1,\Re_2)} \Delta'_{kl} + \overline{\Delta}_{12})^{-1} vec(\overline{T}_{12} * T_{12}^{-1})] * T_{1i}$$

La nouvelle incertitude associée est:

$$\Delta^*_{ij} = M_{j1}^{-1}[I - \Delta'_{ij}(\sum_{WAY(\Re_1,\Re_2)} \Delta'_{kl} + \overline{\Delta}_{12})^{-1}]\Delta'_{ij}(M_{j1}^{-1})^t$$

Cette incertitude est plus "petite" que l'incertitude initiale.

Notations

On note $merge[Arc(obj_k, obj_l), obj_i, obj_j, T, \Delta]$ le résultat sur $Arc(obj_k, obj_l)$ de l'intégration de la mesure de position et d'incertitude relative de $\Re_j$ par rapport à $\Re_i$ ayant pour résultat T et Δ. $merge[Arc(obj_k, obj_l), obj_i, obj_j, T, \Delta]$ est donc équivalent à un arc.

4.2 Opérateur de projection pour la description d'actions mettant en œuvre des contacts

Un contact entre deux objets impose un contrainte sur la position relative des deux objets et contribue ainsi à réduire l'incertitude de position relative. A priori, on ne considère que des contacts point-plan, droite-plan, ou plan-plan. Les autres contacts point-point, ou point-droite, ou droite-droite sont instables et en pratique, on ne cherche *jamais* à les obtenir [The89].
Pour chaque type de contact retenu, notre démarche est la suivante:

1. L'analyse cinématique du contact permet de calculer les contraintes résultantes s'appliquant aux degrés de liberté.

2. Connaissant une matrice de covariance décrivant l'incertitude de position entre les deux objets, la contrainte imposée par le contact peut être interprétée comme une équation de mesure. L'utilisation du filtre de Kalman (éventuellement étendu si les contraintes sont non linéaires) permet de calculer une incertitude plus petite.

Aprés avoir analysé la cinématique des trois types de contact, nous verrons que l'application du filtre de Kalman étendu n'est pas possible pour le contact droite-plan. La démarche complète sera exposée pour le contact le plus fréquent qui est le contact plan-plan.

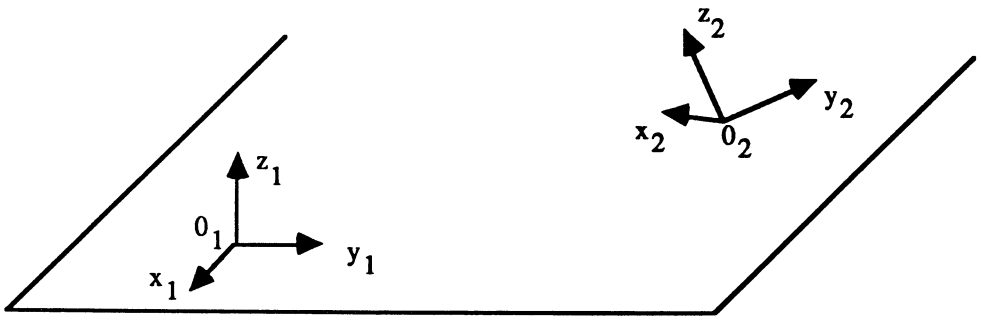


Figure 4.11 : les deux objets sont en contact

4.2.1 Analyse cinématique des contraintes

On considère un objet 1 associé à un repère $\Re_1 = (O_1, x_1, y_1, z_1)$ tel que le plan $\Pi = (O_1, x_1, y_1)$ soit le plan de contact. Un objet 2 associé à un repère $\Re_2 = (O_2, x_2, y_2, z_2)$ a sa position définie par rapport à $\Re_1$ (voir figure 4.11). Soient:

- ε la transformation entre $\Re_1$ et $\Re_2$,
- $\tau = (\tau_x\ \tau_y\ \tau_z)^t$ le vecteur de translation de ε. τ est égal à $\vec{O_1O_2}$,
- $u = (l\ p\ q)^t$ l'axe unitaire de rotation de ε,
- θ l'angle de rotation de ε,
- $\omega = \theta u = (\alpha\ \beta\ \gamma)^t$ le vecteur de rotation de ε.

Pour calculer les contraintes imposées par les différents contacts, on utilise les quaternions [PJ82]. Le quaternion Q associé à la rotation de ε est égal à:

$$Q = c + s(li + pj + qk)$$

où c et s sont respectivement $cos\frac{\theta}{2}$ et $sin\frac{\theta}{2}$. Les quaternions associés aux vecteurs x_2 et z_2 sont respectivement Qx_1Q^{-1} et Qz_1Q^{-1}, d'où leurs coordonnées:

$$x_2 = \begin{pmatrix} 1 - 2s^2(p^2 + q^2) \\ 2(s^2lp + csq) \\ 2(s^2lq - csp) \end{pmatrix}$$

$$z_2 = \begin{pmatrix} 2(s^2lq + csp) \\ 2(s^2pq - csl) \\ 1 - 2s^2(l^2 + p^2) \end{pmatrix}$$

Contact point-plan

O_2 est en contact avec Π si et seulement si:

$$\begin{cases} \tau_y = 0 \\ \tau_z = 0 \end{cases}$$

Ceci est la contrainte de contact point-plan. Elle est linéaire et permet d'utiliser immédiatement le filtre de Kalman.

Contact droite-plan

L'axe (O_2, x_2) est en contact avec Π si et seulement si O_2 appartient à Π et si x_2 est orthogonal à z_1 soit:

$$\begin{cases} \tau_x = \tau_y = 0 \\ s^2 lq - csp = 0 \end{cases} \text{ soit } \begin{cases} \tau_x = \tau_y = 0 \\ \theta = 0 \text{ ou } lq - tg\theta p = 0 \end{cases}$$

Soit encore, après avoir linéarisé $tg\theta$:

$$\begin{cases} \tau_x = \tau_y = 0 \\ \alpha\gamma - (\alpha^2 + \beta^2 + \gamma^2)^{\frac{1}{2}}\beta = 0 \end{cases}$$

Cette contrainte de contact droite-plan est non linéaire. De plus, le point tel que α, β et γ soient nuls est un point singulier pour la seconde égalité: elle n'est pas linéarisable en ce point. L'utilisation du filtre de Kalman étendu n'est pas possible.

Contact plan-plan

Le plan (O_2, x_2, y_2) est en contact avec Π si et seulement si O_2 appartient à Π et si z_2 est égal à z_1, soit:

$$\begin{cases} \tau_x = \tau_y = 0 \\ 1 - 2s^2(l^2 + p^2) = 1 \end{cases}$$

Soit:

$$\begin{cases} \tau_x = \tau_y = 0 \\ \alpha = \beta = 0 \end{cases}$$

Cette contrainte plan-plan est linéaire. On peut utiliser le filtre de Kalman pour réduire l'incertitude.

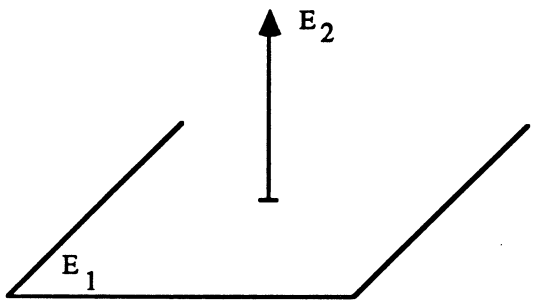


Figure 4.12 : Interprétation géométrique de P

4.2.2 Réduction de l'incertitude pour le contact plan-plan

La contrainte de contact plan-plan s'écrit:

$$P\vec{\varepsilon} = \vec{0} \text{ avec } P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.19}$$

P est une projection; sa matrice est symétrique et $PP = P$. L'équation 4.19 s'interprete comme une équation de mesure. Si μ est la valeur moyenne de ε, Δ sa matrice de covariance, le filtre de Kalman donne de nouvelles estimations μ^* et Δ^* telles que:

$$\begin{aligned} \mu^* &= (I - \Delta P(P\Delta P)^{-1}P)\mu \\ \Delta^* &= (I - \Delta P(P\Delta P)^{-1}P)\Delta \end{aligned} \tag{4.20}$$

Dans notre cas particulier, μ est nul (donc μ^* aussi). L'équation 4.20 donne une nouvelle matrice de covariance "plus petite".

Interprétation du résultat et optimisation des calculs

P est une projection orthogonale sur un sous-espace E_1 de R^6 de dimension p (figure 4.12). Dans une base orthogonale formée de vecteurs de E_1 et de vecteurs de E_2, la matrice de P se décompose en blocs:

$$P = \begin{pmatrix} I_p & 0 \\ 0 & 0_{n-p} \end{pmatrix}$$

où I_p et 0_{n-p} sont respectivement la matrice identité d'ordre p et la matrice nulle d'ordre $n-p$. Dans cette même base, Δ est de la forme:

$$\Delta = \begin{pmatrix} \Delta_{E_1} & \Delta_{E_{12}} \\ \Delta_{E_{12}} & \Delta_{E_2} \end{pmatrix}$$

On en tire:

$$P\Delta P = \begin{pmatrix} \Delta_{E_1} & 0 \\ 0 & 0 \end{pmatrix}$$

D'où:

$$(P\Delta P)^{-1} = \begin{pmatrix} \Delta_{E_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix}$$

Et enfin:

$$\Delta^* = \begin{pmatrix} 0 & 0 \\ 0 & \Delta_{E_2} \end{pmatrix}$$

On peut interpréter ce résultat de la façon suivante. P est la projection orthogonale sur le sous-espace E_1. $(I-P)$ est la projection orthogonale sur le sous-espace supplémentaire E_2. $(I-P)\vec{\varepsilon}$ est la projection de la variable aléatoire $\vec{\varepsilon}$ sur E_2. Sa matrice de covariance est (propriété 4.4):

$$\begin{aligned} \Delta_p &= (I-P)\Delta(I-P)^t \\ &\quad (I-P)\Delta(I-P) \end{aligned}$$

En faisant le calcul dans la même base que précédemment, il s'avère que Δ_p est égale à Δ^*. L'établissement d'un contact plan entre deux objets revient donc formellement à projeter toutes les erreurs de position relatives sur un sous-espace de l'espace des degrés de liberté. Ce résultat est valable pour les matrices de covariance et ne se généralise pas pour les valeurs moyennes. En effet μ^* calculé par le filtre est différent en général de $(I-P)\mu$, sauf si μ est nul.

Cette propriété nous sert pour calculer de façon simple l'incertitude résultant d'un contact plan: si donc on a deux repères $\Re_1$, $\Re_2$ liés par un transformation T_{12} et une incertitude Δ_{12} (exprimée implicitement dans $\Re_2$), si de plus les objets sont en contact plan de normale $n = (n_x\ n_y\ nz)^t$ dans $\Re_2$, pour calculer la matrice de la projection $(I-P)$ précédente, on écrit que:

- l'erreur en translation τ se projette sur le plan de normale n,
- l'erreur en rotation ω se projette sur l'axe dirigé par n.

La matrice de projection sur l'axe dirigé par n est:

$$P_1 = \begin{pmatrix} n_x^2 & n_x n_y & n_x n_z \\ n_x n_y & n_y^2 & n_y n_z \\ n_x n_z & n_y n_z & n_z^2 \end{pmatrix}$$

La matrice de projection sur le plan de normale n est:

$$P_1 = \begin{pmatrix} 1 - n_x^2 & n_x n_y & n_x n_z \\ n_x n_y & 1 - n_y^2 & n_y n_z \\ n_x n_z & n_y n_z & 1 - n_z^2 \end{pmatrix}$$

La matrice $(I - P)$ est donc égale à:

$$(I - P) = \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix}$$

La nouvelle matrice de covariance sera $(I - P)\Delta(I - P)$ qui est plus simple que celle donnée par le filtre de Kalman appliqué brutalement: en particulier, il n'y a pas d'inversion de matrice à faire.

Notations

On note *project* l'opérateur qui calcule les nouveaux paramètres de position relative entre deux objets.
$project[Way(\Re_1, \Re_2), \Re_3, n]$ est l'incertitude de $\Re_2$ par rapport à $\Re_1$ sachant que ces objets sont en contact planaire de normale n exprimée dans $\Re_3$.

5 Notations. Syntaxe des contraintes

Les notations utilisées dans ce chapitre pour les transformations (T_{ij}) et incertitudes (Δ_{ij}) sont trop concises pour pouvoir être utilisées dans la suite du rapport et en particulier dans le chapitre 5. C'est pourquoi nous introduisons ici de nouvelles notations, certes plus lourdes mais qui éviteront toute ambiguité. Ces notations nous servirons en outre à décrire la syntaxe des contraintes C_j définissant les conditions d'applicabilité des actions.

5.1 Notations

Arcs et chemins

Un arc liant un repère $\Re_i$ à un repère $\Re_j$ sera noté $Arc(\Re_i, \Re_j)$. On notera de plus:

- $Arc(\Re_i, \Re_j) : P$ la position nominale de $\Re_i$ par rapport à $\Re_j$.
- $Arc(\Re_i, \Re_j) : U$ l'incertitudes de position de $\Re_i$ par rapport à $\Re_j$.

On retient des notations similaires pour les chemins:

- $Way(\Re_k, \Re_l)$ est le chemin d'origine $\Re_k$ et d'extrémité $\Re_l$.
- $Way(\Re_k, \Re_l) : P$ est la position nominale de $\Re_k$ par rapport à $\Re_l$.
- $Way(\Re_k, \Re_l) : U$ est l'incertitudes de position de $\Re_k$ par rapport à $\Re_l$.

Composition et inversion d'arcs et chemins

On note par le signe $\otimes$ l'opération de composition des arcs et des chemins: si $Way(\Re_i, \Re_j)$ et $Way(\Re_j, \Re_k)$ sont deux chemins successifs du graphe (éventuellement ce sont des arcs élémentaires), le chemin $Way(\Re_i, \Re_k)$ formé de la succession de ces deux chemins est noté:

$$Way(\Re_i, \Re_k) = Way(\Re_i, \Re_j) \otimes Way(\Re_j, \Re_k)$$

Par exemple, le résultat de la proposition 4.9 concernant la composition des transformations nominales s'écrit:

$$[Way(\Re_i, \Re_j) \otimes Way(\Re_j, \Re_k)] : P = Way(\Re_i, \Re_j) : P * Way(\Re_j, \Re_k) : P$$

On note $Way(\Re_i, \Re_j)^{\ominus 1}$ le chemin inverse du chemin $Way(\Re_i, \Re_j)$. Ainsi le résultat de la proposition 4.6 pourra s'écrire:

$$Way(\Re_i, \Re_j)^{\ominus 1} : P = [Way(\Re_i, \Re_j) : P]^{-1}$$

5.2 Syntaxe des contraintes

Une contrainte exprime le fait qu'une incertitude relative entre deux repères est inférieure à un certain seuil. Elle est exprimée par une expression du type

$$< chemin >: U \subseteq \Delta$$

où Δ est une incertitude donnée sous forme de matrice de covariance, $\subseteq$ étant la relation d'ordre décrite au paragraphe 2.4.4.
$< chemin >$ peut bien sûr être un chemin simple de la forme $Way(\Re_i, \Re_j)$. Cela peut être aussi:

- le résultat d'une composition $\otimes$ de deux chemins,
- l'inversion d'un chemin,
- le résultat de la fusion d'informations

En notation B.N.F, cette syntaxe s'écrit:

$$\begin{aligned} < contrainte > &::= < chemin >: U \subseteq \Delta \\ < chemin > &::= Way(< repere >, < repere >) | \\ &\quad < chemin > \otimes < chemin > | \\ &\quad < chemin >^{\ominus 1} | \\ &\quad merge[< chemin >, \ldots] \end{aligned}$$

Exemple

L'expresssion suivante est une contrainte:

$$[Arc(C, table) \otimes Arc(robot, table)^{\ominus 1} \otimes Way(robot, P)] : U \subseteq \Delta$$

Chapitre 5

VCP pour un modèle particulier d'actions

Le chapitre 3 a montré l'inadéquation partielle des langages de niveau effecteur existants à la preuve de programme. Le but de ce chapitre est de présenter un langage de niveau effecteur adapté, de décrire sa sémantique, et de décrire les mécanismes de propagation avant des incertitudes et de propagation arrière des contraintes. Les instructions de ce langage ont été choisies de façon à être de sémantique homogène. Nous ne considérons pas de constructions algorithmiques évoluées (boucles, conditionnelles, ...). La preuve de ces constructions a été étudiée dans le cadre de l'informatique théorique classique.

1 Choix d'un langage particulier

Comme tout langage de programmation de robots de niveau effecteur, le langage que nous avons retenu présente les caractéristiques suivantes que nous allons successivement détailler:

- les types de données spécialisés et les opérations associées aptes à la représentation et à la manipulation de données géométriques
- les instructions de déplacement du manipulateur
- les instructions d'action du préhenseur
- les instructions de lecture de capteurs et d'interaction avec l'environnement.

1.1 Types de données et opérations spécialisées

Nous avons retenu les mêmes types de données que dans le langage LM. Cependant, nous n'avons pas retenu les conventions du langage LM quant à la définition de la position d'un objet (voir chapitre 3 paragraphe 2). En résumé, nous avons retenu les types suivants:

- le type *REPERE* servant à la représentation d'un objet solide.
- le type *TRANSFORMATION* servant à représenter la position d'un repère
- le type *VECTEUR*
- les types *REEL*, *ENTIER*.

A ces types spécialisés sont associées des opérations spécialisées (composition de transformations, calcul de l'inverse ...).

Contrairement aux conventions adoptées dans LM, la position d'un repère n'est pas systématiquement définie par rapport à un repère de référence absolu, mais par rapport à un autre repère quelconque, et susceptible de varier au cours d'une manipulation.

De plus, il existe trois repères prédéfinis:

- le repère *WSPACE* qui est le repère de référence de l'espace de travail du manipulateur. C'est l'équivalent du repère *STATION* de LM.
- le repère *ROBOT* qui représente le socle du robot (contrairement à LM où *ROBOT* représente le préhenseur). Ici *ROBOT* est *toujours* défini par rapport au repère *WSPACE*.
- le repère *GRIPPER* qui représente le préhenseur. *GRIPPER* est *toujours* défini par rapport au repère *ROBOT*.

1.2 Liaison de repères

Considérant les difficultés de définir la sémantique des instructions de liaisons de repères (chapitre 3, paragraphe 2), leur inadéquation à représenter des liaisons unilatérales, nous n'avons pas retenu d'instruction de liaison de repère. Nos conventions quant à la position des repères définis par rapport à des repères de référence quelconques nous permet de façon naturelle de nous passer de liaisons pour les opérations de calibration.

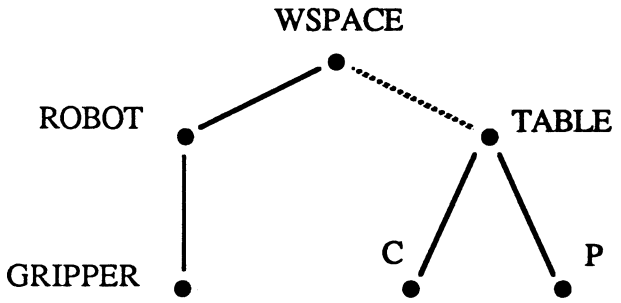


Figure 5.1 : Définition des positions relatives des repères lors de la calibration

En effet, si on considère de nouveau la situation de la figure 3.3, les différents repères seront définis les uns par rapport aux autres conformément à la figure 5.1. Une mise à jour simple de la position de $TABLE$ par rapport à $WSPACE$ mettra à jour également la position des repères définis directement ou indirectement par rapport à $TABLE$.

La représentation d'un objet par plusieurs repères qui n'est qu'un confort de programmation n'est pas envisagée; elle peut cependant se faire en choisissant un repère principal et en définissant les autres par rapport à celui-ci. Les créations et destructions de structures rigides temporaires, occasionnées par l'éxécution des instructions de saisie et de lâcher d'objets seront contenues dans la sémantique de ces actions et leur gestion n'est pas à la charge du programmeur.

1.3 Instruction de déplacement

Un déplacement est commandé par l'instruction:

$$MOVE \ < obj > \ TO \ < obj_2 > *T$$

où:

- $< obj >$ est un repère quelconque autre que $ROBOT$ et $WSPACE$ et dont le repère de référence direct ou indirect est $GRIPPER$,
- $< obj_2 > *T$ définit la position d'arrivée.

Cette instruction a pour effet de modifier la position du repère $GRIPPER$ (par rapport à $ROBOT$) de telle sorte qu'après le déplacement, le repère $< obj >$ coïncide avec

< *position* >. Ce déplacement est supposé être controlé en position seulement. Du fait que pour décrire les actions, on ne représente les objets que par des repères (voir chapitre 4), on ne considère pas de mouvement à compliance dont la sémantique ne peut être définie sur ce modèle.

1.4 Action du préhenseur

Deux instructions permettent de saisir et de lâcher des objets. Une action de saisie est commandée par l'instruction:

$$GRASP \ < obj >$$

où $< obj >$ est un repère autre que $WSPACE$, $ROBOT$ ou $GRIPPER$. L'emploi de cette instruction sous-entend que le préhenseur a été préalablement placé à la position adéquate: l'éxécution de cette instruction n'occasionne théoriquement aucun déplacement.

L'action de relâcher un objet est commandée par l'instruction:

$$DROP \ < obj1 > ONTO \ (< obj2 >, < contact >)$$

où:

- $< obj1 >$ est l'objet tenu par le préhenseur: c'est un repère autre que $WSPACE$, $ROBOT$ ou $GRIPPER$.
- $< obj2 >$ est l'objet sur lequel est posé $< obj1 >$. C'est le nouveau repère de référence de $< obj1 >$. C'est un repère quelconque autre que $WSPACE$, $ROBOT$ ou $GRIPPER$.
- < *contact* > spécifie le type de contact prévu entre $< obj1 >$ et $< obj2 >$ après le lâcher. Cette information est nécessaire pour calculer l'incertitude résultant de l'action. < *contact* > est défini par son type (surfacique ou lineïque) et par un point et un vecteur (qui donnent la position et l'orientation du plan ou de la droite de contact).

De même que pour la prise, le robot est supposé être à la position adéquate et l'éxécution de cette instruction n'occasionne théoriquement aucun déplacement.

1.5 Lecture de capteurs

Bien que dans les langages de niveau effecteur existant, il n'y ait pas d'instruction ou de procédure standard permettant d'acquérir la position d'objets en cours de manipulation, nous supposerons que nous disposons d'un capteur de vision apte à remplir cette fonction. La mesure de la position relative de objets est commandée par l'instruction:

$$LOCATE < obj1 > RELATIVELY\ TO < obj2 >$$

où $< obj1 >$ et $< obj2 >$ sont deux repères quelconques. Cette instruction a pour effet de mettre à jour la position relative de $< obj1 >$ par rapport à $< obj2 >$. L'incertitude résultante est donnée par la précision du capteur..

2 Description de l'effet des actions. Propagation des incertitudes

Dans ce paragraphe, nous allons décrire l'effet de l'éxécution de chacune des instructions introduites précédemment. En même temps nous donnerons précisément le mécanisme de mise à jour du modèle de l'environnement qui nous permet de faire la propagation descendante des incertitudes. Ce mécanisme de mise à jour sera décrit sous forme de "delete-list" et "add-list".

2.1 Instruction de déplacement

Considérons l'instruction:

$$MOVE\ obj1\ TO\ obj2 * T$$

$obj1$ et $obj2$ sont deux repères, T est une transformation (éventuellement l'unité). Nous supposons que l'incertitude de position de $GRIPPER$ après déplacement, conditionnée par la résolution du manipulateur est indépendante de la position d'arrivée et égale à U_{move}[1].

[1]Ceci n'est qu'une convention; on pourrait prendre U_{move} dépendant de la position d'arrivée

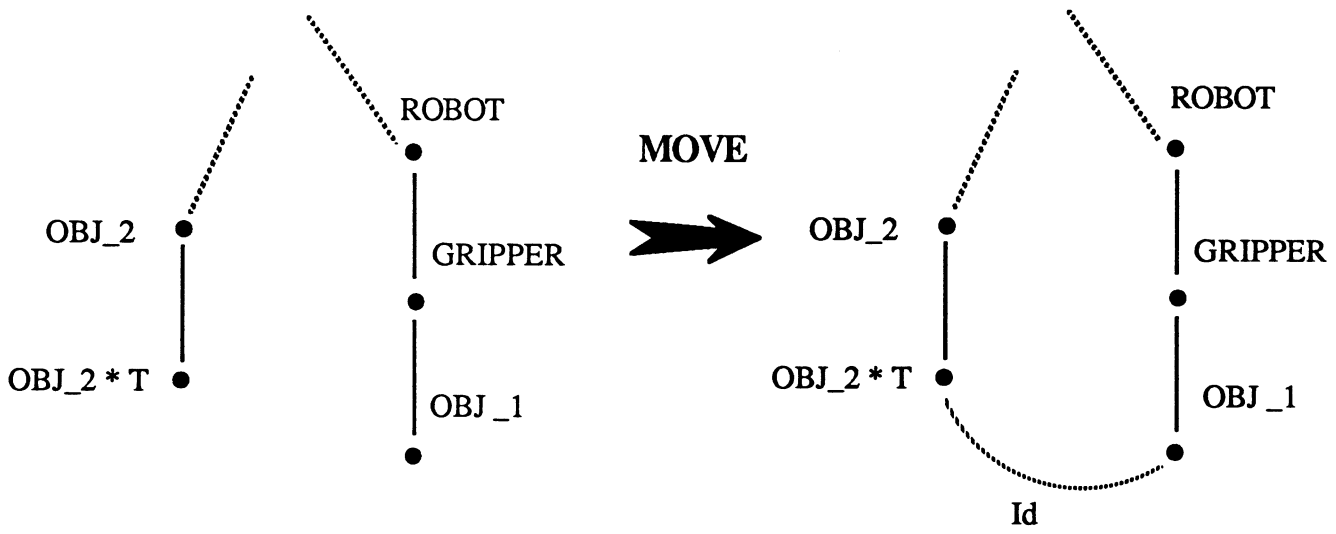


Figure 5.2 : Effet de l'éxécution d'un déplacement

L'effet principal de cette instruction est de modifier la position relative de $GRIPPER$ par rapport à $ROBOT$ pour amener en correspondance les repères $obj1$ et $obj2 * T$. La nouvelle position relative de $GRIPPER$ par rapport à $ROBOT$, soit la transformation qui amène le repère $GRIPPER$ sur $ROBOT$ est la composition (voir figure 5.2):

- de la position relative de $obj2$ par rapport à $ROBOT$
- de la transformation T
- de la position relative de $GRIPPER$ par rapport à $obj1$ qui est l'inverse de la position relative de $obj1$ par rapport à $GRIPPER$.

L'autre effet de l'instruction est de modifier l'incertitude relative de $GRIPPER$ par rapport à $ROBOT$ qui prend la valeur U_{move}. Les autres positions et incertitudes ne sont pas modifiées.

Le calcul de l'état résultant est donc donné formellement par:

delete-list:
$ARC(ROBOT, GRIPPER) : P = P_0$
$\wedge$ $ARC(ROBOT, GRIPPER) : U = U_0$
add-list:

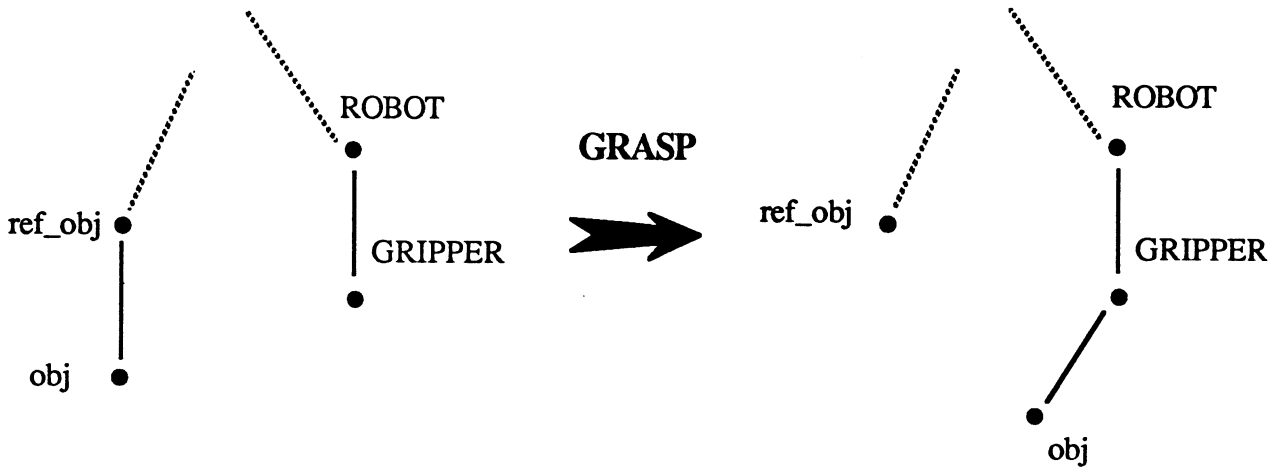


Figure 5.3 : Effet de l'éxécution d'une saisie d'objet

$$ARC(ROBOT, GRIPPER) : P = WAY(ROBOT, obj2) : P * T*$$
$$WAY(obj1, GRIPPER) : P$$
$$\wedge \quad ARC(ROBOT, GRIPPER) : U = U_{move}$$

2.2 Saisie de l'objet

Considérons l'instruction

$$GRASP\ obj$$

Supposons que la position de *obj* est définie par rapport à un repère ref_obj (voir figure 5.3).

Même si théoriquement l'opération de saisie n'occasionne pas de déplacement, elle s'accompagne de petits mouvements parasites de l'objet saisi *obj* par rapport à son support ref_obj. Ces déplacements parasites ont pour effet de modifier l'incertitude de position associée à l'arc $ARC(ref\_obj, obj)$ et l'incertitude de position relative entre $GRIPPER$ et *obj*. Plus précisément, on considére que l'opération de saisie a pour effet de projeter le volume d'incertitude relatif de *obj* par rapport à $GRIPPER$ sur le plan des mors. L'opération de saisie a également pour effet de lier l'objet au préhenseur: autrement dit, la référence de position de l'objet n'est plus son support mais devient

le préhenseur. On décrit donc l'effet complet de l'action de saisie par les opérations suivantes sur le modèle de l'environnement:

1. calcul de la position et de l'incertitude relative de *obj* et de $GRIPPER$
2. destruction de l'arc $ARC(ref\_obj, obj)$
3. création d'un arc $ARC(GRIPPER, obj)$ ayant pour position nominale la position relative calculée au point 1 et pour incertitude la projection sur le plan des mors de l'incertitude calculée au point 1.

Formellement, cela s'écrit:

delete-list:
$ARC(ref\_obj, obj) : P = P_0$
$\wedge$ $ARC(ref\_obj, obj) : U = U_0$
add-list:
$ARC(GRIPPER, obj) : P = WAY(GRIPPER, obj) : P$
$\wedge$ $ARC(GRIPPER, obj) : U = project[WAY(GRIPPER, obj), GRIPPER, (0, 1, 0)$

2.3 Lâcher de l'objet

Considérons l'instruction:

$$DROP\ obj1\ ONTO\ (obj2, contact)$$

La situation géométrique est celle de la figure **5.4**.

De même que pour l'opération de saisie, lâcher un objet n'occasionne théoriquement pas de déplacement mais s'accompagne généralement de mouvements parasites. Ici ce sont la position et l'incertitude de $obj1$ par rapport à $GRIPPER$ qui sont modifiés. Le volume d'incertitude de $obj1$ par rapport à $obj2$ se trouve projeté de façon définie par le contact. Ultérieurement, $obj2$ sera le repère de référence de $obj1$. L'effet complet d'une action de lâcher d'objet est donc décrit par les opérations suivantes sur le modèle de l'environnement:

1. calcul de la position et de l'incertitude relative de $obj1$ et de $obj2$
2. destruction de l'arc $ARC(GRIPPER, obj1)$
3. création d'un arc $ARC(obj2, obj1)$ ayant pour pour position nominale la

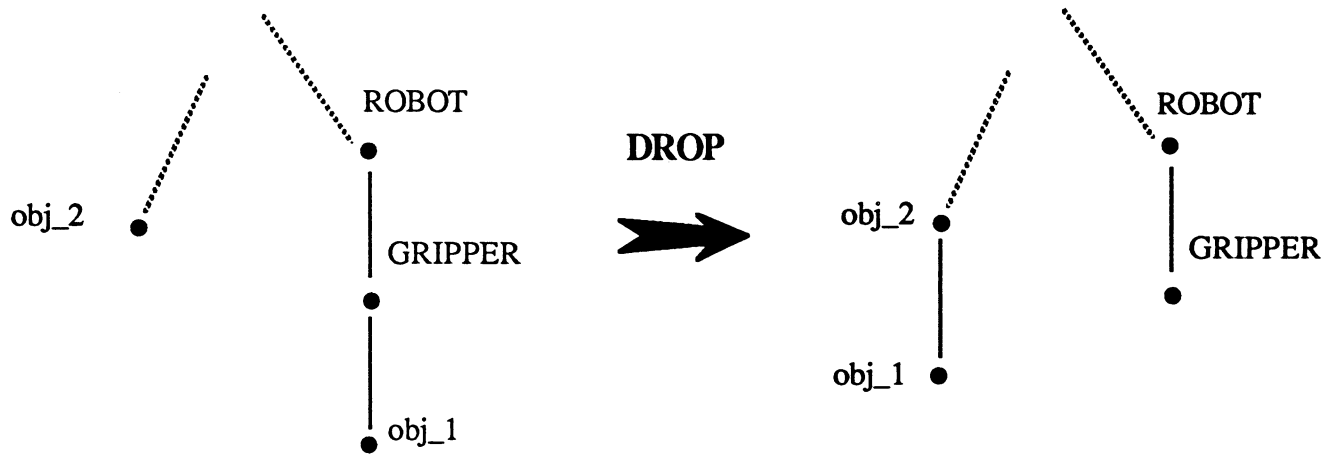


Figure 5.4 : Situation géométrique avant le lâcher d'objet

4. calcul de la position et de l'incertitude relative de $obj1$ et de $obj2$ et pour incertitude la projection définie par le contact de l'incertitude calculée au point 1.

Formellement, cela s'écrit:
delete-list:

$$\begin{aligned} & ARC(GRIPPER, obj1) : P \;=\; P_0 \\ \wedge\; & ARC(GRIPPER, obj1) : U \;=\; U_0 \end{aligned}$$

add-list:

$$\begin{aligned} & ARC(obj2, obj1) : P \;=\; WAY(obj2, obj1) : P \\ \wedge\; & ARC(obj2, obj1) : U \;=\; project[WAY(obj2, obj1), obj2, n\}] \end{aligned}$$

2.4 Lecture de capteurs

L'instruction:

$$LOCATE\ obj1\ RELATIVELY\ TO\ obj2$$

commande une mesure de la position relative de $obj2$ par rapport à $obj1$ avec une incertitude donnée. Cela permet de mettre à jour les positions et incertitudes associées pour

tous les arcs du chemin entre $obj1$ et $obj2$. Formellement, le modèle de l'environnement est mis à jour à l'aide de l'opérateur "merge" (chapitre 4):

Pour tout arc $ARC(obj_k, obj_l)$ du chemin $WAY(obj1, obj2)$:
delete-list:

$$\begin{aligned} & ARC(obj_k, obj_l) : P &=& \ T_0 \\ \wedge\ & ARC(obj_k, obj_l) : U &=& \ U_0 \end{aligned}$$

add-list:

$$ARC(obj_k, obj_l) = merge[Arc(obj_k, obj_l), obj_1, obj_2, T, \Delta]$$

2.5 Application sur un exemple

Le modèle précédent des actions nous permet de faire la propagation avant des incertitudes. Nous allons le faire dans ce paragraphe sur un exemple de programme particulier. Le but de cet exemple est de montrer le principe de la méthode plus que les calculs. C'est pourquoi, nous retiendrons ici une représentation très simplifiée des incertitudes. Nous ne considérons que des mouvements en translation et nous ne considérons pas les incertitudes de position en rotation. Nous prendrons comme incertitudes des matrices 3×3 diagonales. Pour alléger les notations nous noterons $|\sigma_x^2|\sigma_y^2|\sigma_z^2|$ la matrice de covariance diagonale de coefficients σ_x^2, σ_y^2, σ_z^2 sur la diagonale.

Ces restrictions entrainent une grande simplification au niveau des opérateurs de composition et d'inversion d'incertitudes puisque les transfornations n'interviennent plus dans ces calculs. En conséquence, pour calculer la composition des incertitudes le long d'un chemin dans le graphe, il suffira d'additionner terme à terme les coefficients sur la diagonale des matrices de covariance.

Considérons la manipulation vue dans l'introduction (figure **0.1**) consistant à prendre un piston et à l'insérer dans un cylindre. En supposant que les positions initiales des des diffèentes pièces sont connues, la tâche prévue sera remplie par l'éxécution du programme suivant:

```
MOVE gripper TO P * Transl(z,200.);
```

```
GRASP P;
MOVE P TO C * Transl(z,200.);
DROP P ONTO (C,plane,(0,0,0),(0,0,1));
MOVE gripper TO gripper * Transl(z,500.);
```

Nous supposons que nous disposons d'un manipulateur dont la précision est définie par une variance de 0.1 dans les directions x et y et 0.2 dans la direction z. Nous supposons disposer également d'un capteur de vision placé au dessus de la scène qui donne la position d'un objet avec une précision définie par une variance de 0.3 dans les directions x et y et de 10 dans la direction z. Le jeu de montage entre le piston et le cylindre correspond à une variance de 1 dans les directions x et y. Nous supposerons que l'insertion est assurée de réussir si de plus l'incertitude de position du piston dans la direction z est inférieure à 3. Nous supposons de la même façon que l'opération de saisie sera réalisée si l'incertitude de position de l'objet par rapport à la pince est inférieure à 3 dans chaque direction. Nous avons alors le programme contraint suivant à vérifier:

```
CP = { ( (vrai),{ MOVE gripper TO P * Transl(z,200.) })
       ( C1, { GRASP P })
       ( (vrai),{ MOVE P TO C * Transl(z,200.) })
       ( C3, { DROP P ONTO (C,plane,(0,0,0),(0,0,1)) })
       ( (vrai),{ MOVE gripper TO gripper * Transl(z,500.) })}
```

Où:

$$C_1 = [WAY(P, GRIPPER) : U \subseteq |3.0|3.0|3.0|]$$
$$C_3 = [WAY(P, C) : U \subseteq |1.0|1.0|1.0|]$$

L'état initial W_0 de la manipulation est celui représenté par le graphe de la figure 5.5. Sur cette figure, ainsi que sur les suivantes, nous ne portons que les incertitudes et non les positions pour en préserver la lisibilité.

L'état W_1 résultant de l'éxécution du premier mouvement ne diffère de W_0 que par la valeur de la transformation nominale entre $ROBOT$ et $GRIPPER$. Avec nos conventions, l'incertitude de position de $GRIPPER$ n'est pas changée. Cet état étant calculé, on vérifie la satisfaction de la condition C_1:

$$WAY(P, gripper) : U = [ARC(P, Table) \otimes ARC(Table, Wspace) \otimes ARC(WSpace, robot) \otimes ARC(robot, gripper)] : U$$

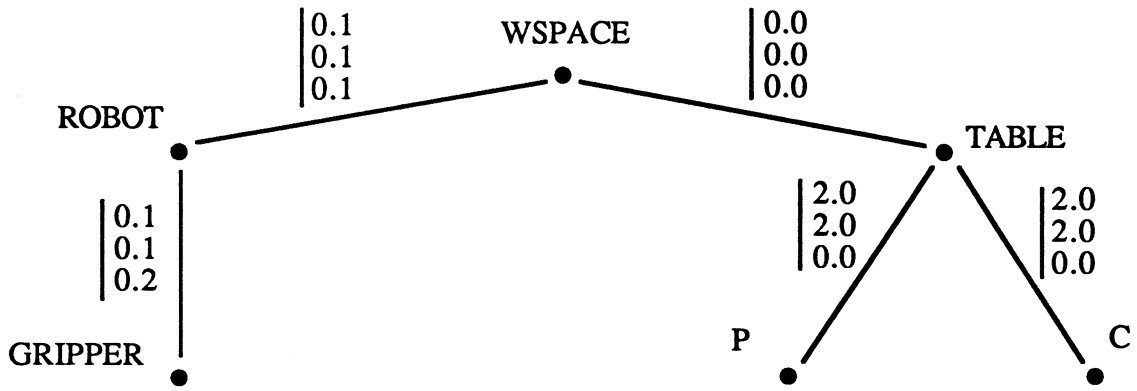


Figure 5.5 : Etat initial de la manipulation

$$= |2+0+0.1+0.1|2+0+0.1+0.1|0+0+0.1+0.2|$$
$$= |2.2|2.2|0.3|$$

La condition est bien satisfaite et nous pouvons calculer l'état suivant.
L'état W_2, résultant de l'éxécution de la saisie du piston est calculée en appliquant les règles données précédemment. La structure du graphe représentant l'environnement est donnée par la figure 5.6.

L'état W_3 résultant de l'éxécution du second mouvement ne diffère de W_2 que par la valeur de la transformation nominale entre $ROBOT$ et $GRIPPER$. L'incertitude de position n'est pas changée. Cet état étant calculé, on peut vérifier la satisfaction de la contrainte C_3:

$$\begin{aligned} WAY(P,C):U &= [ARC(P,gripper) \otimes ARC(gripper,robot) \otimes \\ &\quad ARC(robot,WSpace) \otimes ARC(Wspace,Table) \otimes ARC(Table,C)] : U \\ &= 2.2+0.1+0.1+0+2|0+0.10.1+0+2|0.3+0.2+0.1+0+0| \\ &= |2.2|2.2|0.3| \end{aligned}$$

La condition C_3 n'est pas satisfaite. Le programme n'est pas correct.

3 Propagation arrière des contraintes

Formellement, l'éxécution de l'une des instructions $MOVE$, $GRASP$, $DROP$ et $LOCATE$ est équivalente à l'affectation de certains paramètres du modèle. Par exemple, l'éxécution

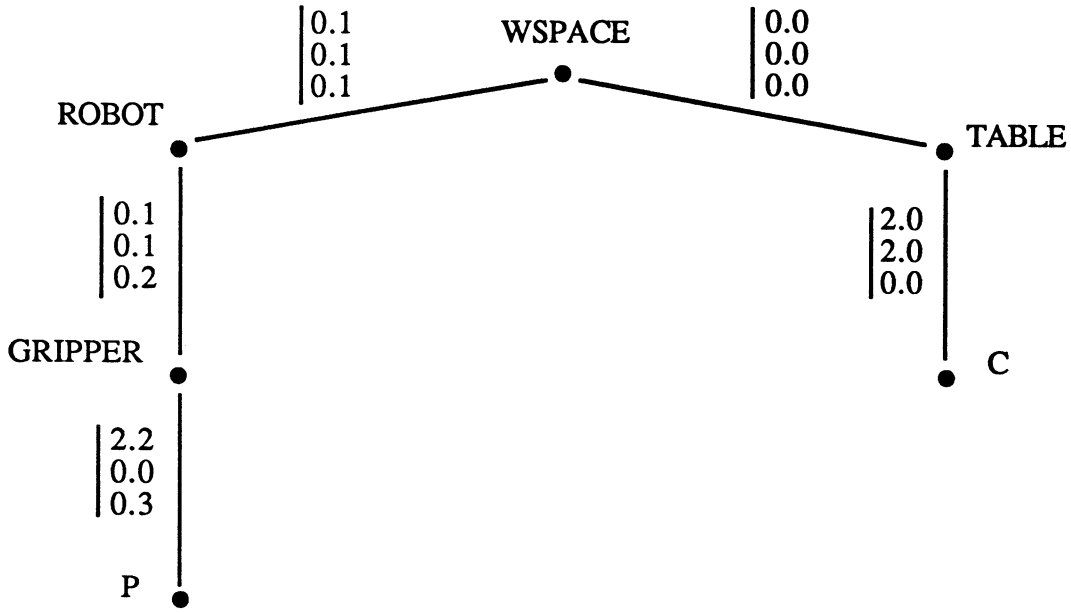


Figure 5.6 : Etat W_2

de "MOVE gripper TO position" est équivalent à l'affectation de nouvelles valeurs à la position et à l'incertitude de *gripper* par rapport à *robot*. La plus faible précondition d'une affectation et d'une condition F est donnée par (voir annexe A):

$$pfpre(\{x := expr\}, F) = F(x|expr)$$

En utilisant cet axiome, la propagation arrière des contraintes d'incertitude qui est une suite de calculs de plus faibles préconditions se fait principalement sous forme de reécriture de termes dans une expression. Le seul point délicat vient du fait que les conditions F que nous manipulons sont des expressions s'évaluant à des incertitudes de positions relatives et que leur évaluation dépend de la définition des positions relatives des objets entre eux, autrement dit de la structure du graphe représentant l'univers. Ainsi par exemple pour évaluer la condition C_3:

$$WAY(P, C) : U \subseteq U_3$$

tirée de l'exemple précédent, n'interviennent pas les mêmes termes dans l'état W_3 (la structure est donnée par la figure 5.6) ou dans l'état W_0 (figure 5.5). Dans le premier

cas intervient en particulier la position du préhenseur alors qu'elle n'intervient pas dans l'autre. De ce fait, pour calculer une précondition, nous aurons besoin de la structure du graphe représentant l'environnement *avant* l'éxécution de l'instruction. Ceci sera fait en gardant un historique de cette structure lors de la propagation avant.

3.1 Instruction de déplacement

L'instruction

$$MOVE\ obj1\ TO\ obj2 * T$$

a pour effet de modifier l'arc $ARC(robot, gripper)$. La plus faible précondition de cette instruction pour une condition C est obtenue en substituant chaque terme $WAY(obj_i, obj_j)$ de C:

- si $ARC(robot, gripper)$ appartient au chemin $WAY(obj_i, obj_j)$ (cas de la figure **5**.7) par:
  $WAY(obj_i, robot) \otimes A_{move} \otimes WAY(gripper, obj_j)$,
- si $ARC(robot, gripper)$ appartient au chemin $WAY(obj_j, obj_i)$ (cas de la figure **5**.8) par:
  $WAY(obj_i, gripper) \otimes A_{move}^{-1} \otimes WAY(robot, obj_j)$
- dans les autres cas, $WAY(obj_i, obj_j)$ reste inchangé.

où A_{move} est l'arc représentant la valeur de l'arc $ARC(robot, gripper)$ après le déplacement, soit:
$A_{move} : P = WAY(robot, obj2) : P * T * WAY(obj1, gripper) : P$
et $A_{move} : U = U_{move}$

3.2 Instruction de saisie

L'instruction

$$GRASP\ obj$$

a pour effet de modifier l'incertitude de position relative de *obj* par rapport à *gripper*. La plus faible précondition de cette instruction pour une condition C est obtenue en substituant chaque terme $WAY(obj_i, obj_j)$ dans C:

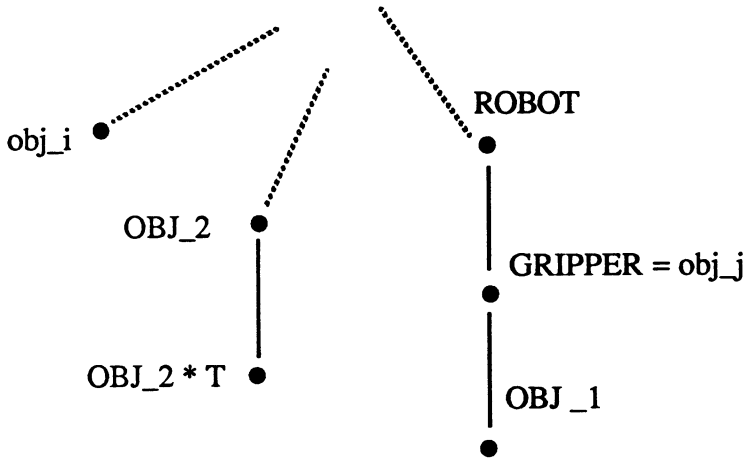


Figure 5.7 : $ARC(robot, gripper)$ appartient au chemin $WAY(obj_i, obj_j)$

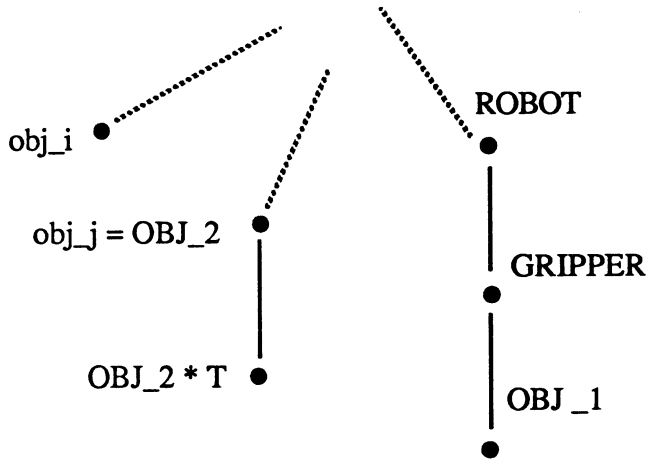


Figure 5.8 : $ARC(robot, gripper)$ n'appartient pas au chemin $WAY(obj_j, obj_i)$

- si $ARC(gripper, obj)$ appartient au chemin $WAY(obj_i, obj_j)$ par: $WAY(obj_i, gripper) \otimes A_{grasp} \otimes WAY(obj, obj_j)$,

- si $ARC(gripper, obj)$ appartient au chemin $WAY(obj_j, obj_i)$ par: $WAY(obj_i, obj) \otimes A^{-1}_{grasp} \otimes WAY(gripper, obj_j)$
- dans les autres cas, $WAY(obj_i, obj_j)$ reste inchangé.

où A_{grasp} est l'arc représentant la valeur de l'arc $ARC(robot, gripper)$ après la saisie,

soit:
$A_{grasp} : P = WAY(gripper, obj) : P$
et $A_{grasp} : U = project[...]$ (voir paragraphe 2.2)

3.3 Instruction de lâcher

L'instruction

$$DROP\ obj1\ ONTO\ (obj2, contact)$$

a pour effet de modifier l'incertitude de position relative de $obj1$ par rapport à $obj2$. La plus faible précondition de cette instruction pour une condition C est obtenue en substituant chaque terme $WAY(obj_i, obj_j)$ dans C:

- si $ARC(obj1, obj2)$ appartient au chemin $WAY(obj_i, obj_j)$ par:
  $WAY(obj_i, obj2) \otimes A_{drop} \otimes WAY(obj1, obj_j)$,

- si $ARC(obj1, obj2)$ appartient au chemin $WAY(obj_j, obj_i)$ par:
  $WAY(obj_i, obj1) \otimes A_{drop}^{-1} \otimes WAY(obj2, obj_j)$

- dans les autres cas, $WAY(obj_i, obj_j)$ reste inchangé.

où A_{drop} est l'arc représentant la valeur de l'arc $ARC(obj1, obj2)$ après le lâcher, soit:
$A_{drop} : P = WAY(obj2, obj1) : P$
et $A_{drop} : U = project[...]$ (voir paragraphe 2.3).

3.4 Instruction de lecture de capteur

L'instruction

$$LOCATE\ obj1\ RELATIVELY\ TO\ obj2$$

a pour effet de modifier tout arc du chemin $WAY(obj1, obj2)$. La plus faible précondition de cette instruction pour une condition C est obtenue en substituant chaque terme $WAY(obj_i, obj_j)$ dans C:

- si $WAY(obj1, obj2)$ a une intersection non vide avec $WAY(obj_i, obj_j)$ (cas de la figure 5.9) et $WAY(obj1, obj2) \cap WAY(obj_i, obj_j) = WAY(obj_k, obj_l)$,
  $WAY(obj_i, obj_j)$ sera remplacé par:
  $WAY(obj_i, obj_k) \otimes A_{locate} \otimes WAY(obj_l, obj_j)$

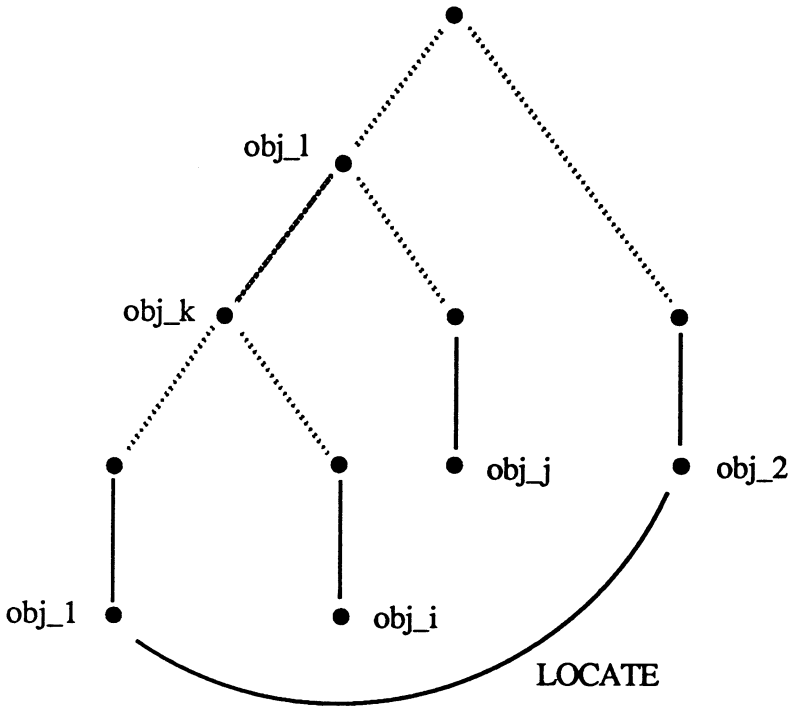


Figure 5.9 : $WAY(obj1, obj2)$ a une intersection non vide avec $WAY(obj_i, obj_j)$

- si $WAY(obj1, obj2)$ a une intersection non vide avec $WAY(obj_j, obj_i)$ (cas de la figure 5.10) et $WAY(obj1, obj2) \cap WAY(obj_j, obj_i) = WAY(obj_k, obj_l)$, $WAY(obj_i, obj_j)$ sera remplacé par:
  $WAY(obj_i, obj_l) \otimes A^{-1}_{locate} \otimes WAY(obj_k, obj_j)$.
- dans les autres cas, $WAY(obj_i, obj_j)$ reste inchangé.

où A_{locate} est l'arc fictif défini par:

$$A_{locate} = merge[ARC(obj_k, \ldots), obj1, obj2, T, \Delta] \otimes \\ \ldots \otimes merge[ARC(\ldots, obj_l), obj1, obj2, T, \Delta]$$

3.5 Application sur un exemple

Reprenons l'exemple précédent. La contrainte C_3 n'est pas satisfaite. On calcule ses plus faibles préconditions C_3^2, C_3^1 et C_3^0.
C_3^2 est la plus faible précondition de C_3 et de l'instruction "MOVE P to C * Transl(z,200.)".

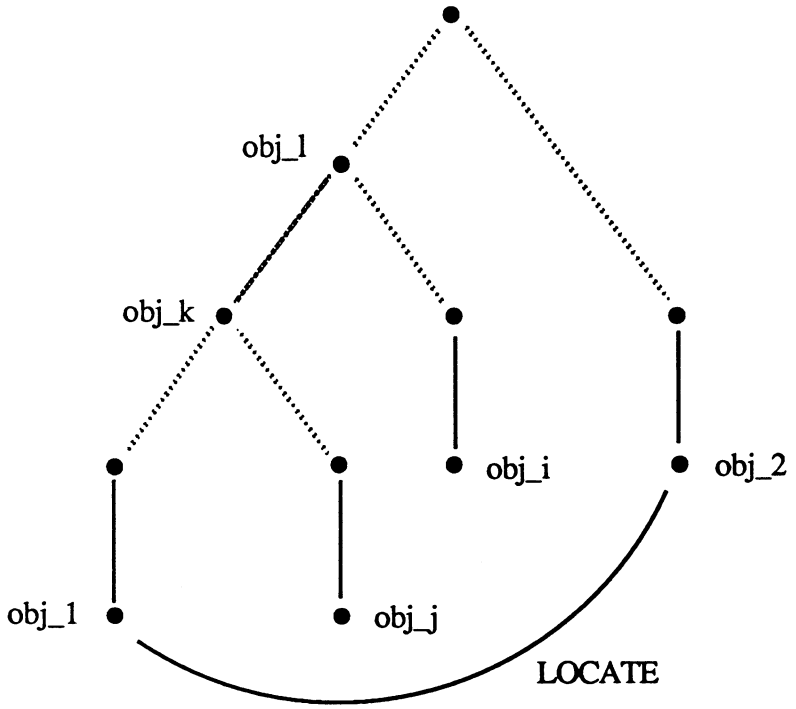


Figure 5.10 : $WAY(obj1, obj2)$ a une intersection non vide avec $WAY(obj_j, obj_i)$

De la règle donnée au paragraphe 3.1, et sachant que $ARC(robot, gripper)$ appartient $WAY(C, P)$, on obtient:

$$\begin{aligned} C_3^2 &= pfpre(\{MOVE\ P\ TO\ C * Transl(z, 250.)\}, C_3) \\ &= (WAY(P, gripper) \otimes A_{move}^{-1} \otimes WAY(robot, C)) : U \subseteq |1.0|1.0|3.0| \end{aligned}$$

Avec:

$$\begin{aligned} A_{move}^{-1} : P &= (WAY(robot, C) : P * Transl(z, 250.) * WAY(P, gripper) : P)^{-1} \\ &= ARC(gripper, P) : P * Transl(z, -250.) * WAY(C, robot) : P \\ A_{move}^{-1} : U &= |0.1|0.1|0.2| \end{aligned}$$

C_3^1 est la plus faible précondition de C_3^2 et de l'instruction "GRASP P". Elle est calculée par la règle donnée au paragraphe 3.2. Des trois termes de la condition C_3^2, seul le terme $WAY(P, gripper)$ sera modifié. La règle s'applique avec obj égal à P, obj_i égal à P, et obj_j égal à $gripper$. D'où:

$$C_3^1 = A_{grasp}^{-1} \otimes A_{move}^{-1} \otimes WAY(robot, C)) : U \subseteq |1.0|1.0|3.0|$$

Avec:

$$A^{-1}_{grasp} : P = \; WAY(P, gripper) : P$$
$$A^{-1}_{grasp} : U = \; project[WAY(obj2, obj1) : U, \{WAY(WSPACE, obj2), plane, (0,0,0), (0,1,0)\}]$$

C^0_3 est la plus faible précondition de C^1_3 et de l'instruction "MOVE gripper TO P * Transl(z,200.)". Or dans C^1_3 ne figure aucun terme contenant $ARC(robot, gripper)$. C^0_3 est équivalente à C^1_3.

La satisfaction d'une de ces contraintes par modification locale du programme assurera son éxécutabilité.

Chapitre 6

Validations expérimentales de l'approche. Extension possible.

Après avoir présenté en détail notre approche, le but de ce chapitre est d'en faire l'analyse. Dans la première partie est analysée la validité de notre approche par rapport au but que l'on s'était fixé (chapitre 1). La seconde partie présente les extensions fonctionnelles que l'on peut faire sans changer beaucoup notre approche, à savoir la prise en compte des incertitudes survenant *pendant* l'éxécution d'une action.

1 Analyse de la validité de l'approche

Il ne nous a pas été possible de tester expérimentalement notre approche de façon globale sur un manipulateur réel. Cependant nous avons procédé à une implantation qui a permis de vérifier séparément un certain nombre des différentes hypothèses que nous avons utilisées. Cela nous a permis d'apprécier la validité de notre approche pour résoudre le problème de la vérification/correction de programme. Nous entendons par "validité" de notre approche, son aptitude à déclarer comme "corrects" ou "incorrects" les programmes qui le seront effectivement lors d'une éxécution par un manipulateur réel. De façon grossière, on peut dire qu'on aura tendance à déclarer correct un programme qui ne l'est pas si on sous-estime les incertitudes, et inversement à déclarer abusivement incorrect un programme si on les surestime. Il est donc d'une importance primordiale d'estimer les incertitudes à leur juste valeur et de faire sur ces incerti-

tudes des calculs qui soient exacts. Nous allons passer en revue les différents points où pourraient se produire une mauvaise estimation des incertitudes due à des hypothèses abusives ou des calculs trop approchés:

- choix de la représentation des incertitudes, et adaptation des paramètres de cette représentation aux mouvements d'un robot et aux résultats d'un capteur,
- calcul des incertitudes au sein du modèle,
- évitement des cycles dans le modèle et intégration de données provenant de capteurs,
- propagation des incertitudes à travers les actions.

1.1 Choix d'une représentation des incertitudes

1.1.1 Les différentes représentations possibles

Nous avons vu qu'on pouvait distinguer deux formes différentes de représentation des incertitudes [MP88]:

- une représentation "ensembliste" pour laquelle on définit une incertitude comme l'ensemble des erreurs possibles,
- une représentation "probabiliste" pour laquelle on définit une loi de probabilité sur l'ensemble des erreurs.

Pour établir un lien entre les deux types de représentation, on peut dire que l'ensemble retenu pour la première s'identifie au support[1] de la loi de probabilité de la seconde. Il est à remarquer que le deuxième type de représentation est plus riche que le premier.

Nous pensons que la différence entre les deux modes de représentation n'est pas très importante tant que l'on a pas à faire beaucoup de calculs sur les incertitudes (en particulier tant que l'on a pas à faire des compositions) et ceci d'autant plus que nous ne disposons pas de données expérimentales sur la répartition des erreurs (voir paragraphe suivant). En effet, l'information qui est importante pour la vérification de programme est de savoir si l'erreur est oui ou non dans une certaine zone d'espace, ceci avec une plus ou moins grande certitude (par exemple avec une probabilité au moins égale à 0.99). Dans ce cas, on peut obtenir des résultats très comparables avec les deux types

[1] On appelle support d'une distribution l'ensemble où celle-ci est non nulle. Les lois de probabilité susceptibles d'être retenues pour représenter des incertitudes sont à support connexe.

de représentation.

Cependant, quand on compose des incertitudes, les résultats obtenus avec l'une ou l'autre des représentations deviennent qualitativement différents, ceci d'autant plus que le nombre de compositions est grand. Pour une représentation ensembliste, l'ensemble des erreurs possibles croît très vite. Pour une représentation probabiliste, l'ensemble dans lequel se trouve une erreur avec une probabilité proche de 1 donnée croît beaucoup moins vite. Cela vient du fait que cette représentation permet de prendre en compte les phénomènes de "compensation statistique des erreurs". En effet pour une incertitude composée de plusieurs incertitudes, les erreurs les plus importantes ne peuvent provenir que de la somme d'erreurs importantes et dans le même sens, et sont donc affectées d'une probabilité très faible.

Exemples

1. La figure 6.1 montre la croissance comparée pour une composition pour les deux types de représentation.

2. Dans le cas d'une représentation ensembliste, quand on fait la composition de N incertitudes d'amplitude σ, le résultat est d'amplitude $N\sigma$. Si on considère que les N incertitudes peuvent être représentées par des lois de probabilité gaussiennes telles que 99 % des erreurs seront dans un ensemble de taille σ, 99 % des erreurs composées seront dans un ensemble de taille seulement $\sqrt{N}\,\sigma$.

Il faut remarquer que la taille des ensembles croît *nécessairement moins vite* avec une représentation probabiliste qu'avec une représentation ensembliste mais aussi que pour une représentation probabiliste, la croissance *dépend fortement des lois de probabilité choisies.* En conclusion, il découle de tout ceci que si l'on choisit une représentation de type ensembliste, on adopte un point de vue résolument pessimiste du problème: un programme qui sera déclaré correct sera *sûr* de fonctionner. En même temps, on rejettera un grand nombre de programmes qui auraient cependant de bonnes chances d'être éxécutés correctement. Le choix d'une représentation probabiliste permet théoriquement de conserver tous les programmes qui ont de très grandes chances d'être éxécutés correctement. Cependant, cela passe par le choix d'une loi de probabilité adaptée aux conditions réelles de la manipulation, sinon on peut avoir suivant les cas une sous-estimation ou une surestimation des incertitudes. Le choix de la loi de probabilité est loin d'être évident.

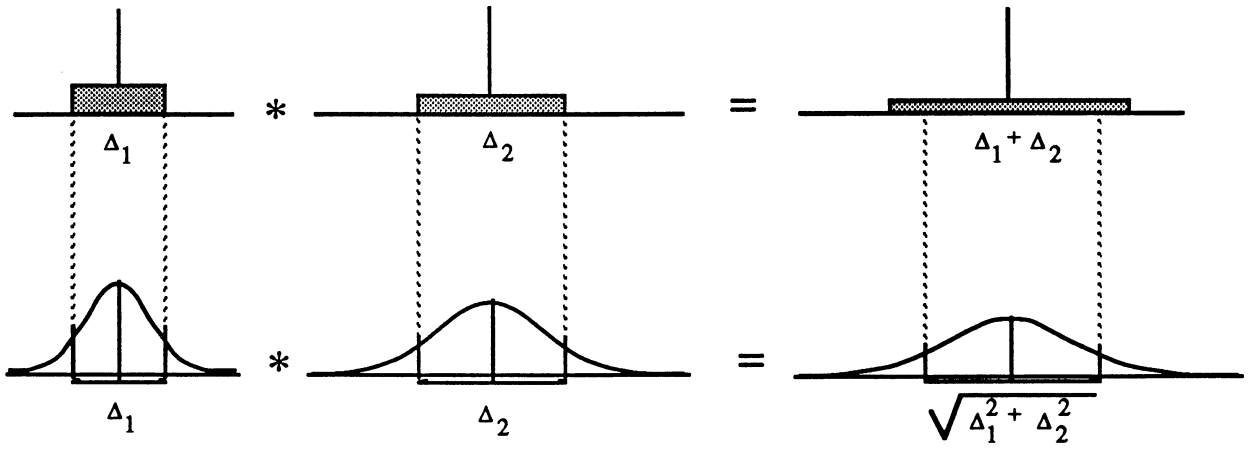


Figure 6.1 : croissance comparée des ensembles d'erreurs pour deux types de représentation

Il faut également noter que le choix de la représentation conditionne fortement la rapidité et la complexité des calculs mis en œuvre, en particulier pour la composition d'incertitudes. Pour la composition, l'opération de base pour une représentation ensembliste est la somme d'ensembles (somme de Minkowski [LW79]) et pour une représentation probabiliste, le produit de convolution de lois de probabilité. Ces deux opérations sont assez complexes dans le cas général. Cependant, si comme nous l'avons fait, on retient des lois de probabilité gaussiennes, le produit de convolution devient relativement aisé.

1.1.2 Justifications de notre choix

Le problème de l'analyse des erreurs de positionnement d'un manipulateur est un vaste sujet de recherche en soi dont l'étude nécessite en outre une mise en œuvre assez lourde de matériel de mesure spécialisé. C'est pourquoi nous n'avons pas procédé nous mêmes à des études expérimentales. De plus, bien qu'il existe une importante littérature portant sur le problème de la calibration des manipulateurs et de l'élimination des erreurs systématiques de positionnement, nous n'avons pu trouver de données expérimentales concernant la répartition statistique des erreurs. Les critères qui ont conduit aux choix que nous avons fait sont donc des critères théoriques qu'il faudrait valider par une expérimentation pratique.

En l'absence de données expérimentales nous avons tout d'abord adopté un point de vue pessimiste en retenant une représentation ensembliste. Cependant la relative simplicité des calculs que nécessite une représentation gaussienne nous a finalement amené à changer pour cette dernière. Deux raisons supplémentaires peuvent être avancées pour également justifier ce choix:

- application du théorème central limite. Si une grandeur X est la somme de n grandeurs X_i qui sont chacune entachées d'incertitude, si ces grandeurs X_i sont indépendantes (au sens probabiliste), si enfin la perturbation de chacune d'entre elles reste faible par rapport à la perturbation totale, le théorème central limite permet d'affirmer que la variable X suivra une loi de probabilité proche d'une loi gaussienne, et ceci d'autant mieux que n est grand, quelles que soient les distributions des variables X_i.

  Un robot est un système mécanique complexe et on peut cosidérer que son comportement dépend d'un grand nombre de paramètres apportant chacun une petite perturbation. En vertu de ce théorème, les erreurs de positionnement peuvent alors être supposées suivre une répartition gaussienne.

- minimum d'entropie. Etant donnés une moyenne et un écart-type, la distribution de probabilité donnant la quantité d'information minimum est la loi gaussienne définie par ces deux paramètres.

1.2 Calcul des incertitudes au sein du modèle

Nous avons vu au chapitre 4 les différents calculs mis en œuvre pour la composition des incertitudes (calcul de l'incertitude relative de deux repères quelconques). Nous avons vu en particulier que tous les calculs se faisaient à partir des trois calculs élémentaires suivants:

1. Etant données une erreur ε associée à une variable aléatoire $\vec{\varepsilon}$ de moyenne nulle et de matrice de covariance Δ et une transformation T, calculer la matrice de covariance Δ' de la variable aléatoire $\vec{\varepsilon}'$ associée à l'erreur $\varepsilon' = T^{-1} * \varepsilon * T$;
2. Etant données deux variables aléatoires $\vec{\varepsilon}_1$ et $\vec{\varepsilon}_2$ de matrices de covariances respectives Δ_1 et Δ_2, calculer la matrice de covariance Δ' de la variable aléatoire $\vec{\varepsilon}'$ associée à l'erreur $\varepsilon' = \varepsilon_1 * \varepsilon_2$;
3. Etant donnée une variable aléatoire $\vec{\varepsilon}$ de matrice de covariance Δ, associée à une erreur ε, calculer la matrice de covariance Δ' de la variable aléatoire ε' associée à l'erreur $\varepsilon' = \varepsilon^{-1}$

A cause des rotations, pour chacun de ces trois problèmes, l'erreur ε' n'est pas fonction linéaire des erreurs ε, ε_1 ou ε_2. Cependant, comme les erreurs en rotation sont supposées petites, nous avons procédé à des approximations linéaires.

1.2.1 Calcul de $\varepsilon' = T^{-1} * \varepsilon * T$

Pour ce calcul, l'approximation qui est faite ne porte que sur le vecteur de translation de ε'. Soit $\overline{\varepsilon'}$ la valeur approchée de ε', $\overline{\tau'}$ son vecteur de translation. L'erreur maximale introduite par la linéarisation, soit $\tau' - \overline{\tau'}$ a un module de l'ordre de $\frac{1}{2}\alpha^2 \parallel t \parallel$, où α est l'angle de ε et t le vecteur translation de la transformation T (voir annexe B où est développé le calcul). Pour un angle α inférieur à 0.1 rad, cette erreur dûe à la linéarisation ne représente donc que 0.5 % de la valeur de $\parallel t \parallel$.

Pour apprécier l'influence de la linéarisation non plus sur le calcul d'une seule erreur mais plus globalement sur le calcul des matrices de covariance, nous avons procédé à plusieurs séries d'essais suivant la procédure suivante:

- calcul de N (N étant de l'ordre de plusieurs milliers) erreurs ε_i réparties suivant une distribution gaussienne,
- calcul de la matrice de covariance Δ des N ε_i,
- calcul des N erreurs $\varepsilon'_i = T^{-1} * \varepsilon_i * T$,
- calcul de la matrice de covariance exacte Δ' des N ε'_i,
- calcul de la matrice de covariance approchée $\overline{\Delta'}$ en appliquant les résultats de la propriété 4.8 du chapitre 4 à la matrice Δ,
- comparaison de Δ' et $\overline{\Delta'}$

Des résultats numériques de ces essais figurent à l'annexe B. La figure **6.2** montre une sortie réalisée sur écran graphique. Comme on pouvait s'y attendre, les blocs des matrices Δ' et $\overline{\Delta'}$ concernant les rotations sont identiques (bloc 3×3 inférieur droit). Les différences les plus importantes se retrouvent sur les coefficients concernant la translation (bloc 3×3 supérieur gauche). Elles sont d'autant plus grandes que $\parallel t \parallel$ est grand. Pour une translation t de module égal à 1, un écart type de 0.1 sur toutes les coordonnées de ε, l'écart le plus important sur les coefficients de Δ' est de $14\ 10^{-5}$, soit une erreur relative de 0.7 %.

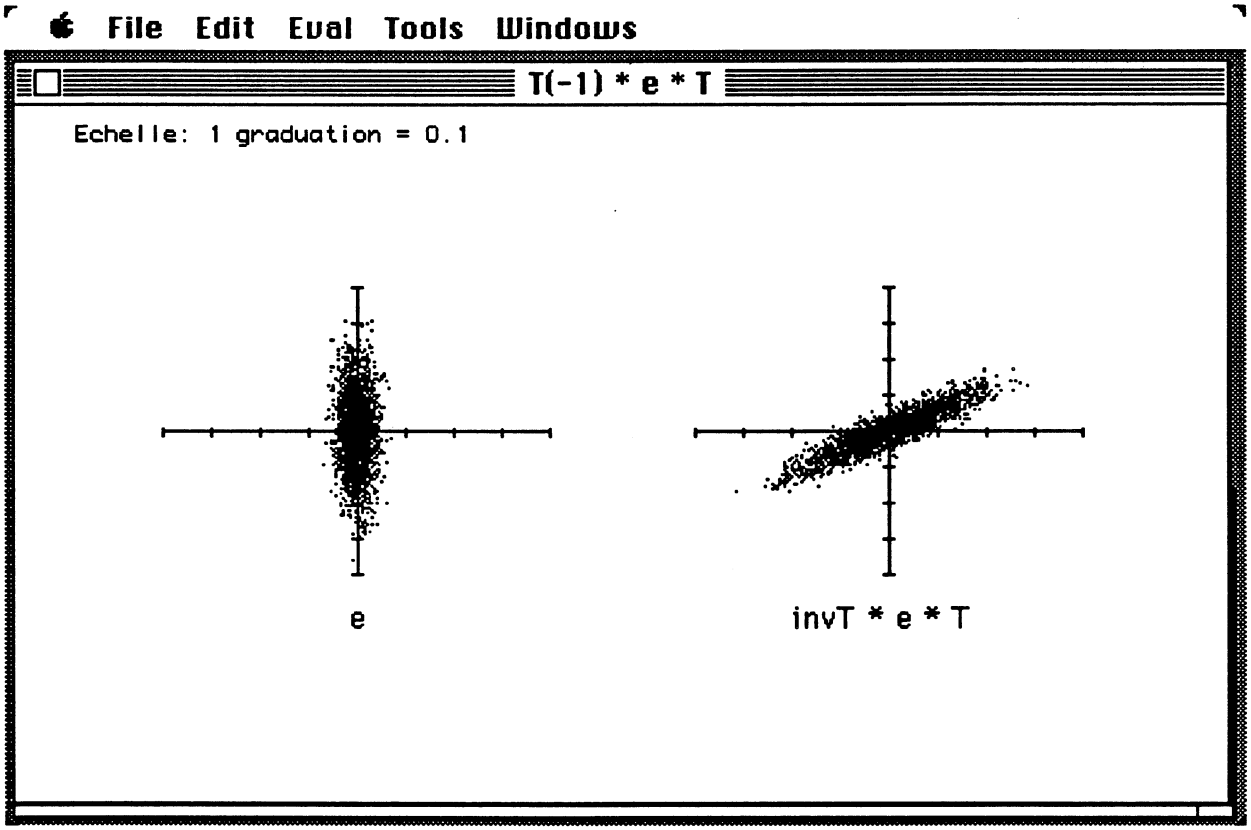


Figure 6.2 : trace d'éxécution de nos essais

1.2.2 Calcul de $\varepsilon' = \varepsilon_1 * \varepsilon_2$

Pour ce calcul, l'approximation de ε' porte à la fois sur la partie translation et la partie rotation. Si comme précédemment on note $\overline{\varepsilon'}$ la valeur approchée de ε', $\overline{\tau'}$ et $\overline{\nu'}$ respectivement les vecteurs de translation et l'axe unitaire de rotation associé, le calcul montre (voir annexe B) que l'erreur $(\overline{\nu'} - \nu)$ sur la rotation est de l'ordre de $\frac{1}{2}\alpha_1\alpha_2(\nu_1 \wedge \nu_2)$ si ν_1 et ν_2 ne sont pas colinéaires. La linéarisation est exacte si les axes ν_1 et ν_2 sont colinéaires. Si les angles de rotation α_1 et α_2 sont de l'ordre de 0.1 rad, l'erreur relative maximale sur l'axe de rotation sera de l'ordre de 5%.
En ce qui concerne le vecteur $\overline{\tau'}$, l'erreur $(\tau' - \overline{\tau'})$ est de l'ordre de $\alpha_1\nu_1 \wedge \tau_2$, soit une erreur relative maximale de l'ordre de 10% pour des angles inférieurs à 0.1 rad.

Comme précédemment, pour caculer l'influence de la linéarisation sur le calcul des matrices de covariance, nous avons procédé aux essais suivants:

- tirage de N erreurs ε_{1i} et ε_{2i} réparties suivant une distribution gaussienne,
- calcul des matrices de covariance Δ_1 et Δ_2 des ε_{1i} et ε_{2i},

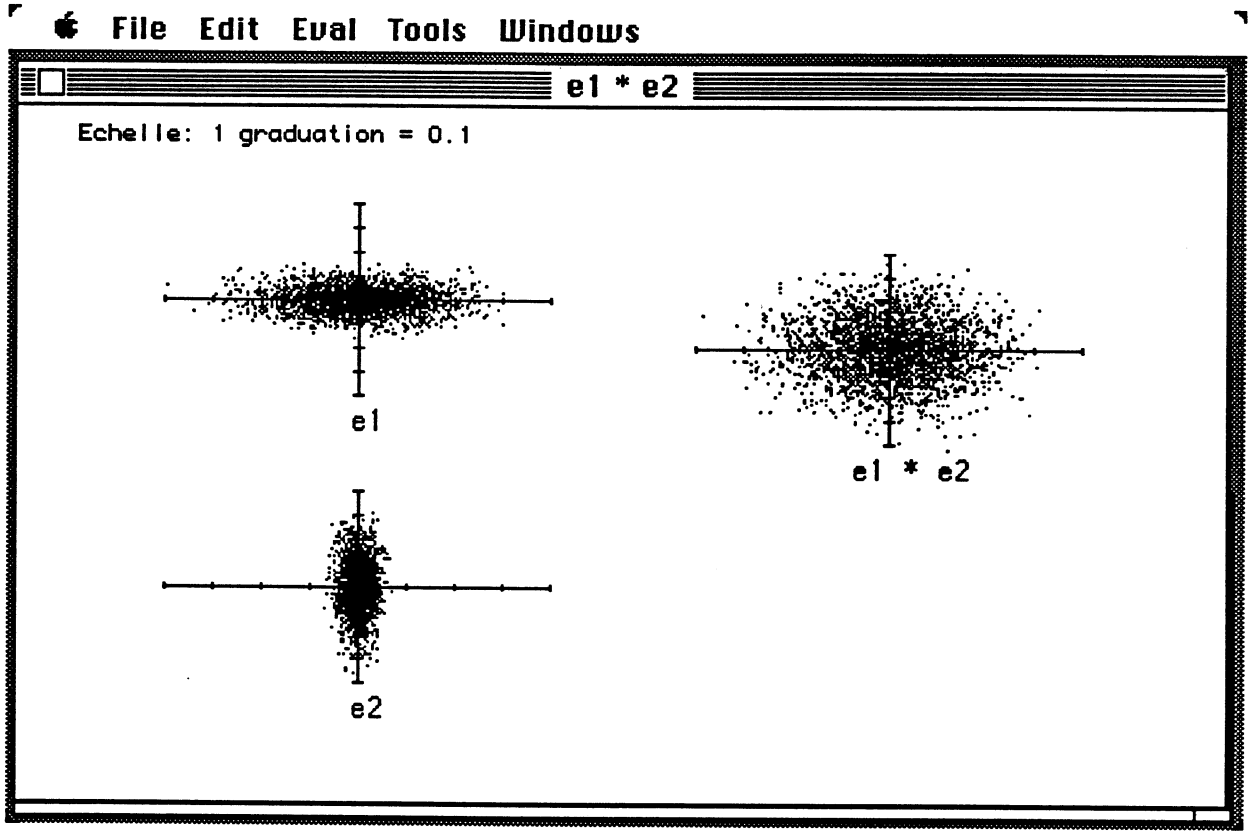


Figure 6.3 : trace d'éxécution de nos essais

- calcul des N erreurs $\varepsilon'_i = \varepsilon_{1i} * \varepsilon_{2i}$,
- calcul de la matrice de covariance exacte Δ' des N ε'_i,
- calcul de la matrice de covariance approchée $\overline{\Delta'} = \Delta_1 + \Delta_2$,
- comparaison de Δ' et $\overline{\Delta'}$

Des résultats figurent à l'annexe B et sont représentés graphiquement par la figure 6.3. Pour des écarts types inférieur à 0.1 sur les coordonnées de ε_1 et ε_2, la différence maximale sur les coefficients de Δ' et $\overline{\Delta'}$ est de 43 10^{-5} soit une erreur relative de 2%. La figure 6.4 montre une trace graphique d'éxécution d'un essai combinant les deux calculs précédents.

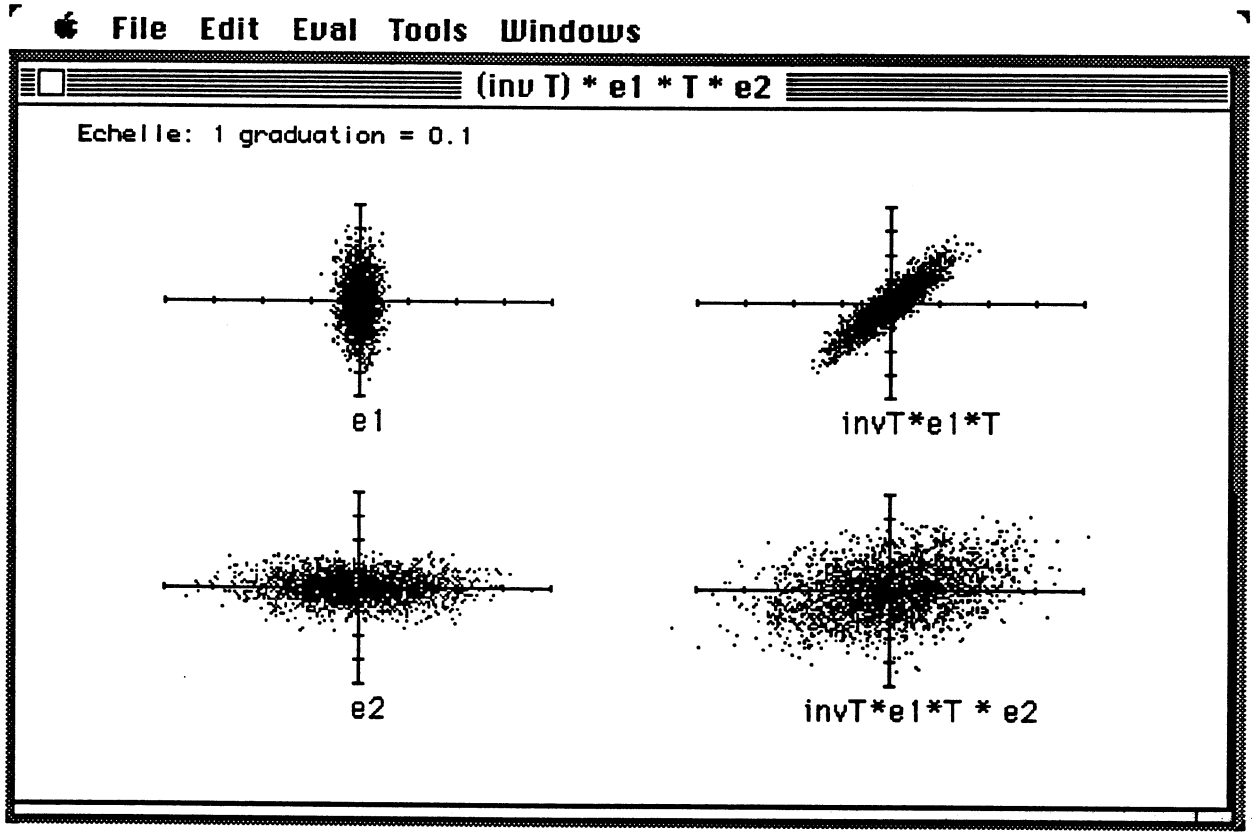


Figure 6.4 : trace d'éxécution de nos essais

1.2.3 Calcul de $\varepsilon' = \varepsilon^{-1}$

Pour ce calcul, l'approximation de ε' porte sur le vecteur τ' de translation de ε'. En reprenant les mêmes notations que précédemment, l'erreur maximale introduite par la linéarisation, soit $\tau' - \overline{\tau}'$ a un module de l'ordre de $\alpha \parallel \tau \parallel$ (voir annexe B où est développé le calcul). Pour des angles inférieurs à 0.1 rad, l'erreur relative maximale sur τ' est de l'ordre de 10%.

Nous avons également procédé aux essais suivants:

- tirage de N erreurs ε_i réparties suivant une distribution gaussienne,
- calcul de la matrice de covariance Δ des ε_i,
- calcul des N erreurs ε'_i,
- calcul de la matrice de covariace de ces N ε'_i,
- comparaison des deux matrices de covariance

Des résultats figurent à l'anexe B. Pour des écarts types inférieur à 0.1 sur les coordonnées de ε, la différence maximale sur les coefficients des deux matrices est de 3.10^{-5} ce qui est tout à fait négligeable.

1.3 Intégration du résultat d'une mesure

Nous avons vu que l'intégration d'une mesure de position relative d'un repère $\Re_j$ par rapport à un repère $\Re_i$ se faisait en reportant sur chacun des arcs du chemin $Way(\Re_i, \Re_j)$ l'information apportée par la mesure puis en "oubliant" cette mesure, c'est-à-dire en supprimant l'arc $Arc(\Re_i, \Re_j)$ créé par cette mesure. Ce mécanisme entraine en général une surestimation des incertitudes assez importante. Pour chaque arc $Arc(\Re_k, \Re_l)$ du chemin $Way(\Re_i, \Re_j)$, notons Δ_{kl} l'incertitude avant msesure, $\Delta*_{kl}$ l'incertitude après intégration du résultat de la mesure. Si l'incertitude de la mesure $\overline{\Delta}_{ij}$ est plus faible que l'incertitude calculée en faisant la composition le long du chemin $Way(\Re_i, \Re_j)$, (ce qui est en général le cas, sinon il n'est pas très intéressant de faire une mesure!), chaque incertitude $\Delta*_{kl}$ sera plus petite que l'incertitude initiale Δ_{kl} mais aussi plus grande que $\overline{\Delta}_{ij}$. Après intégration de la mesure, si on compare les $\Delta*_{kl}$ le long du chemin $Way(\Re_i, \Re_j)$, on trouvera une incertitude beaucoup plus grande que $\overline{\Delta}_{ij}$, dépendant en outre fortement de la longueur du chemin. Si on avait gardé toute l'information au sein du modèle, c'est-à-dire n'ayant pas supprimé l'arc $Arc(\Re_i, \Re_j)$ créé par la mesure, on aurait trouvé une incertitude $\overline{\Delta}*_{ij}$ plus petite que $\overline{\Delta}_{ij}$. Cela peut conduire à considérer comme incorrect un programme qui l'est pourtant. La surestimation apportée ainsi dépend fortement de deux facteurs:

1. la longueur du chemin $Way(\Re_i, \Re_j)$. Plus celle-ci est grande, plus la surestimation est importante. Si $Way(\Re_i, \Re_j)$ est réduit à un seul arc, il n'y a pas de surestimation.

2. l'ordre de grandeur des incertitudes Δ_{kl}. Si toutes ces incertitudes sont faibles devant $\overline{\Delta}_{ij}$ sauf une qui peut être très supérieure, l'intégration de la mesure aura pour effet de diminuer fortement cette incertitude qui est importante, et de la rendre comparable à $\overline{\Delta}_{ij}$. La surestimation occasionnée par "l'oubli" de la mesure sera alors relativement faible. Plus les incertitudes Δ_{kl} sont importantes, plus la surestimation sera grande. Si au moins deux des incertitudes Δ_{kl} sont supérieures ou très supérieures à $\overline{\Delta}_{ij}$, alors la surestimation sera importante.

On peut utiliser des modes de propagation plus complexes qui résolvent ce probléme de surestimation. Cependant ils n'ont pas été implantés et leur intégration dans le contexte de la vérification-correction de programme n'a pas été encore étudié.

1.4 Propagation des incertitudes à travers les actions

Un phénomène tout à fait similaire se produit quand on décrit comme nous l'avons fait, les effets des actions de prise et de dépose. Considérons par exemple la prise d'un objet *obj* dont la position est initialement définie par rapport à ref_obj (voir figure 5.3). Pendant la prise, nous avons vu que la position de *obj* par rapport à ref_obj se modifiait suivant certains degrés de liberté. Sur les autres, cette position ne varie pas (par exemple la position suivant z si l'objet repose sur un plan horizontal) donc l'incertitude ne devrait pas varier. Pourtant, si on calcule l'incertitude de position de *obj* par rapport à ref_obj, après la prise, c'est-à-dire en composant $Way(ref_obj, robot)$, $Arc(robot, gripper)$ et $Arc(gripper, obj)$, on trouvera une incertitude plus grande. Cela vient encore de la perte d'information occasionnée par la destruction de l'arc $Arc(ref_obj, obj)$. Cependant, c'est beaucoup moins grave ici: dès que l'on aura modifié la position de *obj* par l'intermédiaire du robot, il n'y aura plus de perte d'information.

2 Extension possible du système

L'approche de VCP que nous avons présentée jusque là souffre de quelques limitations "fonctionnelles", c'est à dire que l'on ne peut pas traiter jusqu'à présent tous les types de programmes de niveau effecteur. Nous allons voir dans ce paragraphe comment il serait possible d'enrichir notre approche afin de prendre en compte les incertitudes intervenant *pendant* l'éxécution des actions.

Considérer une manipulation comme une succession d'actions différentes séparées par des états de transition est tout à fait naturel et permet de commander le robot par l'intermédiaire d'un programme informatique. C'est aussi pour cette raison que nous avons considéré en première approximation que le bon déroulement d'une manipulation pouvait être vérifié en ne considérant que les états intermédiaires entre les actions. Cependant, contrairement à un programme d'informatique "classique" où les transitions entre états sont tout à fait déterministes, le succès d'un programme de commande de robot peut être fortement conditionné par ce qui se passe *pendant* l'éxécution d'une action. En conséquence, il peut s'avérer important de contrôler l'évolution des incertitudes non plus en des instants privilégiés entre les actions du robot mais aussi pendant l'éxécution de celles-ci.

Les incertitudes survenant au cours d'une action viennent pour la plupart des incertitudes de suivi de trajectoire par le robot. Elles surviennent à un instant donné mais dépendent assez peu des conditions antérieures à l'action (au mieux elles peuvent dépendre de la position du robot juste avant l'action). De ce fait, notre méthode fondée sur la propagation d'incertitudes se prête mal à la prise en compte de ces erreurs. On peut cependant étendre notre approche à la détection de telles causes d'échec en introduisant un modèle d'actions plus complexe en considérant les actions du robot comme des fonctions du temps. Les incertitudes de position du robot ne sont plus données seulement après éxécution d'un déplacement mais aussi comme fonction du temps. Un programme contraint est un programme auquel on associe, outre des contraintes devant être satisfaites dans les états intermédiaires, des contraintes devant être satisfaites à tout instant pendant l'éxécution des actions du robot. Ces dernières contraintes sont des prédicats où intervient une variable temporelle. En cas d'échec causé par la non satisfaction d'une telle contrainte, on peut corriger le programme en modifiant le mouvement du robot pendant cette action ou éventuellement - si les incertitudes dépendent d'un état antérieur - par un patch. Le moyen privilégié de modifier le mouvement du robot pour réduire les incertitudes est l'utilisation de mouvements à compliance analysés au paragraphe suivant.

Conclusion

La programmation automatique des robots, en tant que problème de planification d'actions et de mouvements d'un bras manipulateur passe par une modélisation réaliste des effets de ces actions et de l'environnement du robot. En particulier, il est nécessaire de prendre en compte les incertitudes de positionnement à cause de leur importance primordiale quant au succès d'une manipulation. De plus, du fait de la complexité du problème, due en particulier aux interdépendances entre les actions, il semble raisonnable d'effectuer la planification en deux phases:

- résoudre séparément les problèmes de saisie, transport et montage des objets en ignorant ces interdépendances et en faisant certaines hypothèses sur les incertitudes,
- vérifier la validité globale du programme ainsi engendré et en cas de non-validité, modifier localement ce programme pour réduire les incertitudes critiques et ainsi le rendre correct.

Compte tenu de ces hypothèses, nous avons développé un ensemble d'outils formels et de méthodes de calcul permettant de décrire les actions principales d'un robot en vue de faire la *la vérification-correction* d'un programme de manipulation.
Plus précisément, notre contribution s'est faite sur deux points essentiels:

- nous avons développé un modèle de l'environnement du robot incluant une représentation explicite des incertitudes de positionnement. Pour cela, nous avons repris des modèles utilisés par ailleurs. Nous les avons adaptés à notre problème particulier, procédé à des approximations et mesuré par le calcul et expérimentalement

l'impact de ces approximations. Il est à noter que notre approche n'est pas dépendante d'une représentation particulière des incertitudes. La preuve en est que nous avons pu changer de représentation sans bouleverser notre approche. Les propriétés "minimales" pour une représentation des incertitudes sont les suivantes:

  - des propriétés algébriques qui permettent de calculer l'incertitude de position relative de deux objets quelconques: composition et inversion de chemins.
  - des propriétés "topologiques": relation d'ordre sur les incertitudes, image par une projection, ou plus généralement par une fonction quelconque, et fusion d'informations redondantes.

- nous avons développé un modèle de propagation des incertitudes à travers les actions.

  - propagation "vers l'avant" dans un but prédictif, pour, partant des incertitudes initiales, calculer la suite des états possibles de l'environnement à chaque instant et vérifier que ces états satisfont des contraintes imposées par les tolérances du montage,
  - propagation "vers l'arrière" pour calculer de nouvelles contraintes sur les états antérieurs, afin d'apporter une aide à la correction.

Notre contribution est cependant partielle et le problème de la vérification-correction de programme est loin d'être clos. En particulier, il reste à résoudre les points suivants sur lesquels nous nous sommes arretés:

1. détermination automatique des actions à rajouter pour rendre le programme correct
2. validation expérimentale sur un manipulateur réel des modèles d'action et des représentations des incertitudes
3. intégration de l'approche dans un système de programmation automatique

Nous pensons que le problème de la programmation automatique off-line passe absolument par la résolution de ces trois points. Si de trop grandes difficultés venaient à apparaitre, en particulier pour le premier point, nous pensons que cela devrait remettre en cause le principe même de la programmation off-line. Cependant, même si ce but ultime ne peut être atteint, il faut souligner que les outils développés sont une base très importante pour la réalisation de logiciels d'aide à la conception de programme de manipulation pour assister un programmeur humain.

Bibliographie

[AF88] N. Ayache and O. Faugeras. *Maintaining representations of the environment of a mobile robot.* Technical Report 789, INRIA, fevrier 1988.

[Bac78] John Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of problems. *Communication of the ACM*, 21(8), Aout 1978.

[BB83] P. Berlioux and Ph. Bizard. *Construction, Preuve et évaluation de programmes.* Dunod, 1983.

[Bro82] R. A. Brooks. Symbolic error analysis and robot planning. *Int. Jour. of Robotics Research*, 1(4):29–68, winter 1982.

[Bro84] R. A. Brooks. Aspects of mobile robot visual map making. In *2nd Int. Symp. in Robotics Research*, Tokyo, Aout 1984.

[Bro85] R. A. Brooks. Aspects of mobile robot visual map making. In *IEEE Int. Conf. on Robotics and Automation*, St Louis, USA, March 1985.

[Buc87] S. T. Buckley. *Planning and teaching Compliant Motions Strategies.* PhD thesis, M.I.T. Cambridge, 1987.

[CL85] R. Chatila and J. P. Laumond. Position referencing and consistent world modeling for mobile robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 138–145, Saint Louis, mars 1985.

[Don86a] B. R. Donald. Robot motion planning with uncertainty in the geometric models of the robot and environment: a formal framework for error detection and recovery. In *IEEE Int. Conf. on Robotics and Automation*, San Francisco, USA, April 1986.

[Don86b] B. R. Donald. A theory of error detection and recovery: robot motion planning with uncertainty in the geometric models of the robot and environment. In *International workshop on geometric reasoning*, Oxford University, England, July 1986.

[Don87] B. R. Donald. *Error detection and recovery for robot motion planning with uncertainty.* PhD thesis, M.I.T. Cambridge, July 1987.

[Dur87a] H. F. Durrant-Whyte. Consistent integration and propagation of disparate sensor observations. *Int. Jour. of Robotics Research*, 6(3):3–24, fall 1987.

[Dur87b] H. F. Durrant-Whyte. Uncertain geometry in robotics. In *IEEE Int. Conf. on Robotics and Automation*, pages 851–856, Raleigh, mars 1987.

[EM86] M. Erdmann and M.T. Mason. An exploration of sensorless manipulation. In *IEEE Int. Conf. on Robotics and Automation*, pages 1569–1574, San Francisco, avril 1986.

[Erd84] M. Erdmann. *On Motion Planning with Uncertainty.* AI-TR 810, M.I.T. Cambridge, 1984.

[Erd85] M. Erdmann. Using backprojections for fine motion planning with uncertainty. In *IEEE Int. Conf. on Robotics and Automation*, Saint Louis, 1985.

[Erd86] M. Erdmann. Using backprojections for fine motion planning with uncertainty. *Int. Jour. of Robotics Research*, 5(1), spring 1986.

[Gan86] C. Gandon. *Introduction de la Compliance Dans la Programmation des Robots.* PhD thesis, Institut National Polytechnique de Grenoble, octobre 1986.

[GP81] A. Giraud and R. Prajoux. La problématique de l'assemblage automatique en robotique. In *3èmes Journées Scientifiques et Techniques de la PRODUCTION AUTOMATISEE*, ADEPA, Toulouse, 3, 4 et 5 juin 1981.

[Hoa69] C. A. R. Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 1969.

[Lat88] J. C. Latombe. *Motion Planning with Uncertainty: The Preimage Backchaining Approach.* STAN-CS 88-1196, Stanford University, march 1988.

[Lau87] C. Laugier. *Raisonnement géométrique et méthodes de décision en robotique. Application à la programmation automatique des robots.* PhD thesis, Institut National Polytechnique de Grenoble, décembre 1987.

[LB85] T. Lozano-Pérez and R. A. Brooks. *An Approach to Automatic Robot Programming.* AI memo 842, M.I.T. Cambridge, april 1985.

[Liv78] C. Livercy. *Théorie des programmes. Schémas, preuves, sémantique.* Dunod, 1978.

[LMT84] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *Int. Jour. of Robotics Research*, 3(1), spring 1984.

[Loz82] T. Lozano-Perez. *Robot Programming.* A.I. Memo 698, M.I.T. Cambridge, decembre 1982.

[LW71] P. Lucas and K. Walk. *Annual Review in Automatic Programming 6*, chapter On the formal description of PL/1, pages 105–1182. Pergamon Press, Oxford, 1971.

[LW79] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning free paths among polyhedral obstacles. *Comm. ACM*, 1979.

[Mas82] M. T. Mason. *Manipulator Grasping and Pushing Operations.* AI-TR 690, M.I.T. Cambridge, 1982.

[Mas84] M. T. Mason. Automatic planning of fine motions: correctness and completeness. In *IEEE Int. Conf. on Robotics and Automation*, Atlanta, march 13-15 1984.

[Maz87a] E. Mazer. *Handey: un modèle de planificateur pour la programmation automatique des robots.* Thèse d'Etat, Institut National Polytechnique de Grenoble, 1987.

[Maz87b] I. Mazon. *Modélisation des incertitudes de positionnement. Application à l'aide à la programmation de tâches de manipulation robotisées.* Technical Report 87 395, LAAS, Toulouse, decembre 1987.

[MB86] M.T. Mason and R.C. Brost. Automatic grasp planning: an operation space approach. In *6th Symp. on Theory and Practice of Robots and Manipulators*, Cracow (Poland), September 1986.

[MM85] E. Mazer and J.F. Miribel. *Le langage LM.* Cepadues Editions, 1985.

[Mor88] H. P. Moravec. Certainty grids for mobile robots. *AI magazine*, summer 1988.

[MP88] I. Mazon and P. Puget. Modélisation des incertitudes de positionnement. In *Journées Géométrie et Robotique*, CNRS-INRIA, LAAS, Toulouse, mai 1988.

[MS85] M.T. Mason and J.K. Salisbury. *Robot hands and the mechanics of manipulation. Artificial Intelligence*, MIT Press, 1985. Compilation of Mason's and Salisbury's Ph.D Theses.

[Nil82] N. Nilsson. *Principles of Artificial Intelligence.* Springer Verlag, 1982.

[Per86] J. Pertin-Troccaz. *Modélisation du raisonnement géométrique pour la programmation des robots.* PhD thesis, INPG, mars 1986.

[PJ82] E. Previn and J.A.Webb. *Quaternions in computer vision and robotics.* Technical Report CS-82-150, Carnegie-Mellon University, 1982.

[PP86] P.Puget and J. Pertin-Troccaz. *Contrôle dans le système de programmation automatique SHARP. Gestion des interdépendances liées aux contraintes d'accessibilité et d'incertitudes.* Technical Report RR 615 IMAG, 50 LIFIA, LIFIA, juin 1986.

[Pug85] P. Puget. *Problèmes de prise en compte des incertitudes en robotique d'assemblage.* Master's thesis, Institut National Polytechnique de Grenoble, 1985.

[Pug88] P. Puget. La vérification/correction de programme au sein d'un système de programmation automatique de robots. In *Journées Géométrie et Robotique*, CNRS-INRIA, LAAS, Toulouse, mai 1988.

[Ram88] F. Ramparany. *Un cadre pour la perception multisensorielle en robotique.* PhD thesis, Institut National Polytechnique de Grenoble, janvier 1988.

[SC86] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. Jour. of Robotics Research*, 5(4):56–68, winter 1986.

[SCS86] R. C. Smith, P. Cheeseman, and M. Self. Estimating uncertain spatial relationships. In *AAAI workshop on uncertainty in Artificial Intelligence*, Philadelphia, August 1986.

[SCS87] R. C. Smith, P. Cheeseman, and M. Self. A stochastic map for uncertain spatial relationships. In *IEEE Int. Conf. on Robotics and Automation*, Raleigh, mars 1987.

[SK87] S. Shekhar and O. Khatib. Force strategies in real time fine motion assembly. In *ASME Winter Annual Meeting*, Boston, 1987.

[Tay76] R. H. Taylor. *A synthesis of manipulator control programs from task-level specifications.* memo AIM 282, Stanford University, Computer Science Department, July 1976.

[The89] Pascal Theveneau. *Utilisation du raisonnement géométrique pour la planification en robotique d'assemblage: le système PAMELA.* PhD thesis, INPG, 1989.

[TP87] J. Troccaz and P. Puget. Dealing with uncertainties in robot planning using program proving techniques. In *4th Int. Symp. in Robotics Research*, Aout 1987.

[Vol88] R.A. Volz. Report of the robot programming language working group: nato worshop on robot programming languages. *IEEE Journal of Robotics and Automation*, 4(1):86–90, fevrier 1988.

[vWea75] A. van Wijngaarden et al. Revised report on the algorithmic language algol 68. *Acta Informatica*, 5:1–236, 1975.

[Whi87] D.E. Whitney. Historical perspective and state of the art in robot force control. *Int. Jour. of Robotics Research*, 6(1), Spring 1987.

[Zie85] C. Zieliński. Semantics of the low level robot language instructions. In *15th International Symposium On Industrial Robots*, Tokyo, Japan, septembre 1985.

Annexe A

Rappels sur la logique de Hoare

1 Introduction

La logique de Hoare fournit un ensemble d'outils formels permettant de faire la preuve de programme. On distingue:

- la preuve de correction partielle qui s'intéresse au fait que si un programme s'arrête, alors il fournit un résultat correct,
- la preuve d'arrêt qui démontre l'arrêt du programme pour toutes les données pour lesquelles il a été écrit.

Nous ne nous intéressons ici qu'à la preuve de correction partielle.
De façon générale, pour la preuve de correction partielle, on cherche à prouver un énoncé de la forme $E\{P\}S$ où E et S sont des conditions et P une séquence d'instructions. La signification de cet énoncé est "si E est vraie avant l'éxécution de P, alors si P se termine, S est vraie après éxécution de P". E est appelée précondition et S postcondition.

2 Axiomes et règles de déduction pour les preuves de correction partielle

Toute démonstration consiste à déduire un théorème $E\{P\}S$ à partir d'un ensemble d'axiomes et de théorèmes déjà établis en utilisant des règles de déduction. Les axiomes et théorèmes sont aussi des énoncés $E\{P\}S$. Nous donnons ici une liste de règles classiques premettant de prouver des énoncés à partir d'autres. Pour prouver des relations entre conditions (implications, équivalences entre conditions par exemple), on utilise les propriétés de l'algèbre de Boole et des domaines de définition des variables des programmes.

2.1 Règle de la précondition

si E{P}S
et E' $\Rightarrow$ E alors E' {P} S

2.2 Règle de la postcondition

si E{P}S
et S $\Rightarrow$ S' alors E' {P} S'

2.3 Règle du "et"

si E{P}S
et E{P}S' alors E' {P} S et S'

2.4 Règle du "ou"

si E{P}S
et E'{P}S alors E ou E' {P} S

2.5 Règle de la composition des instructions

si E {P} F
et F {Q} S alors E {P;Q} S

3 Plus faible précondition, plus forte postcondition

3.1 Plus faible précondition

Soient P une séquence d'instructions et S une postcondition. Appelons $\mathcal{E}_S$ l'ensemble des préconditions E telles que $E\{P\}S$. S'il existe (et c'est pratiquement toujours le cas) une condition E_S telle que:

$$\forall E \in \mathcal{E}_S, E \Rightarrow E_S$$

(E_S est l'élément minimal de $\mathcal{E}$ pour la relation d'ordre d'implication) E_S est alors appelée *"plus faible précondition associée à P et à S"*.
Il est très intéressant de connaitre E_S en fonction de S, auquel cas la démonstration de l'énoncé $E\{P\}S$ est équivalent à la démonstration de l'implication $E \Rightarrow E_S$ (d'après la règle de la précondition). Si E_S n'était pas l'élément minimal, il n'y aurait pas équivalence. En termes de preuve de programme de manipulation de robots, il faut remarquer aussi que E_S en tant qu'élément minimal est la précondition la plus facile à satisfaire.

3.2 Plus forte postcondtion

De même, étant donnés une suite d'instructions P et une précondition E, appelons $\mathcal{S}_E$ l'ensemble des conditions S telles que $E\{P\}S$. S'il existe un élément S_E tel que:

$$\forall S \in \mathcal{S}_E, S_E \Rightarrow S$$

S_E est alors appelé *"plus forte postcondition associée à P et à E"*. Si étant donnés P et E, on sait calculer S_E, établir l'énoncé $E\{P\}S$ est équivalent à établir l'implication $S_E \Rightarrow S$.
Pour faire la vérification d'un programme (propagation descendante des incertitudes), il serait possible de le faire par calcul de plus forte postconditions. Si l'état initial de la manipulation est caractérisé par une condition E_0, le programme est correct si et seulement si les postconditions successives de E_0 impliquent les conditions d'applicabilité C_i. Cette formulation est équivalente à celle que nous avons donnée. Cependant, la formulation de la plus forte postcondition d'une instruction est souvent mal aisée, en particulier pour l'intruction d'affectation.

4 Cas particulier de l'affectation

Nous nous bornerons ici à donner la plus faible précondition d'une instruction d'affectation $\{x := expr\}$. Pour la postcondition, nous invitons le lecteur interessé à se reporter à [Liv78] ou [BB83].
La plus faible précondition d'une instruction d'affectation $\{x := expr\}$. est égale à la condition obtenue en remplaçant dans E toutes les occurences de la variable x par $expr$ notée $E(x|expr)$.

Exemples

$$(xy \geq 0) \quad \{z := x * y\} \quad (z \geq 0)$$
$$(q+1 \geq 0) \quad \{q := q+1\} \quad (q \geq 0)$$

Annexe B

Calculs sur les erreurs et incertitudes

1 Calcul de $\varepsilon' = \varepsilon^{-1}$

1.1 Calcul de ε'

Soit $\vec{\varepsilon} = (\tau\omega)^t$ un 6-vecteur associé à une erreur ε, ρ sa rotation, τ son vecteur de translation, ν l'axe de rotation, α son angle, et ω le vecteur de rotation égal à $\alpha\nu$. Soient ω' et τ' les paramètres correspondant pour l'erreur inverse $\varepsilon' = \varepsilon^{-1}$

La rotation de ε' est ρ^{-1}. Son vecteur de rotation est l'opposé du vecteur de rotation de ρ, soit $-\omega$.
Le vecteur de translation de ε' est égal à $-\rho^{-1}\tau$. Ce vecteur n'est pas fonction linéaire de ε. Pour trouver une approximation linéaire, nous utilisons la propriété suivante:

Propriété B.1 *Soit r une rotation vectorielle d'axe unitaire v et d'angle θ. L'image par r d'un vecteur t est égale à:*

$$rt = cos\theta\, t + sin\theta\, v \wedge t + (1 - cos\theta)(v.t)v$$

Cette propriété se démontre facilement en utilisant les quaternions. En appliquant cette égalité à $-\rho^{-1}t$, on obtient:

$$\tau' = -\rho^{-1}\tau$$

$$= \; -cos\alpha\,\tau + sin\alpha\,\nu \wedge \tau - (1 - cos\alpha)(\nu.\tau)\nu$$

Soit finalement:

$$\tau' = -\tau + sin\alpha\,\nu \wedge \tau + (1 - cos\theta)[\tau - (\nu.\tau)\nu] \qquad \text{(B.21)}$$

Cette équation permet d'avoir facilement le développement limité de τ' quand α est proche de zéro. Au premier ordre, τ' est égal à $-\tau$. On prendra donc pour valeur approchée de ε':

$$\varepsilon' \simeq \begin{pmatrix} -\tau \\ -\omega \end{pmatrix}$$

1.2 Matrice de covariance associée

Si Δ est la matrice de covariance associée à l'erreur aléatoire ε, l'application de la propriété 4.4 du chapitre 4 avec M égale à $-Id$ permet d'affirmer l'égalité de Δ et Δ'.

1.3 Erreur sur ε' introduite par la linéarisation

L'erreur introduite par la linéarisation ne concerne que τ'. D'après l'équation B.21, le terme prépondérant de l'erreur est $sin\alpha \wedge \tau$. Son module maximum est de l'ordre de $\alpha \parallel \tau \parallel$. Pour des angles inférieurs à 0.1 rad, l'erreur relative sur τ' est de l'ordre de 10%.

1.4 Détermination expérimentale de l'erreur sur Δ'

La trace d'éxécution suivante est un exemple d'essai que nous avons fait pour déterminer expérimentalement l'erreur commise sur Δ', suivant la procédure décrite au paragraphe 1.2.3 du chapitre 6. Nous avons procédé ici à 10000 tirages. La matrice de covariance Δ est ici diagonale et les variances sont égales à 0.01 sur chacun des degrés de liberté (soit un écart-type de 0.1).

- le premier résultat est la moyenne des 10000 tirages de ε,
- le second est la matrice de covariance effective de ces 10000 tirages,
- le troisième est la moyenne des 10000 transformations inverses de ε,
- le quatrième est la matrice de covariance de ces inverses.

La différence la plus grande entre ces deux matrices est de 3.10^{-5}, ce qui est parfaitement négligeable.

```
NIL
> (test-inv 10000 '(0.01 0.01 0.01 0.01 0.01 0.01))

moyenne de la transformation epsilon
     0.00029      0.00052     -0.00003      0.00068      0.00050      0.00137
matrice de covariance

     0.00992      0.00003      0.00001      0.00014      0.00004      0.00018
     0.00003      0.00990     -0.00003     -0.00009     -0.00000      0.00002
     0.00001     -0.00003      0.01024      0.00010      0.00022      0.00017
     0.00014     -0.00009      0.00010      0.01010      0.00005      0.00004
     0.00004     -0.00000      0.00022      0.00005      0.01023     -0.00019
     0.00018      0.00002      0.00017      0.00004     -0.00019      0.01006

moyenne de la transformation inverse de epsilon
    -0.00008     -0.00043     -0.00007     -0.00068     -0.00050     -0.00137
matrice de covariance

     0.00991      0.00002      0.00002      0.00014      0.00001      0.00017
     0.00002      0.00992      0.00001     -0.00007     -0.00001      0.00002
     0.00002      0.00001      0.01023      0.00008      0.00023      0.00017
     0.00014     -0.00007      0.00008      0.01010      0.00005      0.00004
     0.00001     -0.00001      0.00023      0.00005      0.01023     -0.00019
     0.00017      0.00002      0.00017      0.00004     -0.00019      0.01006
NIL
>
```

2 Calcul de $\varepsilon' = T^{-1} * \varepsilon * T$

2.1 Calcul de ε'

Soient $\vec{\varepsilon} = (\tau\omega)^t$ un 6-vecteur associé à une erreur ε, T une transformation de rotation r et de vecteur de translation t. Nous allons calculer les paramètres τ' et ω' associé à la transformation $\varepsilon' = T^{-1} * \varepsilon * T$.

Le vecteur τ' est égal à:

$$\tau' = r^{-1}\left(\rho t - t + \tau\right)$$

Or d'après la propriété B.1, ρt est égal à:

$$\rho t = cos\alpha\, t + sin\alpha(\nu \wedge t) + (1 - cos\alpha)(\nu.t)\nu$$

D'où:

$$\rho t - t = \omega \wedge t + (1 - cos\alpha)[(\nu.t)\nu - t] - (\alpha - sin\alpha)\nu \wedge t$$

Le premier terme est du premier ordre en α, les seconds et troisièmes termes sont respectivement du second et troisième ordre. On en tire:

$$\tau' = r^{-1}(\omega \wedge t + \tau) + r^{-1}\delta \tag{B.22}$$

Avec δ, vecteur du second ordre en α:

$$\delta = (1 - cos\alpha)[(\nu.t)\nu - t] - (\alpha - sin\alpha)\nu \wedge t$$

La valeur approchée de τ' est $r^{-1}(\omega \wedge t + \tau)$. Si $\mathcal{M}_{\sqcup}$ est la matrice:

$$\mathcal{M}_t = \begin{pmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{pmatrix}$$

τ' est égal au premier ordre à:

$$\tau' \simeq [r^{-1}\mathcal{M}_{\sqcup}]\sqcup + \nabla^{-\infty}\tau$$

La rotation associée à ε' est $r^{-1}\rho r$ où ρ est la rotation associée à ε. $r^{-1}\rho r$ a pour vecteur de rotation l'image par r^{-1} du vecteur de rotation de ρ, soit $r^{-1}\omega$. D'où finalement:

$$\begin{pmatrix} \tau' \\ \omega' \end{pmatrix} \simeq \begin{pmatrix} r^{-1} & r^{-1}\mathcal{M}_{\sqcup} \\ 0 & r^{-1} \end{pmatrix} \begin{pmatrix} \tau \\ \omega \end{pmatrix}$$

2.2 Matrice de covariance associée

Si M est la matrice:

$$M = \begin{pmatrix} r^{-1} & r^{-1}\mathcal{M}_t \\ 0 & r^{-1} \end{pmatrix}$$

l'application de la propriété 4.4 donne:

$$\Delta' = M\Delta M^t \tag{B.23}$$

2.3 Erreur sur ε' introduite par la linéarisation

L'erreur introduite par la linéarisation ne concerne que τ'. D'après l'equation B.22, cette erreur a pour norme la norme de δ. δ est combinaison linéaire des deux vecteurs orthogonaux $[(\nu.t)\nu - t]$ et $(\nu \wedge t)$. Le premier est égal à la projection orthogonale de t sur ν moins le vecteur t, donc son module est maximal quand t et τ sont orthogonaux. Il en est de même pour $(\nu \wedge t)$.
En conclusion, δ est de module maximum quand t et ν sont orthogonaux et nul quand ils sont colinéaires. Son module maximum est en outre égal à:

$$\| \delta \|_{max} = \sqrt{(1 - cos\alpha)^2 + (\alpha - sin\alpha)^2} \| t \|$$

D'où $\| \delta \|_{max}$ est de l'ordre de $\frac{\alpha^2}{2} \| t \|$, soit une erreur relative de 5% pour des angles inférieurs à 0.1 rad.

2.4 Détermination expérimentale de l'erreur sur Δ'

La trace d'éxécution suivante est un exemple d'essai que nous avons fait pour déterminer expérimentalement l'erreur commise sur Δ', suivant la procédure décrite au paragraphe 1.2.1 du chapitre 6. Nous avons procédé ici à 3000 tirages. La transformation T est définie par un vecteur de translation de égal à x, et une rotation d'axe z et d'angle $\frac{\pi}{3}$. La matrice Δ est diagonale et toutes les variances sur chacun des degrés de liberté ont été choisies egaux à 0.01 (soit un écart-type de 0.1).

- le premier résultat est la moyenne des 3000 tirages de ε,
- le second est la matrice de covariance Δ_e effective de ces 3000 tirages,
- le troisième est la moyenne des transformations $\varepsilon' = T^{-1} * \varepsilon * T$,
- le quatrième est est la matrice de covariance effective de ces transormations,
- le dernier résultat est la matrice de covariance approchée calculée en appliquant la relation B.23 à Δ_e.

La différence la plus grande entre ces deux matrices se trouve au coefficient situé à l'intersection de la troisième ligne et de la troisième colonne. Elle est de 14.10^{-5} soit une erreur relative de 7.10^{-3}

```
()
NIL
```

```
> (test-t 3000
  '(1.0 0.0 0.0)
          '(0.0 0.0 1.0)
          (/ pi 3)
          '(0.01 0.01 0.01 0.01 0.01 0.01))

moyenne de la premiere transformation
     0.00245     0.00092    -0.00174     0.00046     0.00128    -0.00104
matrice de covariance

     0.01030    -0.00008     0.00002     0.00033    -0.00002     0.00009
    -0.00008     0.01016    -0.00001    -0.00022     0.00013    -0.00013
     0.00002    -0.00001     0.01018     0.00003    -0.00004     0.00002
     0.00033    -0.00022     0.00003     0.01014     0.00019     0.00018
    -0.00002     0.00013    -0.00004     0.00019     0.01007     0.00001
     0.00009    -0.00013     0.00002     0.00018     0.00001     0.01035

moyenne de la transformation resultat
    -0.00388     0.00668    -0.00293     0.00134     0.00025    -0.00104
matrice de covariance

     0.01766     0.00422    -0.00005     0.00014    -0.00006     0.00881
     0.00422     0.01290    -0.00006    -0.00007     0.00030     0.00499
    -0.00005    -0.00006     0.02018    -0.00877    -0.00486     0.00004
     0.00014    -0.00007    -0.00877     0.01025    -0.00013     0.00010
    -0.00006     0.00030    -0.00486    -0.00013     0.00996    -0.00015
     0.00881     0.00499     0.00004     0.00010    -0.00015     0.01035

matrice de covariance approchee

     0.01777     0.00430    -0.00008     0.00015    -0.00006     0.00890
     0.00430     0.01278    -0.00009    -0.00008     0.00031     0.00503
    -0.00008    -0.00009     0.02032    -0.00883    -0.00491     0.00001
     0.00015    -0.00008    -0.00883     0.01025    -0.00013     0.00010
    -0.00006     0.00031    -0.00491    -0.00013     0.00996    -0.00015
     0.00890     0.00503     0.00001     0.00010    -0.00015     0.01035
resultat
```

```
NIL
>
>
```

3 Calcul de $\varepsilon' = \varepsilon_1 * \varepsilon_2$

3.1 Calcul de ε'

Soient $\vec{\varepsilon_1} = (\tau_1\omega_1)^t$ et $\vec{\varepsilon_2} = (\tau_2\omega_2)^t$ deux 6-vecteurs associés à deux erreurs ε_1 et ε_2. Soit $\vec{\varepsilon'} = (\tau'\omega')^t$ le 6-vecteur associé à l'erreur $\varepsilon' = \varepsilon_1 * \varepsilon_2$.

Calcul de ω'

Pour trouver un développement limité de ω' à l'ordre 3, nous avons utilisé des quaternions. Si Q_1, Q_2 et Q sont les quaternions représentant les rotations respectives de ε_1, ε_2 et ε', en reprenant nos notations habituelles, nous avons:

$$Q_1 = cos\frac{\alpha_1}{2} + sin\frac{\alpha_1}{2}\nu_1$$

$$Q_2 = cos\frac{\alpha_2}{2} + sin\frac{\alpha_2}{2}\nu_2$$

D'où

$$\begin{aligned} Q &= Q_1Q_2 \\ &= (cos\frac{\alpha_1}{2}cos\frac{\alpha_2}{2} - sin\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}(\nu_1.\nu_2)) \\ &\quad + (cos\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}\nu_2 + cos\frac{\alpha_2}{2}sin\frac{\alpha_1}{2}\nu_1 + sin\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}(\nu_1 \wedge \nu_2)) \end{aligned}$$

On en tire:

$$cos\frac{\alpha'}{2} = (cos\frac{\alpha_1}{2}cos\frac{\alpha_2}{2} - sin\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}(\nu_1.\nu_2)) \qquad \text{(B.24)}$$

$$sin\frac{\alpha'}{2}\nu' = (cos\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}\nu_2 + cos\frac{\alpha_2}{2}sin\frac{\alpha_1}{2}\nu_1 + sin\frac{\alpha_1}{2}sin\frac{\alpha_2}{2}(\nu_1 \wedge \nu_2)) \qquad \text{(B.25)}$$

Ces égalités sont la base de notre développement limité de ω'. ω' s'écrit:

$$\begin{aligned} \omega' &= \alpha'\nu' \\ &= \frac{\alpha'}{\sqrt{1 - cos^2\frac{\alpha'}{2}}} sin\frac{\alpha'}{2}\nu' \\ &= 2\frac{arcos(cos\frac{\alpha'}{2})}{\sqrt{1 - cos^2\frac{\alpha'}{2}}} sin\frac{\alpha'}{2}\nu' \end{aligned}$$

Si on considère la fonction

$$f \longmapsto \frac{arcosx}{\sqrt{1-x^2}}$$

f admet un développement limité à l'ordre 1 au voisinage de 1:

$$f(x) = 1 - \frac{x-1}{3} + O((x-1)^2)$$

De l'égalité B.24, on déduit un développement limité à l'ordre 3 de $cos\frac{\alpha'}{2}$:

$$cos\frac{\alpha'}{2} = 1 - \frac{1}{8}(\alpha_1^2 + \alpha_2^2 + 2\alpha_1\alpha_2(\nu_1.\nu_2)) + O(\alpha^4)$$

On en tire le développement limité de $f(cos\frac{\alpha'}{2})$ à l'ordre 3:

$$f(cos\frac{\alpha'}{2}) = 1 + \frac{1}{24}(\alpha_1^2 + \alpha_2^2 + 2\alpha_1\alpha_2(\nu_1.\nu_2)) + O(\alpha^4)$$

De l'égalité B.25, on déduit un développement limité à l'ordre 3 de $sin\frac{\alpha'}{2}\nu'$:

$$sin\frac{\alpha'}{2}\nu' = \frac{\alpha_2}{2}\nu_2 + \frac{\alpha_1}{2}\nu_1 + \frac{\alpha_1\alpha_2}{4}(\nu_1 \wedge \nu_2) - \frac{\alpha_1^2\alpha_2}{16}\nu_2 - \frac{\alpha_2^2\alpha_1}{16}\nu_1 + O(\alpha^4)$$

D'où finalement le développement limité de ω' à l'ordre 3:

$$\begin{aligned}\omega' &= 2\,f(cos\frac{\alpha'}{2})sin\frac{\alpha'}{2} \\ &= \alpha_1\nu_1 + \alpha_2\nu_2 + \frac{\alpha_1\alpha_2}{2}(\nu_1 \wedge \nu_2) \\ &\quad + \frac{1}{24}(-2\alpha_1^2\alpha_2 + \alpha_2^3 + 2\alpha_1\alpha_2^2(\nu_1.\nu_2))\nu_2 + \frac{1}{24}(-2\alpha_2^2\alpha_1 + \alpha_1^3 + 2\alpha_1^2\alpha_2(\nu_1.\nu_2))\nu_1 + O(\alpha^4)\end{aligned}$$

En conclusion, la vleur approchée au premier ordre de ω' est égale à $(\omega_1 + \omega_2)$.

Calcul de τ'

τ' est égal à $(\rho_1\tau_2 + \tau_1)$. En appliquant la propriété B.1, on obtient comme précédemment:

$$\tau' = (\tau_1 + \tau_2) + (\omega_1 \wedge \tau_2) + (1 - cos\alpha_1)[(\nu_1.\tau_2)\nu_1 - \tau_2]$$

La valeur approchée de τ' est donc $(\tau_1 + \tau_2)$.

3.2 Matrice de covariance associée

La valeur approchée de $\vec{\varepsilon'}$ est la somme de $\vec{\varepsilon_1}$ et $\vec{\varepsilon_2}$. La matrice de covariance associée est la somme de Δ_1 et Δ_2.

3.3 Erreur sur ε' introduite par la linéarisation

Si ν_1 et ν_2 ne sont pas colinéaires, l'erreur sur ω' est donc de l'ordre de $\frac{\alpha_1\alpha_2}{2}(\nu_1 \wedge \nu_2)$. Si ces vecteurs sont colinéaires, la linéarisation est exacte. L'erreur est maximale quand ν_1 et ν_2 sont orthogonaux et est de l'ordre de 5.10^{-3} pour des angles inférieurs à 0.1 rad, soit une erreur relative de de l'ordre de 5%.
Concernant τ', l'erreur introduite est de l'ordre de $\omega_1 \wedge \tau_2$. L'erreur maximale est de module de l'ordre de 10^{-2} pour des angles inférieurs à 0.1 rad et des translations inférieures à 0.1, soit une erreur relative de 10%.

3.4 Détermination expérimentale de l'erreur sur Δ'

La trace d'éxécution suivante est un exemple d'essai que nous avons fait pour déterminer expérimentalement l'erreur commise sur Δ', suivant la procédure décrite au paragraphe 1.2.2 du chapitre 6. Nous avons procédé ici à 10000 tirages. Les matrices de covariance Δ_1 et Δ_2 sont ici diagonales et les variances sont égales à 0.01 sur chacun des degrés de liberté (soit un écart-type de 0.1).

- le premier résultat est la moyenne des 10000 tirages de ε_1,
- le second est la matrice de covariance Δ_1 effective de ces 10000 tirages,
- le troisième est la moyenne des 10000 tirages de ε_2,
- le quatrième est est la matrice de covariance Δ_2 effective de ces 10000 tirages,
- le cinqième est la moyenne des 10000 transforamations $\varepsilon' = \varepsilon_1 * \varepsilon_2$,
- le sixième est la matrice de covariance de ces 10000 transformations,
- le dernier résultat est la matrice de covariance approchée calculée en faisant la somme des matrices Δ_1 et Δ_2 précédentes.

La différence la plus grande entre ces deux matrices se trouve au coefficient situé à l'intersection de la cinqième ligne et de la cinquième colonne. Elle est de 43.10^{-5} soit une erreur relative de 2.10^{-2}.

```
> (test-* 10000
  '(0.01 0.01 0.01 0.01 0.01 0.01)
  '(0.01 0.01 0.01 0.01 0.01 0.01))

moyenne de la premiere transformation
```

```
    0.00134    -0.00033    -0.00001     0.00159    -0.00139    -0.00054
matrice de covariance

    0.01001    -0.00008    -0.00021    -0.00007    -0.00006    -0.00015
   -0.00008     0.01026     0.00005    -0.00009     0.00010    -0.00007
   -0.00021     0.00005     0.00978    -0.00016    -0.00019    -0.00005
   -0.00007    -0.00009    -0.00016     0.01006     0.00012     0.00006
   -0.00006     0.00010    -0.00019     0.00012     0.00987    -0.00007
   -0.00015    -0.00007    -0.00005     0.00006    -0.00007     0.01004

moyenne de la seconde transformation
   -0.00080    -0.00025    -0.00038    -0.00062    -0.00001    -0.00098
matrice de covariance

    0.00979     0.00002     0.00011    -0.00011    -0.00010     0.00000
    0.00002     0.00998    -0.00005     0.00003     0.00007    -0.00001
    0.00011    -0.00005     0.01012    -0.00008    -0.00008    -0.00006
   -0.00011     0.00003    -0.00008     0.01015    -0.00001     0.00005
   -0.00010     0.00007    -0.00008    -0.00001     0.01021    -0.00004
    0.00000    -0.00001    -0.00006     0.00005    -0.00004     0.00989

moyenne de la transformation resultat
    0.00040    -0.00060    -0.00017     0.00087    -0.00133    -0.00160
matrice de covariance

    0.01989    -0.00001     0.00008    -0.00031    -0.00035    -0.00035
   -0.00001     0.02015    -0.00024     0.00004     0.00015    -0.00007
    0.00008    -0.00024     0.02001    -0.00042    -0.00034    -0.00004
   -0.00031     0.00004    -0.00042     0.02040     0.00018     0.00004
   -0.00035     0.00015    -0.00034     0.00018     0.02051    -0.00018
   -0.00035    -0.00007    -0.00004     0.00004    -0.00018     0.02004

matrice de covariance approchee

    0.01980    -0.00006    -0.00010    -0.00019    -0.00016    -0.00015
   -0.00006     0.02024    -0.00001    -0.00007     0.00017    -0.00008
   -0.00010    -0.00001     0.01990    -0.00024    -0.00027    -0.00012
```

```
    -0.00019    -0.00007    -0.00024     0.02021     0.00012     0.00011
    -0.00016     0.00017    -0.00027     0.00012     0.02008    -0.00011
    -0.00015    -0.00008    -0.00012     0.00011    -0.00011     0.01993
NIL
>
```

Annexe C

Représentation ensembliste des incertitudes

1 Principe de représentation des incertitudes

Dans cette représentation particulière, une incertitude est définie comme *l'ensemble* des erreurs ε susceptibles de se produire. Une erreur est comme au chapitre 4 la transformation entre la position théorique d'un repère et sa position réelle.

Une transformation géométrique peut-être définie par la donnée d'un vecteur de R^3 (vecteur de translation), d'un vecteur de la sphère unité $S(1)$ (axe de rotation) et d'un angle de rotation. L'ensemble des transformations est donc isomorphe à l'ensemble produit cartésien $R^3 \times S(1) \times [-\pi, +\pi]$. On représentera une incertitude par un sous-ensemble de $R^3 \times S(1) \times [-\pi, +\pi]$. Cet espace produit n'ayant pas de propriétés mathématiques "sympathiques" (en particulier, on ne peut pas le pourvoir d'une structure d'espace vectoriel euclidien), il est très difficile de caractériser un ensemble de forme quelconque. On est alors amené à approximer un sous-ensemble de cet espace par un produit de sous-ensembles $T \times U \times A$, où T est un sous-ensemble particulier de R^3 particulier, U un sous-ensemble particulier de $S(1)$ et A un sous-ensemble de $[-\pi, +\pi]$.

Etant donné un sous-ensemble E de $R^3 \times S(1) \times [-\pi, +\pi]$, trouver l'approximation de E revient à:

1. projeter E sur sur chacun des espaces R^3, $S(1)$ et $[-\pi, +\pi]$, ce qui donne trois sous-ensemble Tr', U' et A' respectivement de R^3, $S(1)$ et $[-\pi, +\pi]$.

2. approximer Tr', U' et A', par des ensembles Tr, U et A de forme particulière.

L'approximation de E sera le produit $Tr \times U \times A$. Il est important de remarquer que lors de chaque approximation, on choisit un ensemble *incluant* l'ensemble réel, ce qui revient à grossir les incertitudes. Ceci est compatible avec notre définition de l'incertitude: toute erreur de position pouvant survenir se trouvera encore dans l'ensemble d'erreurs approximé.
Dans la pratique, on ne détermine jamais précisément un sous-ensemble de $R^3 \times S(1) \times [-\pi, +\pi]$, puis on calcule son approximation. D'une façon naturelle, on décompose l'incertitude de position d'un objet en un terme de translation et un terme de rotation, ce qui conduit directement aux ensembles Tr, U et A.

1.1 Exemples

Solide sur un plan $z = cste$:(cf figure 4.3)

Considérons le cube reposant sur un plan "horizontal" $z = cste$ de la figure 4.3. L'incertitude en translation pourra être représentée par un disque d'axe z et de rayon ε. En ce qui concerne l'orientation, les seules rotations permises pour le cube sont celles d'axe z. On pourra donc prendre pour ensemble U l'ensemble réduit à $\{z\}$. Pour A, on prendra un intervalle $[-\alpha, +\alpha]$.

Solide dans l'espace: (cf figure 4.4)

Si on considère le solide libre dans l'espace de la figure 4.4, l'incertitude de position en translation n' a plus de direction privilégiée. On la représentera (par l'intermédiaire de Tr) par un ensemble tridimensionnel, une sphère par exemple. Ce sera la même chose pour les rotations: l'ensemble U sera la sphère unité. A sera encore un intervalle $[-\alpha, +\alpha]$.

2 Opérations sur les incertitudes

Les trois opérations élémentaires permettant le calcul de position et d'incertitude relative de deux repères quelconques se traitent dans cette représentation de la façon exposée ci-dessous.

2.1 Calcul de $\varepsilon' = \varepsilon^{-1}$

Soit un ensemble d'erreurs ε défini par le produit $Tr \times U \times A$. Au premier ordre, le vecteur de rotation de ε^{-1} est égal à l'opposé de celui de ε. Le vecteur de translation de ε^{-1} est aussi l'opposé de celui de ε (voir appendice B). Il en résulte que l'ensemble Tr' est égal au symétrique de Tr par rapport à l'origine, U' est également le symétrique de U par rapport à l'origine. A' est égal à A.

2.2 Calcul de $\varepsilon' = T^{-1} * \varepsilon * T$

L'axe de rotation de ε' est égal à l'image par r^{-1}, inverse de la rotation associée à T, du vecteur de rotation de ε. U' est égal à l'image de U par r^{-1}. A' est égal à A.
Le vecteur de translation τ' de ε' est au premier ordre égal à:

$$\tau' = r^{-1}(\omega \wedge t + \tau)$$

(avec les notations de l'appendice B)
L'ensemble Tr' associé à ε' (ensemble des vecteurs τ') est obtenu en effectuant successivement les opérations suivantes:

- produit vectoriel de l'ensemble U par le vecteur t puis homothétie de rapport α (borne de l'ensemble A) pour obtenir l'ensemble des vecteurs $(\omega \wedge t)$.
- somme de Minkowski de l'ensemble obtenu avec l'ensemble Tr.
- image de l'ensemble obtenu par la rotation r^{-1}.

2.3 Calcul de $\varepsilon' = \varepsilon_1 * \varepsilon_2$

Le vecteur de rotation de ε' est au premier ordre égal à la somme du vecteur de rotation de ε_1 et ε_2. L'axe de rotation est susceptible de se trouver dans le secteur circulaire limité par les axes de ε_1 et ε_2. L'angle de ε est inférieur à la somme des angles extrêmes de ε_1 et ε_2.

Le vecteur de translation de ε est au premier ordre la somme des vecteurs de translation de ε_1 et ε_2. Tr' est donc égal à la somme de Minkowski des ensembles Tr_1 et Tr_2.

3 Forme particulière des ensembles Tr, U et A

En résumé, les opérations de composition et d'inversion d'incertitudes se traduisent par des opérations sur les ensembles Tr et U.

En particulier, pour l'ensemble Tr, ces opérations sont:

1. image par une rotation connue
2. grossissement (somme de Minkowski de deux ensembles)

Pour l'ensemble U, il faut calculer l'ensemble des produits vectoriels avec un vecteur donné.

La complexité de ces calculs est très mal maitrisée pour des ensembles de quelconques de dimension 3. C'est pourquoi, dans la pratique, on a fait le choix d'approximations pour des ensembles de forme simple, décrits par un nombre réduit de paramètres et avec lesquels on puisse faire les opérations ci-dessus, sinon de façon exacte, au moins avec une bonne approximation. Le choix de ces ensembles particuliers est dicté aussi par le type de relations d'incertitude rencontrées dans la pratique où les contacts entre objets, en particulier les contacts de type plan sur plan ont une influence prépondérante sur la forme des ensembles Tr et U (cas du premier exemple).

Nous avons donc choisi pour l'ensemble Tr les approximations suivantes:

- segment de droite (pour les incertitudes en translation monodimensionnelles)
- disque (pour les incertitudes en translation bidimensionnelles)
- sphère et cylindre (pour les incertitudes en translation tridimensionnelles)

Pour l'ensemble U, nous considérons soit la sphère unité $S(1)$ complète, soit un ensemble réduit à un vecteur unitaire unique.

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs

LIEGOIS Alain, Professeur

JORRAND Philippe, Directeur de recherche au CNRS

Monsieur PUGET Pierre

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "INFORMATIQUE"

Fait à Grenoble, le 13 Février 1989

Georges LESPINARD
Président
de l'Institut National Polytechnique
de Grenoble

Résumé

Cette thèse présente une approche appelée verification-correction de programme: elle permet de s'assurer de la validité globale et éventuellement de corriger un programme de manipulation de robot. Ce programme, produit par un système de programmation automatique, est obtenu en planifiant séparément les opérations de saisie, transport et montage des objets et en ignorant les incertitudes géométriques qui introduisent des interdépendances entre les actions. Un tel programme est correct si, entre chaque action du robot, les incertitudes sont inférieures à un certain seuil imposé par la tâche. La première phase dite de vérification consiste à propager "vers l'avant" les incertitudes initiales de la manipulation et à s'assurer qu'en tout point les contraintes sont satisfaites. En cas de non satisfaction d'une contrainte, celle-ci est propagée "vers l'arrière" pour calculer de nouvelles contraintes minimales sur les états antérieurs. Ces contraintes sont utilisées pour corriger le programme en ajoutant de nouvelles instructions qui réduisent les incertitudes. L'approche de vérification-correction de programme s'appuie sur un modèle de l'environnement incluant une représentation des incertitudes. Elle s'appuie aussi sur une description adaptée des actions du robot. Les techniques de propagation avant et arrière sont inspirées de méthodes utilisées en informatique fondamentale (sémantique opérationnelle et logique de Hoare) et adaptées au problème spécifique de la programmation des robots.

Mots clés

Robotique, Programmation automatique des robots, Incertitudes, Preuve de programme.