NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-53291-2

PLANIFICATION ET OPTIMISATION
DE TRAJECTOIRE POUR UN MANIPULATEUR

MARIO TETREAULT

Thèse déposée à
l'Ecole des études supérieures et de la recherche
en vue de l'obtention de la maîtrise ès sciences appliquées
en génie mécanique

UNIVERSITE D'OTTAWA

REMERCIEMENTS

Avant de débuter la présentation de cette thèse de maîtrise, je veux remercier les personnes qui m'ont supporté tout au long de mon travail. Plus particulièrement, je veux remercier M. Atef Fahim pour l'équipement fourni et les discussions relatives à mon sujet; même si elles étaient souvent très animées et orageuses. De même, je veux souligner la contribution de M. Dan S. Necsulescu pour initier le projet et son encouragement dans des moments plus difficiles. Je veux aussi remercier M. Redekop pour les connaissances transmises sur la robotique; ce qui m'a donné une connaissance globale du sujet. Je dois aussi mentionner le support apporté par M. Mohammed Mahjoub. Je m'en voudrais d'oublier Mme Johanne Sévigny, ma femme, pour son attention, ses encouragements, sa patience et sa participation directe dans l'élaboration de mon algorithme.

En terminant, je veux remercier le Conseil national de recherche en science et génie (CNRSG) et l'université d'Ottawa pour l'aide monétaire apportée.

SOMMAIRE

L'apparition de tâches industrielles de plus en plus répétitives et la nécessité d'augmenter la productivité provoquent le besoin d'automatiser en utilisant, par exemple, la robotique. La planification de trajectoire quasi-optimale pour les robots permet d'accroître la performance de ces tâches.

En regardant l'état des connaissances de la planification de trajectoires, on a remarqué deux catégories d'algorithmes: l'approche cinématique et l'approche dynamique. L'utilisation de la dynamique permet d'obtenir de meilleures performances; mais cela complique la planification de la trajectoire. En comparant les algorithmes avec les caractéristiques idéales d'une trajectoire, on a fait ressortir leurs qualités et leurs faiblesses.

De cet état de connaissance, on a remarqué le besoin de combler simultanément les trois lacunes suivantes: le manque de contrôle sur la forme géométrique de la courbe cartésienne, le manque d'optimisation temporelle et le besoin d'utiliser l'algorithme pour un robot avec plus de trois degrés de liberté. Aucun des algorithmes existants ne satisfait simultanément ces trois contraintes puisque la tendance actuelle est de développer des algorithmes pour résoudre un problème spécifique (satisfaire un nombre réduit de caractéristiques idéales) étant donné les difficultés reliées à la planification de trajectoire. Cette thèse présente un algorithme qui satisfait l'ensemble des caractéristiques idéales d'une trajectoire et qui s'applique à la majorité des tâches industrielles rencontrées.

En dépit de sa proche relation avec la cinématique, la dynamique fait généralement partie du problème d'asservissement. Pour permettre l'optimisation, on a utilisé la dynamique dans la planification de la trajectoire; ce qui élimine le besoin de modifier la trajectoire lors de l'asservissement. Cette incorporation de la dynamique s'est faite en utilisant une propriété des mécanismes: l'échelonnement dynamique du temps.

L'optimisation a été faite en utilisant une méthode globale de solution puisque le calcul d'une trajectoire optimale n'est pas possible en temps réel. Cette méthode contient une étape en calcul hors-ligne et une étape en temps réel de calcul. Plusieurs caractéristiques doivent être satisfaites, et certaines d'entre-elles affectent les autres. L'optimisation est donc un compromis entre ces différentes contraintes.

Les résultats ont montré une amélioration sur la forme géométrique de la courbe cartésienne et une réduction du temps par rapport à l'algorithme de l'interpolation linéaire articulaire. Le compromis entre l'optimisation géométrique, l'optimisation temporelle et la simplicité de calcul a conduit à une solution sous-optimale par rapport au temps.

TABLE DES MATIERES

LISTE DES SIGLES.

Sigles	Significations
ILA	Interpolation linéaire articulaire.
MGS	Méthode Globale de Solution.
AP	Algorithme Proposé.
PGT	Planification Globale de la Tâche.
PID	Proportionnel, Intégral et Différentiel.
PLT	Planification Locale de la Trajectoire.
RPY	Matrice d'orientation de la main par rapport au système de référence (Roll-Pitch-Yaw).

LISTE DES TABLEAUX.

LISTE DES FIGURES.

LISTE DES VARIABLES.

Variables	Significations
A_i^{i-1}, A_i	Matrice de transformation homogène des coordonnées qui relie le système d'axes "F_i" par rapport au système d'axes "F_{i-1}".
A_{ijk}	Position articulaire au début du sous-segment "k" (radian).
$Amax_{ijk}$	Accélération articulaire maximale pour le sous-segment "k" (rad/s^2).
B,b	Cet indice réfère au système de base.
B_{ijk}	Egale à A_{ijk+1}.
C(t)	Matrice (nxnxn) qui tient compte des effets centripètes et de Coriolis sur les efforts articulaires.
C_{ijk}	Vitesse articulaire au début du sous-segment "k" (rad/s).
D_{ijk}	Egale à C_{ijk+1}.
$\vec{e}_{wj}$	Vecteur vitesse articulaire unitaire (rad/s).
E_{ijk}	Accélération articulaire initiale (rad/s^2).
F_{ijk}	Accélération articulaire finale (rad/s^2).
G	Matrice de tranformation homogène qui relie le système d'axes de la main par rapport au dernier système d'axes de la matrice "T".
$\vec{G}(t)$	Vecteur qui tient compte de l'effet de gravité sur les efforts articulaires.
Hz	Unité de fréquence (fois/sec. ou Hertz).

Variables	Significations (suite)
i	Réfère à l'articulation "i" (i = 1,n).
$I(t)$	Matrice généralisée (nxn) d'inertie du robot.
j	Point de visite "j" ou segment "j".
J	Matrice jacobienne.
k	Sous-segment "k".
Kw_j	Grandeur du vecteur vitesse articulaire généralisée au point "j" (rad/s).
L_i	Longueur du lien "i".
Lc_i	Position du centre de gravité par rapport à F_{i-1}(mètres).
m	Nombre de segments dans une trajectoire.
m_i	Masse du lien "i" (Kg).
M	Masse transportée par le robot (Kg).
n	Nombre de degrés de liberté du manipulateur.
$\vec{n}(t)$	Vecteur efforts articulaires (N ou N-m).
$\vec{n}_a(t)$	Vecteur efforts articulaires sans les effets de la gravité (N ou N-m).
$\vec{p}_j$	Position cartésienne de la main au point "j" par rapport au système de référence (m).
$\vec{pr}_j$	Position linéaire du point de rectification au segment "j" (m).
$\vec{Pr}_j$	Position généralisée du point de rectification au segment "j" (m ou rad.).

Variables	Significations (suite)
q_i	Coordonnée généralisée.
$q_{ijk}(t)$	Trajectoire articulaire (radian).
R, r	Cet indice réfère au système de référence.
r(t)	Facteur d'échelonnement du temps.
T	Position généralisée de l'organe terminal du manipulateur par rapport au système de base.
T_{acc}	Temps requis par le robot pour accélérer du repos à sa vitesse maximale (sec).
T_{ijk}	Demi-temps alloué pour le sous-segment "k" (sec.).
T_j	Temps alloué pour le segment "j" (sec.).
$\vec{v}_j$	Orientation de la vitesse linéaire cartésienne au point "j" (m/s).
$\vec{V}_j$	Vecteur vitesse cartésienne généralisée (m/s et rad/s).
$Vmax_{ijk}$	Vitesse articulaire maximale (rad/s).
$\vec{w}_j$	Vitesse de rotation cartésienne unitaire (rad/s).
X	Position généralisée de l'organe terminal par rapport au système de référence.
Z	Matrice de transformation homogène qui relie le système de base par rapport au système de référence.

Variables	Significations (suite)
α_i	Accélération articulaire pour le lien "i" (rad/s²).
α_{ijk}	Accélérations articulaires maximales (r/s²).
ϵ_j	Facteur de forme (%).
$\vec{\Phi}_j$	Orientation cartésienne de la main au point "j" par rapport au système de référence en utilisant la représentation RPY.
$\vec{\Phi r}_j$	Orientation cartésienne du point de rectification au segment "j" (rad).
τ_i	Moment de torsion de l'articulation "i" (N-m).
w_i	Vitesse articulaire pour l'articulation "i" (rad/s).

GLOSSAIRE: la traduction des termes techniques.

Termes techniques	Significations
asservissement en position (position control)	Le système d'asservissement peut utiliser la position (position, vitesse ou accélération) dans sa boucle de retour pour améliorer la poursuite de la trajectoire.
asservissement optimal (optimal control)	Cela consiste à trouver la rétroaction optimale d'un système asservi. L'optimalité fait référence à un indice de performance quelconque.
asservissement par "bang-bang"	Le régulateur demande au moteur de fournir son effort maximum.
cinématique directe (forward kinematic)	Connaissant le mouvement articulaire du robot, on veut calculer le mouvement cartésien.
cinématique inverse (inverse kinematic)	Connaissant le mouvement cartésien, on veut calculer le mouvement articulaire du robot.
coordonnées généralisées (generalized coordinates)	Les coordonnées généralisées réfèrent à un vecteur constitué de 3 mouvements linéaires et 3 mouvements de rotation.
courbe articulaire	La cinématique inverse de la courbe cartésienne produit une courbe pour chaque articulation.
courbe cartésienne (path)	C'est la courbe géométrique générée par la main dans l'espace cartésien. Cette courbe comprend la position et l'orientation cartésiennes de la main.
dépassement (overshoot)	La trajectoire peut dépasser le point visé.

Termes	Significations (suite)
échelonnement dynamique du temps (dynamic time scaling).	C'est une propriété dynamique des mécanismes avec une chaîne cinématique.
espace articulaire (joint space)	Les degrées de liberté du robot constituent les variables de l'espace.
espace cartésien (cartesian space)	C'est l'espace tridimensionnel cartésien tel qu'on le connaît.
hors-ligne (off-line).	Le calcul est fait avant l'exécution.
interpolation linéaire articulaire (joint motion interpolation)	Cet algorithme fut présenté par PAUL [50].
interpolation linéaire cartésien (cartesian motion interpolation)	Cet algorithme fut présenté par PAUL [50].
articulation limitative	Pour optimiser le temps, seulement un des moteurs doit être saturé.
planification de la trajectoire (trajectory planning).	Cela consiste à concevoir une trajectoire.
points intermédiaire (via points)	Les points situés entre la position initiale et finale.
poursuite de la trajectoire (path tracking)	Le système d'asservissement doit minimiser l'erreur entre la trajectoire générée et celle donnée.
programmation dynamique (dynamic programming)	Cela s'appuie sur la recherche intensive à l'aide de grillage.
raccordement polynômial (spline)	Cela consiste à relier des polynômes de faibles degrés pour générer une trajectoire continue.

Termes	Significations (suite)
raccordement polynomial partiel (X-spline)	Ce type de raccordement polynomial utilise un nombre restreint de points (3 ou 4) pour produire un segment de trajectoire.
recherche intensive (graph search)	Ce processus d'optimisation est basé sur une discrétisation de l'espace de solution. Il suffit de choisir la solution optimale.
segment	C'est la trajectoire entre deux points successifs. Un segment est subdivisé en sous-segments.
système de base (base frame)	C'est le système d'axes attaché à la base du robot.
système de référence (reference frame)	C'est le système global de référence.
temps réel ou en-ligne (on-line)	Le calcul est fait en cours d'exécution.

Introduction:

L'apparition de tâches industrielles de plus en plus répétitives rend l'automatisation de la production attrayante. Cela permet de réduire les temps morts; ce qui augmente la productivité. Face à une concurrence internationale vive, l'industrie moderne doit se tourner vers l'automatisation de ses procédés industriels; la robotique représente une des façons de le faire.

A l'heure actuelle, on compte plusieurs applications où la mise en oeuvre de la robotique s'est avérée un succès: le chargement/déchargement de machines, l'assemblage simple, l'inspection de pièces, les procédés de soudure et de peinture, etc. Ce qui constitue une variété de tâches à automatiser. Pour accomplir ces différents travaux, le robot doit suivre une trajectoire qui représente la tâche à accomplir. La multitude d'applications crée un besoin varié de trajectoires: courbe cartésienne précise, mouvement rapide du manipulateur, vitesse cartésienne désirée, etc.

Avant de poursuivre la discussion sur la planification de trajectoire, il faut tout d'abord se demander ce que signifie le terme "trajectoire". Par définition, une trajectoire réfère à l'histoire de la position, de la vitesse et de l'accélération en fonction du temps. Cela suggère qu'une trajectoire est composée d'une courbe géométrique et d'un temps. Certaines tâches requièrent une courbe précise dans l'espace cartésien; pour d'autres, le temps est beaucoup plus important. Plusieurs algorithmes utilisent seulement des critères géométriques pour la planification de la trajectoire; la trajectoire est générée en utili-

sant des contraintes cinématiques (position, vitesse et accélération) seulement. Une trajectoire ayant deux composantes, il est aussi important de déterminer et d'optimiser simultanément le temps associé à la courbe géométrique pour garantir l'optimalité d'une trajectoire.

Le problème de base de la planification de la trajectoire consiste à déplacer un robot d'une position initiale à une position finale en spécifiant le mouvement du manipulateur (T) par rapport au système de référence (R). De façon générale, il existe deux méthodes pour décrire une trajectoire: la description explicite et la description implicite. La description explicite consiste à donner une fonction analytique continue de la courbe cartésienne. Dans la seconde méthode, seulement un certain nombre de points est spécifié pour décrire la trajectoire. Un mouvement implicite est alors supposé entre ces points. Souvent, plus de détails sont requis sur les tâches implicites. On ajoute alors des points intermédiaires pour controller la forme de la trajectoire ou pour éviter des objets contenus dans l'espace de travail. En mettant beaucoup de points intermédiaires, une courbe cartésienne est approximée. La description implicite permet une description plus simple de la trajectoire, ce que l'utilisateur d'un robot préfère pour faciliter son travail.

Les deux types de description de la tâche utilisent l'espace cartésien. Ce mouvement étant généré par une chaîne cinématique (le robot), la trajectoire doit respecter les contraintes physiques du manipulateur. Il devient alors évident que la trajec-

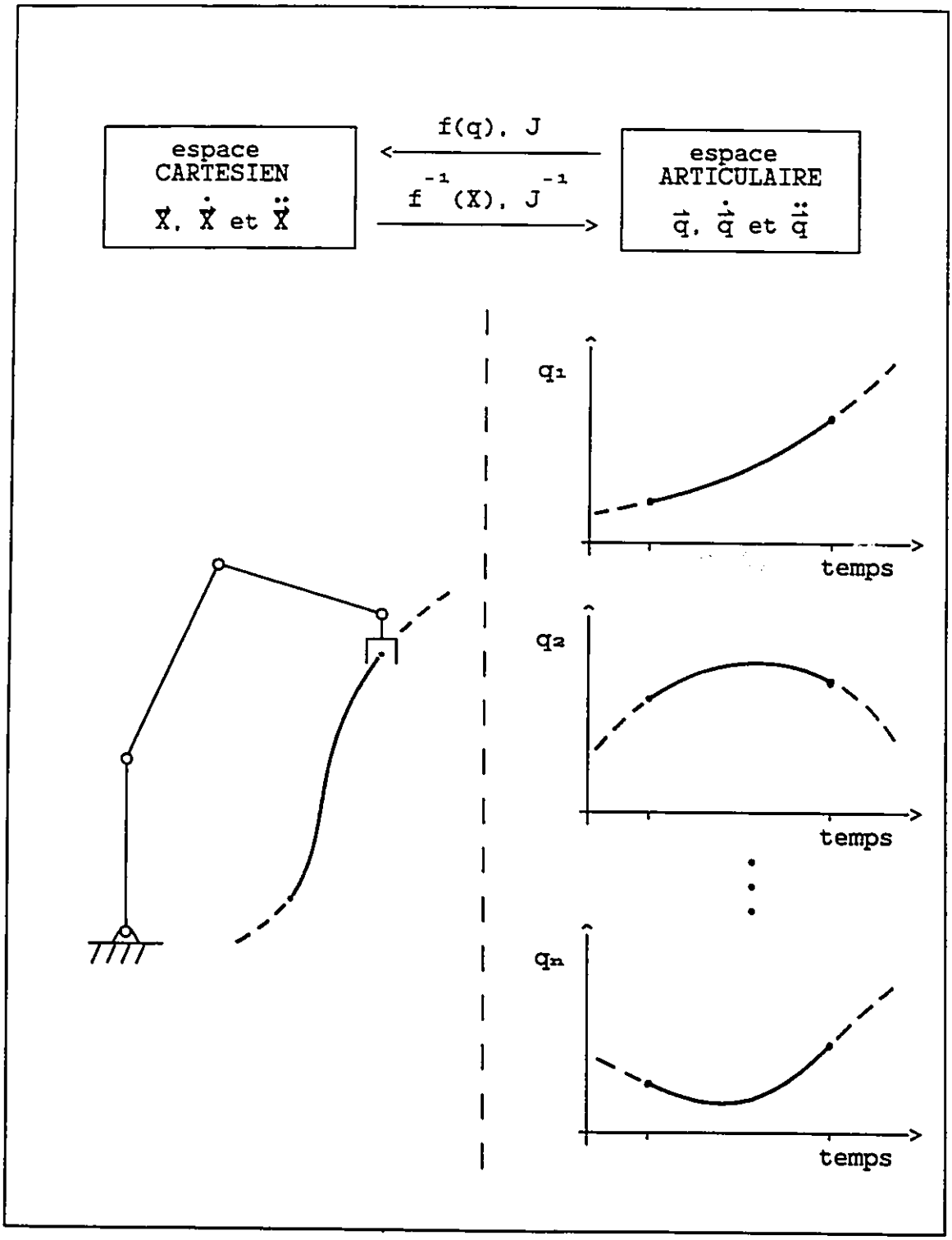


Fig.1.0 - la dualité spatiale.

toire doit satisfaire des contraintes mixtes contenues dans l'espace cartésien et dans l'espace articulaire (figure "1.0"). Mathématiquement, cela représente une relation bijective entre deux domaines.

La conception d'une trajectoire paraît simple. Souvent, on confond la trajectoire avec la courbe cartésienne générée. Mathématiquement, la courbe cartésienne est une fonction d'un seul paramètre "s" qui représente la distance sur la courbe. Une trajectoire n'est pas qu'une simple ligne dans l'espace tridimensionnel, elle résulte de la coordination de plusieurs articulations pour générer le mouvement désiré. La difficulté reliée à la planification de la trajectoire provient de la dualité spatiale des contraintes, des limites physiques du robot, de la présence d'objets dans l'espace de travail, des critères d'optimisation et de la non-linéarité du mouvement.

La trajectoire représente une séquence de configurations dans le temps qui sert d'entrée pour le système d'asservissement du robot. Selon LEE [33], l'approche courante pour la conception du système d'asservissement des robots industriels est de considérer chaque articulation comme un servomécanisme simple. Pour ces robots, le système d'asservissement est conçu pour la pire configuration du robot; donc, le manipulateur opère en dessous de sa capacité. Pour corriger ce problème, il faut éliminer les contraintes globales sur l'asservissement. Cela est possible en introduisant la dynamique dans la planification de la trajectoire ou lors de l'asservissement.

La présente thèse consiste à faire une étude sur la planification des trajectoires. Pour ce faire, il faudra analyser l'état des connaissances et présenter une nouvelle façon de concevoir des trajectoires. De façon plus spécifique, on veut développer un algorithme de planification de trajectoire pouvant être utilisé pour une variété de tâches et pouvant optimiser le temps pour un robot avec plus de trois degrés de liberté. Comme mentionné précédemment, une trajectoire peut être optimisée lors de la planification ou lors de l'asservissement. D'un point de vue scientifique, la dynamique fait généralement partie de l'asservissement et est utilisée pour produire des règles pour la rétroaction du système d'asservissement. Il n'y a donc aucune garantie que la trajectoire planifiée soit réalisable. L'introduction de la dynamique dans la planification de la trajectoire permet d'optimiser une trajectoire lors de la planification, d'éviter la conception de trajectoires irréalisables et d'éliminer le besoin de les corriger lors de l'asservissement.

Afin d'optimiser la trajectoire, on fera un compromis entre la courbe cartésienne et le temps. Pour ce faire, on utilisera une approximation multi-polynômiale entre les points dans le domaine articulaire pour générer une courbe cartésienne prévisible. Puis, le temps sera minimisé sous l'influence de la dynamique du manipulateur. Ce qui mènera à une solution sous-optimale par rapport au temps et à la courbe cartésienne, car un compromis est fait entre eux.

Pour débuter l'analyse sur la génération de trajectoire, le chapitre "1" énumèrera les caractéristiques que doit posséder une trajectoire idéale. Puis, on exposera l'état des connaissances relatives à la planification de la trajectoire. Au chapitre "2", les aspects généraux relatifs à l'algorithme proposé seront étudiés. Cela consistera à expliquer la méthode globale de solution et les principes d'optimisation utilisés. Au chapitre "3", on décrira plus particulièrement le nouvel algorithme de calcul en axant la présentation sur la modélisation mathématique et les étapes de calcul. Au dernier chapitre, on présentera une simulation numérique pour une tâche particulière. Les résultats de l'interpolation linéaire articulaire et de l'algorithme proposé seront ensuite comparés.

CHAPITRE I

L'ETAT DES CONNAISSANCES.

L'étude de la génération de trajectoire requiert l'analyse des disciplines suivantes: la cinématique, la dynamique, la planification de trajectoire, l'asservissement et l'évitement d'objets. Pour éviter de surcharger le texte, seule la planification de la trajectoire sera exposée dans ce chapitre. Les autres disciplines sont brièvement présentées aux annexes "A à D".

Avant de faire l'analyse des différents algorithmes de planification de trajectoire, on fera ressortir les caractéristiques idéales d'une trajectoire. Ce qui donnera des critères de comparaison.

1.1 - Les caractéristiques idéales d'une trajectoire:

Dans cette section, on expose les caractéristiques idéales d'une trajectoire en faisant abstraction des difficultés physiques. Ces caractéristiques ont été mentionnées par différents auteurs; entre autres, BRADY et al. [5], LEE et al. [33] et PAUL [50].

La dualité spatiale de la planification de trajectoire fait que certaines caractéristiques sont naturellement reliées à un ou l'autre des espaces. Les trois prochaines ont un rapport direct avec la courbe cartésienne générée. Tout d'abord, la prévisibilité de la courbe cartésienne est importante dans un contexte d'évitement d'objet pour prévenir toutes collisions du robot avec son environnement. Cela signifie que l'opérateur doit être

capable de prédire le mouvement qu'exécutera le robot. Un mouvement prévisible n'est pas nécessairement une ligne droite. La deuxième caractéristique, la précision de la courbe cartésienne, est importante dans les opérations où un déplacement précis de la main est requis; la soudure à l'arc en est un exemple. La dernière caractéristique, la continuité cartésienne, représente une contrainte physique de position pour permettre l'existence d'une solution.

Le déplacement du manipulateur est produit par le mouvement local des différents liens dans l'espace articulaire. Puisque chaque moteur a des performances physiques limitées, la planification ne doit pas produire une trajectoire en dehors des limites articulaires de positions, de vitesses, d'accélérations et d'efforts; ce qui produirait une trajectoire impossible. De plus, il faut aussi s'assurer de la continuité du mouvement. Au minimum, les positions et les vitesses articulaires doivent être continues. La continuité des accélérations est aussi souhaitable: les mouvements brusques et impulsifs augmentent l'usure des mécanismes et provoquent des vibrations.

Plusieurs autres caractérisques générales sont souhaitables et sont énumérées ci-après. La généralité de l'algorithme n'est pas essentielle, mais elle permet l'utilisation d'un algorithme unique pour générer la variété de tâches. En milieu industriel, l'algorithme doit être simple à utiliser pour faciliter le travail de l'opérateur; une description simple de la tâche est requise. La caractéristique suivante concerne la possibilité de varier le temps. Un robot travaille généralement avec d'autres

machines. Pour coordonner les activités dans la cellule de travail, l'algorithme doit allouer la possibilité de fixer le temps entre les points. Par contre, dans les situations où le temps n'est pas prescrit, la méthode doit permettre la minimisation du temps pour augmenter la productivité. La prochaine caractéristique, l'efficacité du calcul, facilite la mise en oeuvre de l'algorithme. Une trajectoire simple à calculer et rapide d'exécution permet l'utilisation de l'algorithme en temps réel de calcul. De plus, en théorie, la trajectoire doit être facilement transférable au système d'asservissement pour satisfaire un taux de mise à jour (updating rate) de 20 à 200 Hz selon BRADY [5]. Pour terminer, la trajectoire ne doit pas dégénérer et doit garder les mêmes performances près d'une singularité.

1.2 - Les trajectoires: l'approche cinématique.

Ici, on étudie les algorithmes de planification de trajectoire qui utilisent une approche purement cinématique. L'introduction de contraintes dynamiques est difficile, car la connaissance de la cinématique est nécessaire pour le calcul de la dynamique. En négligeant l'aspect dynamique du mouvement, ces algorithmes peuvent créer des trajectoires impossibles à exécuter; elles sont alors modifiées au cours de l'asservissement.

On les catégorise généralement en deux groupes: la génération de trajectoire dans le domaine cartésien et dans le domaine articulaire. Le choix de l'un ou l'autre des espaces de travail comporte des avantages et des inconvénients.

1.2.1 - Les algorithmes dans l'espace cartésien:

La génération de trajectoire dans le domaine cartésien est moins fréquente à cause de la difficulté à respecter les contraintes données dans le domaine articulaire. Parmi ces algorithmes, trois approches sont plus communes: la génération de courbe cartésienne, l'interpolation linéaire cartésienne et la conversion des contraintes articulaires en contraintes cartésiennes.

La génération de courbes cartésiennes consiste à spécifier une courbe cartésienne désirée avec une contrainte sur la charge utile transportée. Les robots industriels d'aujourd'hui utilisent ce type de planification de trajectoire. Une fois planifiée, la trajectoire est envoyée au système d'asservissement qui fait l'évaluation répétitive des variables d'asservissement en utilisant le calcul du mouvement inverse.

Basé sur la connaissance des limites expérimentales sur les vitesses et les accélérations cartésiennes, PAUL [49] a proposé un algorithme appelé "l'interpolation linéaire cartésienne". Entre chaque point de visite, on coordonne le mouvement du manipulateur en utilisant des vitesses cartésiennes constantes. Cela génère des courbes cartésiennes linéaires. Pour permettre la continuité du mouvement aux points intermédiaires, les segments linéaires sont reliés par une transition.

Dans la troisième approche, LUH et LIN [44] ont tenté de convertir les limites de vitesses et d'accélérations du domaine articulaire au domaine cartésien. Ils ont utilisé l'inter-

polation linéaire cartésienne entre les points. Puis, ils ont optimisé la trajectoire en tenant compte des contraintes cartésiennes développées à partir de l'espace articulaire. La conversion des limites articulaires dans le domaine cartésien est difficile puisque la relation entre les deux domaines est non-linéaire. De plus, la dynamique d'un manipulateur génère des équations couplées.

Plusieurs raisons font qu'il est avantageux de choisir le domaine cartésien pour planifier une trajectoire. La planification étant faite dans le domaine cartésien, les caractéristiques reliées à cet espace sont satisfaites: les trajectoires sont prévisibles, précises et continues. Ces algorithmes permettent la génération simple de ligne cartésienne droite. Dans ce cas, la longueur de la courbe cartésienne est courte (sans être la plus rapide) entre le point initial et final; et ce mouvement rectiligne minimise les forces d'inerties sur la main et sur les objets transportés. Il faut aussi noter que l'interaction entre l'objet et l'espace cartésien est meilleure pour ces algorithmes, car le mouvement de l'objet correspond à la courbe cartésienne planifiée.

D'un autre côté, la planification dans l'espace cartésien ne permet pas de vérifier directement les limites physiques du domaine articulaire; ces algorithmes n'assurent pas l'existence physique d'une solution puisque l'inverse cinématique est requise. Cette inverse requiert beaucoup de calcul et elle est généralement calculée lors de l'asservissement. Etant donné la faible rapidité des ordinateurs actuels, il est

difficile de satisfaire le taux d'asservissement requis. De plus, la correspondance continue entre le domaine cartésien et articulaire fait apparaître le problème de singularité. La solution de la cinématique inverse de position n'est pas toujours unique; il faut surveiller la continuité du mouvement dans le domaine articulaire. Pour terminer l'énumération des désavantages, il faut mentionner les difficultés reliées à l'optimisation du temps et à la vérification de l'absence de collision aux niveaux des liens.

1.2.2 - Les algorithmes dans l'espace articulaire:

La planification dans le domaine articulaire présente moins d'inconvénients que la conception de trajectoire dans l'espace cartésien.

De façon générale, la méthode de solution se présente comme suit. Un nombre de points connus dans le domaine cartésien sont ramenés dans l'espace articulaire par la cinématique inverse de position. Puis une ou plusieurs fonctions analytiques différentes pour chaque articulation sont utilisées pour relier les positions articulaires.

Pour faciliter la solution, plusieurs auteurs posent des hypothèses simplificatrices. La plus commune consiste à s'assurer de l'existence d'une solution en faisant un découplement de la trajectoire: par exemple, PAUL [50], alloue un temps nécessaire pour passer de la vitesse minimale à la vitesse maximale dans ses transitions. Cette hypothèse réduit l'optimalité d'une trajectoire. D'autres auteurs ne se soucient

tout simplement pas de l'existence d'une solution ou bien ils restreignent le type de trajectoire modélisée.

On peut séparer les algorithmes en trois approches: un polynôme d'ordre élevé, une fonction unique pour chaque segment et une multi-fonction pour chaque segment.

A - Un polynôme unique d'ordre élevé:

Pour satisfaire un nombre important de contraintes physiques (incluant la possibilité de collision), BRADY et al. [5] mentionnent qu'une fonction unique pour chaque articulation peut être utilisée pour relier tous les points. Cette approche n'est plus vraiment utilisée à cause des nombreux désavantages qui s'y rattachent: l'ordre du polynôme est proportionnel au nombre de contraintes; la difficulté reliée à la vérification des contraintes est proportionelle à l'ordre du polynôme; la précision du calcul diminue en fonction de l'ordre du polynôme; la fonction provoque plusieurs dépassements dus à la présence de plusieurs racines; les coefficients sont difficiles à calculer; la méthode de solution varie en fonction de l'ordre; la fonction est dépendante de l'ensemble des points; et les contraintes sont locales (appliquées aux points).

B - Une fonction pour chaque segment:

Les points de la trajectoire peuvent être reliés par différentes fonctions analytiques en imposant des contraintes de continuité. Généralement, les fonctions utilisées sont des polynômes de faible degré, car elles facilitent la

résolution du problème. Cette modélisation mathématique permet d'éliminer la majorité des inconvénients reliés au polynôme unique d'ordre élevé.

Plusieurs ordres de polynômes peuvent être utilisés. La fonction linéaire représente le polynôme le plus simple. Cet ordre n'est habituellement pas utilisé puisque les vitesses sont discontinues aux points intermédiaires; un polynôme cubique est plus approprié. Par contre, le polynôme du troisième degré provoque une accélération discontinue entre les segments lorsque le polynôme satisfait les deux contraintes initiales et les deux contraintes finales. Cet inconvénient est enlevé en choisissant un polynôme du cinquième degré.

Dans le cas d'une fonction unique pour chaque segment, le raccordement polynômial est la méthode la plus utilisée. Cette méthode consiste à relier les positions articulaires par l'ajustement de la courbe (curve fitting). Des polynômes cubiques ou du 4^{e} degré sont généralement utilisés pour approximer le mouvement d'un segment. Selon LIN et al. [35], cela génère des trajectoires lisses avec de faibles dépassements n'exédant pas 10°.

Les chercheurs se sont surtout intéressés à quatre problèmes reliés à l'utilisation du principe de raccordement polynômial: l'approximation articulaire d'une courbe cartésienne, la déviation de cette approximation par rapport à la courbe cartésienne désirée, l'optimisation temporelle et le calcul en temps réel.

Le développement de la méthode du raccordement polynômiale provient de l'observation suivante: puisqu'il est très difficile d'imposer des contraintes articulaires sur une courbe cartésienne désirée, il est préférable d'utiliser une approximation articulaire pour la générer. LIN et CHANG [36] ont proposé ce genre d'algorithme. En utilisant une séquence de points intermédiaires transformées dans le domaine articulaire par la cinématique inverse de position, la trajectoire articulaire est interpolée en faisant un raccordement polynômial. La formulation mathématique est basée sur l'ajustement de la courbe (curve fitting) en prenant seulement quatre points; ce qui permet un calcul en temps réel. Cette méthode ne tient pas compte des limites physiques du robot, et aucune optimisation est présente.

L'approximation de la courbe cartésienne dans le domaine articulaire introduit une déviation entre la courbe obtenue et la courbe désirée. En utilisant une méthode similaire à celle présentée précédemment, LUH et LIN [43] ont réduit l'erreur de positionnement en faisant une approximation par les moindres carrés. Si l'erreur demeure plus élevée qu'une limite donnée, alors il faut augmenter le nombre de points pris sur la courbe cartésienne. Dans le même ordre d'idée, BENHABIB et al [3] ont minimisé cette erreur en utilisant le degré redondant d'un manipulateur à sept degrés de mobilité.

Dans le but d'augmenter les performances d'une trajectoire, LIN et al. [35] ont présenté un algorithme qui op-

timise le temps entre chaque point sous l'influence de contraintes globales sur les vitesses, les accélérations et les dérivées des accélérations articulaires. Leur méthode requiert beaucoup de calcul. Afin de respecter la contrainte du taux d'asservissement, ils ont séparé le problème de l'asservissement optimal en deux étapes: planification de la trajectoire (hors-ligne) et poursuite de la trajectoire en temps réel. Pour ce faire, "n-2" points du domaine cartésien sont choisis et deux points libres (en position) sont ajoutés pour faciliter la solution. En utilisant le principe du raccordement polynômial, une matrice d'ordre "n" est construite. L'optimisation du temps se fait par calculs itératifs sous l'influence des contraintes physiques ci-haut énoncées.

Généralement, les algorithmes de raccordement polynômial utilisent le calcul hors-ligne à cause de sa formulation mathématique (grosse matrice). Pour permettre un calcul en temps réel, CHAND et DOTY [6] ont utilisé le raccordement polynômial partiel (X-spline). Ils ont déterminé le nombre de points qui doivent être connus d'avance pour le calcul du raccordement partiel. Cela consiste à utiliser un nombre restreint de points pour accélérer le calcul.

C - Une multi-fonction pour chaque segment:

Comparativement au raccordement polynômial, la multi-fonction est une solution plus locale puisqu'elle calcule la trajectoire d'un segment en utilisant seulement les points

adjacents du segment. L'exemple le plus connu est l'interpolation linéaire articulaire (ILA). Cette méthode fut présentée par PAUL [50] et a été modifiée par TAYLOR [70]. L'ILA est présentée à l'annexe "E".

a) l'interpolation linéaire articulaire:

Globalement, l'interpolation linéaire articulaire consiste à assumer un mouvement linéaire pour chaque articulation. La continuité du mouvement est satisfaite par l'ajout de transitions polynômiales du 5^e degré à la jonction des segments. L'ordre du polynôme se réduit à quatre à cause de la symétrie imposée aux transitions. Chaque transition débute dans le segment précédent le point et se termine dans le segment suivant. Pour s'assurer de la possibilité d'une telle trajectoire, on pose les deux contraintes suivantes: les vitesses articulaires sont limitées par des limites globales et le temps de la transition "$2 \cdot T_{acc}$" est suffisamment long pour permettre au manipulateur d'accélérer de sa plus faible vitesse à sa plus grande. En fait, on limite l'accélération articulaire par rapport à la pire configuration du manipulateur. Cela implique que le temps de chaque segment est toujours supérieur ou égale à "$2 \cdot T_{acc}$".

L'interpolation à la jonction des segments fait que le manipulateur ne visite pas les points intermédiaires. Pour passer par ceux-ci, le manipulateur doit s'y arrêter. Cela est inacceptable dans un contexte d'optimisation puisque l'arrêt augmente le temps de "$2 \cdot T_{acc}$".

Le temps de transition est une constante pour un manipulateur et est calculé par la relation suivante:

$$T_{acc} = \underset{1 \leq i \leq n}{MAX} \left[\frac{w_i^{max}}{\alpha_i^{max}} \right]$$

où "w^{max} et α^{max}" sont les valeurs pour la pire configuration du robot.

Les actuateurs d'un robot ont des caractéristiques assez différentes: le porteur est généralement plus lent que le poignet. Par exemple, BRADY [5] mentionne que le rapport de la vitesse maximale du lien le plus rapide sur le lien le plus lent est d'environ "4" pour le manipulateur Stanford. Donc, pour cet algorithme, toutes les articulations sont limitées par le "T_{acc}" du lien le plus lent; ce qui réduit l'optimalité de la solution.

b) l'approximation linéaire de TAYLOR [70]:

Etant donné les désavantages reliés à la planification de la trajectoire dans l'espace cartésien, TAYLOR [70] a proposé une approximation articulaire de la courbe cartésienne. Il est difficile d'effectuer un mouvement rectiligne dans l'espace articulaire, car un mouvement linéaire articulaire ne génère pas une ligne droite cartésienne. Cette observation est illustrée à la figure "1.1".

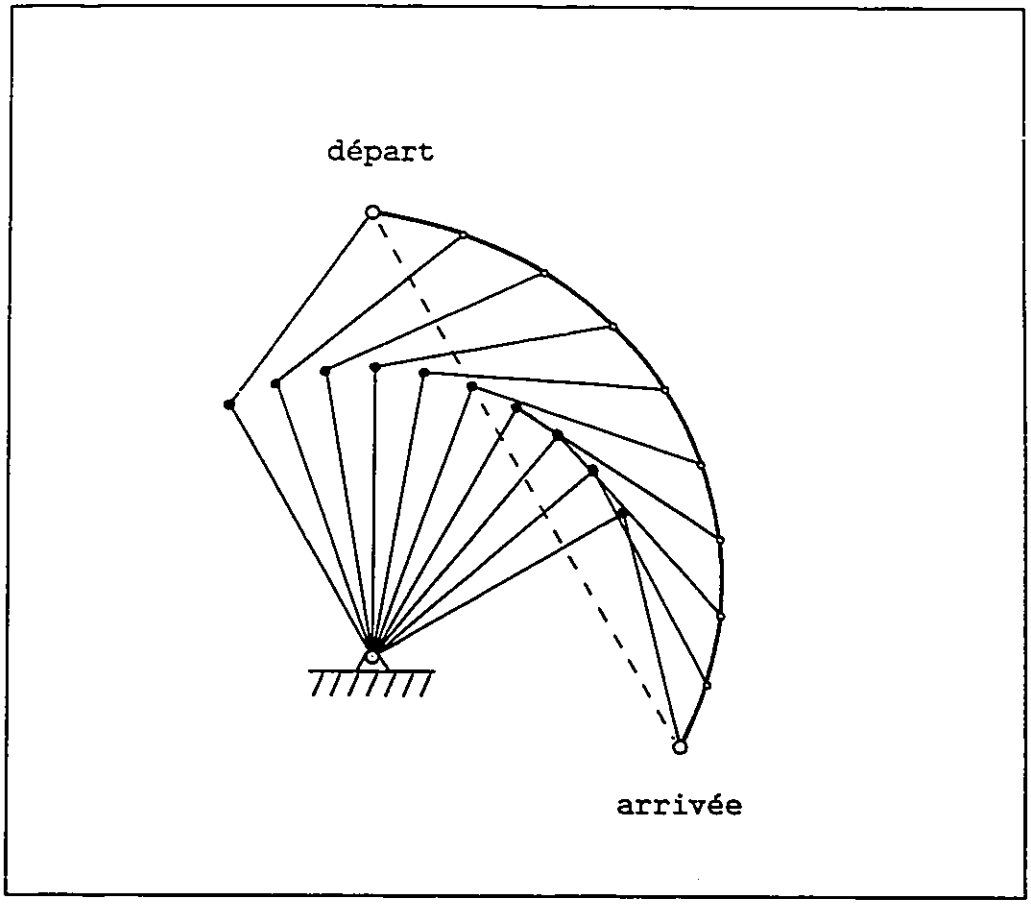


Fig.1.1 - Un mouvement linéaire dans le domaine articulaire ne génère pas une courbe cartésienne rectiligne.

L'algorithme de TAYLOR [70] est un compromis entre l'efficacité de l'implémentation et la précision de la courbe cartésienne. Cette approche utilise une première phase pour calculer un nombre suffisant de points intermédiaires sur la courbe cartésienne pour limiter la déviation de la courbe cartésienne générée par rapport à la courbe désirée. Pour ajouter ces points, il assume que le maximum de déviation arrive au milieu d'un segment. Cela n'est généralement pas vrai, mais cette

simplification est raisonnable. La deuxième phase consiste à relier ces points en utilisant l'ILA.

Dans l'ILA, le robot passe par les points intermédiaires seulement s'il arrête à ces points. TAYLOR [70] ne force pas l'arrêt aux points, mais l'ajout considérable de points diminue la distance "$\Delta\theta$" d'un segment. Cette distance étant courte par rapport à la limite de temps "$2 \cdot T_{acc}$", la vitesse du robot aux points intermédiaires est presque nulle:

$$\frac{\Delta\theta}{2 \cdot T_{acc}} \approx 0$$

Ces vitesses quasiment nulles aux points augmentent le temps de la trajectoire. Plus le nombre de points intermédiaires est élevé, plus la déviation de la courbe générée est faible par rapport à celle désirée. Par contre, cette précision de la courbe cartésienne se fait au détriment du temps de la trajectoire.

Les performances de cet algorithme près d'une singularité sont moins bonnes. De fait, l'ajout de points intermédiaires près d'une singularité peut faire une faible correction sur le mouvement: une petite erreur dans l'espace cartésien peut mener à une grosse variation dans l'espace articulaire, et vice-versa.

La planification dans le domaine articulaire satisfait un bon nombre des caractéristiques idéales d'une trajectoire. Comme on peut s'y attendre, la planifi-

cation articulaire possède toutes les caractéristiques associées au domaine articulaire. La limitation des positions, des vitesses et des accélérations articulaires est facilement accomplie. Il en est de même pour la continuité du mouvement puisque des contraintes de continuité sont imposées aux points. Un mouvement continu dans le domaine articulaire génère un mouvement continu dans le domaine cartésien.

Les trajectoires développées dans le domaine articulaire peuvent être appliquées à une variété de tâches. Ces trajectoires peuvent servir à la génération de courbe cartésienne en utilisant une approximation dans le domaine articulaire. De plus, les trajectoires ainsi conçues sont plus faciles à optimiser puisque les contraintes physiques sont associées au domaine articulaire. Ainsi, on peut obtenir un mouvement rapide du manipulateur en optimisant son mouvement par rapport au temps.

La spécification de la trajectoire en utilisant le principe de satisfaction de contraintes aux points rend la description de la tâche simple puisque l'opérateur a peu de positions et de vitesses à donner. Cette tâche étant convertie dans le domaine articulaire par l'inverse cinématique, la solution au problème de planification est simple et rapide; car il n'y a pas de correspondance continue entre le domaine cartésien et articulaire. Cela élimine aussi les singularités mathématiques. En évitant la transformation du domaine cartésien au domaine ar-

ticulaire lors de l'asservissement, le taux d'asservissement est facilement atteint.

La planification dans le domaine articulaire possède aussi quelques inconvénients. Dans une tâche planifiée dans l'espace articulaire, le robot visite une série de points qui définissent la courbe cartésienne. Ce qui résulte en l'application de contraintes locales seulement. Cela à tendance à produire des dépassements aux points; ce qui produit une courbe cartésienne imprévisible entre les points de visite. Les algorithmes planifiés dans le domaine articulaire ont plus de difficulté à satisfaire les caractéristiques relatives à l'espace cartésien. L'approximation de courbe cartésienne dans l'espace articulaire est naturellement moins précise.

De façon générale, la planification dans le domaine articulaire est beaucoup plus avantageuse que dans le domaine cartésien; l'algorithme proposé dans ce mémoire utilisera l'espace articulaire. L'utilisation de limites globales pour l'optimisation des trajectoires est inacceptable puisque les maximums de vitesses et d'accélérations varient beaucoup en fonction du mouvement du robot. Cela conduit à une solution sous-optimale.

1.3 - Les trajectoires: l'approche dynamique.

En dépit de sa proche relation avec la dynamique, les planifications de trajectoire conventionnelles n'en tiennent pas compte. Cela mène à la sous-utilisation des robots. Généralement, la dynamique fait partie du problème de l'asservissement du robot puisqu'elle permet de générer des lois pour la rétroaction du système. De plus, la connaissance de la trajectoire est essentielle au calcul de la dynamique; d'où l'idée d'optimiser le temps le long d'une trajectoire donnée lors de l'asservissement. Ce problème fait partie de "l'asservissement optimal direct".

Comme on peut s'y attendre, la résolution de ce problème en temps réel de calcul est très compliquée à cause de l'existence de limites physiques, de la non-linéarité des équations de la dynamique et de la présence d'objets dans l'espace de travail. Selon KAHN et ROTH [23], une forme explicite représentant la rétroaction du temps minimum est impossible. Ils ont alors proposé une approximation linéaire (linéarisation de la dynamique par expansion de Taylor) de la rétroaction optimale; ce qui conduit à une solution sous-optimale. La linéarisation de la dynamique permet l'application de la théorie de l'asservissement optimal linéaire. Cette théorie stipule que l'asservissement "bang-bang" est optimal pour les mécanismes linéaires. Depuis la mise en évidence de l'échelonnement dynamique du temps par HOLLERBACH [22], les approches basées sur la linéarisation de la dynamique par l'expansion de Taylor sont douteuses. Cette propriété montre que les termes d'ordre supérieur négligés lors de

la linéarisation ont la même signification par rapport aux termes d'accélérations; peu importe la vitesse du manipulateur.

Reconnaissant les difficultés de l'asservissement optimal direct, on utilise un calcul en deux étapes: (1) planification de la trajectoire (hors-ligne) et (2) poursuite de la trajectoire (temps réel). Cette dernière sert à minimiser la déviation de la position et de la vitesse par rapport à celles calculées lors de la planification de la trajectoire. Quant à elle, la première partie sert à optimiser la trajectoire en tenant compte de la dynamique du manipulateur. Cela est rendu possible par un calcul hors-ligne. Les solutions numériques peuvent mener à une solution optimale, mais elles ne tiennent pas compte des perturbations extérieures qui peuvent agir sur le système en mouvement. De plus, ces solutions sont optimales pour une condition initiale et finale données et le calcul doit être refait pour chaque nouvelle trajectoire.

Pour plusieurs applications, l'environnement de travail du robot est statique et répétitif. Dans ces cas, la planification hors-ligne de la trajectoire est possible à cause de l'aspect statique de la tâche. De plus, il est avantageux d'optimiser les tâches répétitives pour augmenter les performances de production; que ce soit le temps, l'énergie ou tout autre indice de performance.

Dans cette section, on étudiera la planification de la trajectoire qui utilise des contraintes dynamiques pour l'optimisation. On divisera ces algorithmes en trois groupes pour faciliter l'étude. Le premier groupe traite d'un algorithme qui

utilise une optimisation partielle. L'obtention de la solution optimale par rapport à un indice donné est produite par les deux approches suivantes: la théorie de l'asservissement optimal (deuxième groupe) et la discrétisation de l'espace solution (troisième groupe). La théorie de l'asservissement optimal consiste à exprimer la trajectoire sous forme paramètrique et à utiliser le principe de l'extrémum. Cela conduit à la résolution d'un ensemble d'équations différentielles couplées à un problème de deux valeurs aux limites. Le troisième groupe repose sur la recherche d'une solution par discrétisation de l'espace solution.

1.3.1 - L'optimisation partielle:

KIM et SHIN [27] ont développé un algorithme qui s'apparente beaucoup à l'ILA. Entre chaque position, ils utilisent des vitesses articulaires constantes en assumant qu'un des liens atteint sa vitesse maximale programmée. Puis, à la jonction de deux segments, on ajoute une transition composée de trois accélérations pour permettre la continuité du mouvement. L'algorithme utilise donc des multi-fonctions pour modéliser les segments.

Cette méthode possède des différences par rapport aux trajectoires planifiées avec l'approche cinématique. Comparativement à l'ILA, une erreur articulaire absolue peut être spécifiée pour la déviation à chaque point. A chaque transition, ils imposent des contraintes locales approximatives sur les accélérations articulaires développées à partir de la dynamique du manipulateur. Cela est appliqué seulement dans

le voisinage des points. Il n'y a donc aucune optimisation entre les transitions. Les vitesses maximales étant choisies pour la pire configuration du robot et pour le pire lien, la trajectoire n'est pas optimale par rapport au temps.

Cet algorithme possède un autre inconvénient. Il nécessite une certaine distance entre les points puisqu'il assume qu'une des articulations peut atteindre sa vitesse maximale. Cela permet de s'assurer de l'existence d'une solution et est en fait l'équivalent du "$2 \cdot T_{acc}$" de l'ILA. En imposant une distance minimale entre les points, cet algorithme ne s'applique pas à tous les types de tâches; l'approximation articulaire d'une courbe cartésienne en est un exemple.

1.3.2 - La théorie de l'asservissement optimal:

L'asservissement optimal est un problème mathématique résolu par le calcul des variations. Ce problème repose sur la résolution de l'équation d'Euler-Lagrange [7]. Cela permet de minimiser une fonction coût arbitraire qui dépend de l'indice de performance désiré. En robotique, on choisit généralement le temps ou l'énergie. Etant donné la complexité de la formulation mathématique, les algorithmes de cette catégorie assument que la courbe cartésienne est connue en tout temps.

A - L'utilisation du temps comme indice de performance:

Pour sa recherche de doctorat, J. E. BOBROW a montré que seulement une des articulations doit être asservie par un "bang-bang" pour assurer l'optimalité par rapport au

temps. Pour générer une trajectoire optimale selon le temps, un des moteurs doit fournir l'effort maximum (saturation d'un des moteurs); et on appelle cette articulation "l'articulation limitative". Pour les autres, il faut éviter les accélérations élevées puiqu'elles influenceraient l'articulation limitative par couplage dynamique.

Basé sur cette théorie, BOBROW et al. [4] ont présenté un algorithme qui produit un mouvement optimal selon le temps pour un robot avec deux ou trois degrés de liberté. Ce problème d'asservissement optimal est sujet aux contraintes dynamiques du manipulateur et se base sur l'idée suivante: si le manipulateur va à plus grande vitesse, alors le temps sera minimum. Au cours de son déplacement, le robot change de configuration; ce qui peut occasionner un changement de l'articulation saturée. Le problème à résoudre consiste donc à trouver ces points de changement.

DUBOWSKY et SHILLER [11] ont présenté une extension de l'algorithme précédent pour un robot avec six degrés de liberté. SHILLER et DUBROWSKY [63] ont par la suite fait une simulation numérique basée sur l'extension. En conclusion, ils ont montré qu'il est très difficile (et même presqu'impossible) d'utiliser la théorie de l'asservissement optimal pour développer la trajectoire pour un robot avec six degrés de liberté. Cela est causé par la complexité de la formulation mathématique.

B - L'utilisation de l'énergie comme indice de performance:

La minimisation de l'énergie consumée est très importante pour les mouvements rapides et la manipulation d'objets lourds. VUKOBRATOVIC et KIRCANSKI [71] ont développé un algorithme qui minimise l'énergie consommée le long d'une courbe cartésienne en optimisant la vitesse le long de la courbe donnée.

SCHMITT et al. [61] ont utilisé l'équation de "Raleigh-Rith" pour approximer la fonction coût associée à l'énergie. Cela permet de simplifier la solution.

Etant donné la complexité et la non-linéarité des équations dynamiques, et des différentes contraintes, l'application de la théorie de l'asservissement optimal mène à une analyse contenant plusieurs problèmes pratiques. La solution est faisable pour un robot avec deux ou trois degrés de liberté; pour les robots avec six degrés de liberté, la solution est presqu'impossible. L'application de la théorie de l'asservissement optimal conduit à un problème difficile à résoudre; se basant sur les caractéristiques idéales, cette approche ne satisfait pas la simplicité de solution et l'efficacité du calcul. Selon HOLLERBACH [22], ces algorithmes requièrent plusieurs heures de calcul. De plus, cette méthode demande la connaissance de la courbe cartésienne pour produire sa solution; seulement une des deux composantes (le temps) d'une trajectoire est optimisée.

1.3.3 - La discrétisation de l'espace solution:

Dans cette section, les algorithmes minimisent le temps en faisant varier la courbe cartésienne et en poussant l'articulation limitative à son maximum. Il existe plusieurs façons de résoudre ce problème, et quatre d'entre-elles sont présentées.

Une première façon, développée par RAJAN [55], combine la théorie de l'asservissement optimal et la recherche intensive. Elle consiste à exprimer la courbe cartésienne sous forme de courbes articulaires paramétriques et d'utiliser l'algorithme présenté par BOBROW [4] pour déterminer le temps minimum pour une courbe cartésienne donnée. Puis, il suffit de faire varier cette courbe cartésienne jusqu'à ce que le vrai minimum soit trouvé. Utilisant les mêmes concepts que la théorie de l'asservissement optimal, cela résulte aux mêmes inconvénients.

La seconde façon a été proposée par LIN et CHANG [37]. L'algorithme optimise un indice de performance quelconque en utilisant une approximation par raccordement polynômial. Ils augmentent le nombre de polynômes cubiques jusqu'à ce qu'il n'y ait plus d'amélioration. Donc, l'exactitude de l'approximation dépend du nombre de points libres "m", et la courbe exacte est atteinte lorsque "m" tend vers l'infini. Cette méthode n'est pas tellement efficace, car elle n'utilise pas de lois précises pour accélérer la convergence de l'algorithme (random walk).

SHIN et McKAY [65] ont présenté une autre façon d'optimiser la trajectoire en utilisant la programmation dynamique. Si l'on désire optimiser le temps pour une courbe cartésienne donnée, alors on peut exprimer toutes les courbes articulaires avec seulement deux paramètres (position et vitesse). Puis, l'utilisation d'un grillage fin permet de résoudre le problème. Puisque le choix de la courbe cartésienne affecte l'optimalité temporelle, ils ont présenté [67] un autre algorithme pour approximer la courbe cartésienne qui minimise le temps entre deux points. Une combinaison des deux méthodes précédentes permet de trouver la trajectoire optimale entre deux points [66].

La quatrième façon fut développé par SAHAR et HOLLERBACH [57]. Ils ont proposé un algorithme basé sur la recherche intensive d'une solution sous forme d'une grille d'optimisation. La méthode comporte trois aspects: un grillage dans l'espace articulaire qui représente l'ensemble des solutions possibles, l'utilisation de la propriété d'échelonnement dynamique du temps et un algorithme de recherche intensive (graph search). L'arbre de recherche représente toutes les solutions possibles, et il suffit de choisir le chemin qui donne l'optimalité par rapport au temps. Cela permet de trouver le temps minimum entre deux points arbitraires. Pour un robot planaire avec deux degrés de liberté, ils ont obtenu le profil optimal de vitesse articulaire illustré à la figure "1.2":

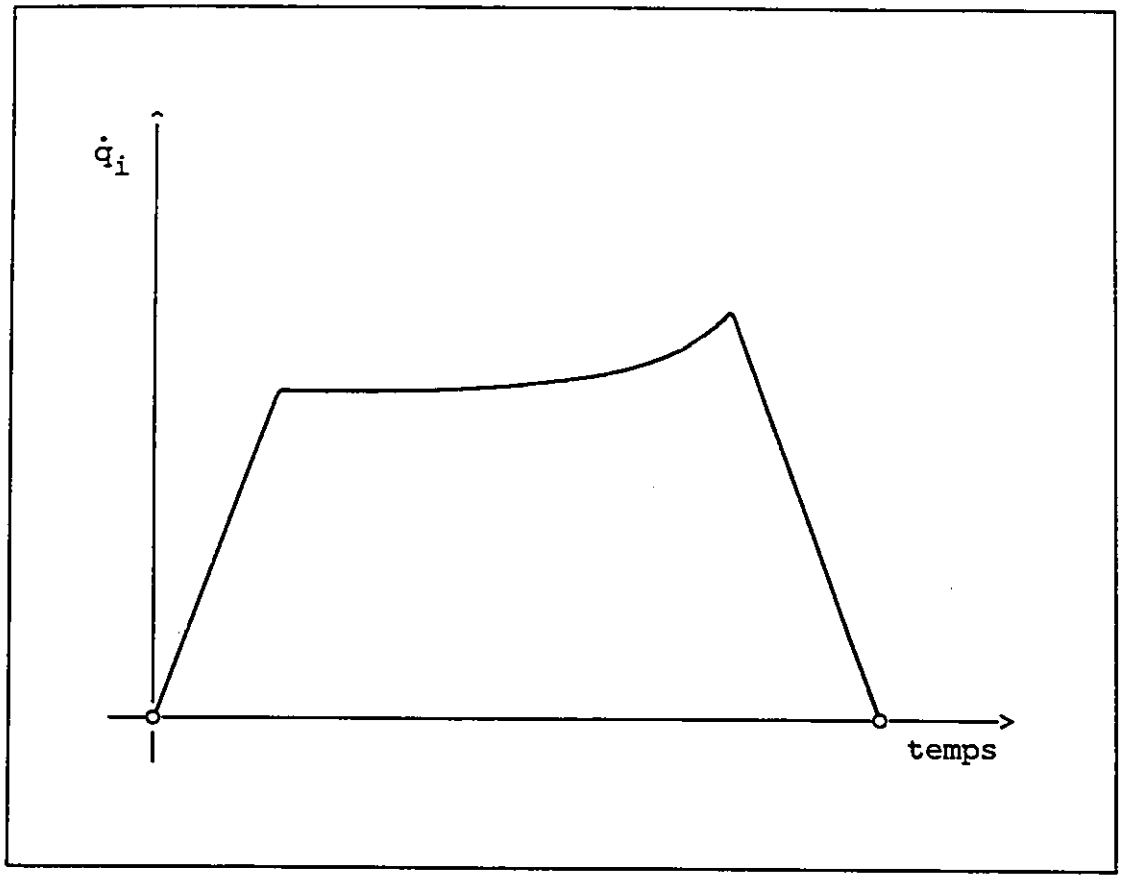


Fig.1.2 - Le profil de vitesse optimal.

Ce profil s'apparente à un mouvement constitué de deux transitions et d'un déplacement à vitesse constante. Les algorithmes basés sur l'asservissement optimal arrivent à des résultats semblables. Pour une grille de 15x15, un calcul hors-ligne de quelques heures est requis. Selon SAHAR et HOLLERBACH [57], le problème n'est pas simple à résoudre:

> "the complexity of the state-space search may currently rule out six-degree-of-freedom implementations, but an implementation on a three-joint mani-

pulator seems possible."[1]

En regardant la méthode, on se rend compte que la solution se fait dans le domaine articulaire. C'est ce qui permet de faciliter la solution. Par contre, elle souffre des mêmes problèmes que la théorie de l'asservissement optimal: cette approche ne satisfait pas la simplicité de solution et l'efficacité du calcul.

Pour les trajectoires présentées ci-haut, on se rend compte que deux facteurs influencent l'optimalité temporelle d'une trajectoire: le choix de la courbe cartésienne et la dynamique du robot. En fait, il existe d'autres facteurs qui influencent le temps. Par exemple, SCHEINMAN et ROTH [60] ont étudié l'effet sur le temps de la grosseur du manipulateur et de la position du robot par rapport à la courbe cartésienne désirée. Il est possible de choisir la grosseur et la géométrie du manipulateur qui optimise le temps. Pour ce faire, ils ont utilisé une linéarisation de la dynamique pour des robots avec deux degrés de liberté.

En utilisant des limites pour la pire configuration du robot, on obtient une solution sous-optimale basée sur des contraintes cinématiques seulement. Il est essentiel d'introduire les contraintes dynamiques lors de la planification de la trajectoire pour générer une trajectoire optimale et réalisable. Comme on l'a constaté, la théorie de l'asservissement optimal et l'optimisation

---

[1] SAHAR, Gideon, et John M. HOLLERBACH. "Planning of Minimum-Time Trajectories for Robot Arms", The International Journal of Robotics Research, vol.5, no. 3, automne 1986.

numérique conduisent à l'obtention d'une trajectoire optimale pour une performance donnée. Par contre, ces algorithmes mènent à une solution tellement compliquée qu'ils sont appliqués à des robots avec deux ou trois degrés de liberté. Pour un robot avec six degrés de liberté, il faut faire un compromis entre l'optimalité et la facilité de solution: c'est le but de l'algorithme présentée dans cette thèse.

CHAPITRE II

LES ASPECTS GENERAUX DE L'ALGORITHME DE CALCUL.

Le chapitre "2" aborde les concepts généraux de l'algorithme proposé. On étudiera les hypothèses posées et on définira le problème à résoudre. Puis, on présentera la structure générale de l'algorithme et les principes d'optimisation utilisés. Pour terminer, on discutera des problèmes rencontrés.

2.1 - Les hypothèses de calcul:

Généralement, les algorithmes de calcul reposent sur certaines hypothèses simplificatrices; le présent algorithme en utilise deux. En supposant des liens rigides, on peut négliger l'élasticité des liens. Cela facilite la modélisation cinématique et dynamique du manipulateur. Cette première hypothèse est très réaliste étant donné la conception actuelle des robots.

Pour la dernière hypothèse, on assume que l'amortissement (friction et viscosité) est négligeable. Généralement, ces effets de frottement sont faibles comparativement aux efforts mis en jeu.

2.2 - L'élaboration du problème à résoudre:

A partir des caractéristiques idéales de la trajectoire, on peut faire un certain nombre de choix quant à la façon de résoudre le problème de la planification de trajectoire. Cela est présenté à la section 2.2.1. Puis, à partir de ces choix, on

déterminera le problème à résoudre.

2.2.1 - La satisfaction des caractéristiques idéales:

Au chapitre "I", on a vu qu'il existe plusieurs caractéristiques idéales reliées à une trajectoire. Evidemment, une trajectoire réelle ne peut pas posséder toutes ces caractéristiques; chacune sera analysée de manière à faire ressortir son effet sur le développement de l'algorithme.

La planification de la trajectoire est un outil utilisé par un opérateur. Cette personne a généralement peu de connaissances en robotique et ne connaît pas le fonctionnement de l'algorithme. Pour faciliter le travail de l'utilisateur, il faut s'assurer que l'algorithme utilise une description peu compliquée de sa tâche. Une façon simple est de spécifier un certain nombre de points qui représentent les positions de travail ou qui servent à l'évitement d'obstacles. L'algorithme proposé utilise une représentation implicite de la tâche.

Cette représentation fait apparaître le problème de prévisibilité de la trajectoire à cause de l'utilisation de contraintes locales (les points) seulement. Ce problème est fréquemment rencontré dans la planification de trajectoire dans le domaine articulaire. L'utilisateur de l'algorithme doit avoir une idée de la courbe cartésienne qui sera générée avant même l'exécution de l'algorithme. Généralement, l'opérateur compare le mouvement de l'organe terminal au mouvement naturel d'un corps unique. Une façon d'atteindre cet objectif

est de mettre des contraintes géométriques de forme. Cela sera présenté à la section 2.4.1 sur l'optimisation géométrique.

La description implicite est souhaitable pour les opérations de "prend et place" et les déplacements rapides. Par contre, cette description est moins appropriée pour la génération d'une courbe cartésienne donnée. L'algorithme proposé permet l'approximation de la courbe cartésienne en utilisant un nombre élevé de points intermédiaires. Il est raisonnable de penser que l'utilisation de contraintes de formes, plus particulièrement l'orientation de la vitesse cartésienne aux points, réduit le nombre de points requis pour l'approximation d'une courbe cartésienne.

La planification de la trajectoire peut se faire dans un ou l'autre des espaces de travail. Le présent algorithme modélise les trajectoires dans l'espace articulaire pour satisfaire aisément les limites articulaires et pour obtenir les variables propres à l'asservissement (variables articulaires). De toute évidence, il est important de satisfaire les limites articulaires pour générer une trajectoire réaliste. En tenant compte de l'ensemble des limites cinématiques et dynamiques, l'efficacité de calcul de la trajectoire est diminuée. Comme on le sait, les points sont convertis dans le domaine articulaire en utilisant la cinématique inverse de position. Pour assurer la continuité du mouvement, on impose des contraintes de continuité. Le choix de l'espace articulaire permet aussi d'éviter les singularités mathémati-

ques puisqu'il n'y a pas une correspondance continue entre le domaine cartésien et le domaine articulaire.

Pour certaines applications industrielles, l'optimisation du temps s'avère très importante. Le choix de l'espace articulaire permet de simplifier l'optimisation puisque la dynamique contient des limites articulaires. Cela contribue donc à améliorer l'efficacité du calcul de la trajectoire. Pour certaines autres applications, il est nécessaire de coordonner le mouvement du robot avec d'autres machines. La formulation mathématique de l'algorithme proposé rend possible la spécification d'un temps pour chaque segment.

Comme on l'a déjà vu, une méthode globale de solution (MGS) est nécessaire, car l'asservissement optimal en temps réel est impossible. Tout comme LIN et al. [35], cette MGS se divise en deux parties: le calcul hors-ligne et le calcul en temps réel. Dans la première partie, on fait la planification et l'optimisation de la trajectoire. Dans la seconde, on fait la poursuite de la trajectoire. En utilisant une modélisation mathématique de l'espace articulaire, il est alors facile de satisfaire le taux d'asservissement. La MGS sera vue à la section 2.3.

Le calcul hors-ligne de la MGS restreint le champs d'application de l'algorithme proposé aux tâches décrites dans un environnement statique. Cela réduit la généralité de l'application de l'algorithme et ne satisfait pas l'efficacité de calcul. Il faut noter que ces inconvénients sont mineurs puisque l'automatisation se fait généralement dans un milieu

où la tâche est répétitive et l'environnement est statique. Se basant sur l'idée de KANT et ZUCKER [26], il est possible de tenir compte de l'environnement dynamique en faisant une pré-planification pour les aspects statiques et une modification de la trajectoire au moment de l'asservissement pour les aspects dynamiques.

2.2.2 - Le problème à résoudre:

De la discussion précédente, le problème à résoudre se décrit comme suit: connaissant la description implicite de la tâche, quel est la trajectoire nécessaire pour l'exécuter sous l'effet des limites articulaires? On cherche donc les positions, les vitesses et les accélérations articulaires qui génèrent le mouvement désiré pour une tâche donnée.

Il existe deux types de points qui doivent être donnés: les points fixes (positions de travail, d'arrêt ou d'attente) et les points libres (évitement d'objets). Seuls les points fixes sont connus de l'opérateur; l'emplacement des points libres étant sujet à l'optimisation du temps.

L'algorithme se base sur la satisfaction de contraintes cinématiques aux points de visite: continuité de la position, de la vitesse et de l'accélération. En utilisant l'inverse cinématique, on peut connaître les positions articulaires aux points. Pour les vitesses et les accélérations articulaires, le problème est un peu plus compliqué. Aux points fixes, les vitesses et/ou les accélérations sont nulles ou constantes. Par contre, aux points libres, les vitesses et les accéléra-

tions sont inconnues. Il faudra se donner une façon de les spécifier.

Les algorithmes des sections 1.3.2 et 1.3.3 permettent de trouver la trajectoire optimale par rapport au temps entre deux points quelconques. L'algorithme proposé est moins spécifique et tente de résoudre le problème complet de planification de la trajectoire. Pour corriger les inconvénients de ces algorithmes, on se donne une formulation mathématique simple qui permet le calcul pour des robots avec plus de trois degrés de liberté et on ajoute des contraintes de forme. De plus, l'algorithme proposé permet de relier ensemble plus de deux positions. À la position initiale et finale, la cinématique du robot est complètement déterminée puisque le robot est immobile. Pour le problème de cette thèse, l'existence de points intermédiaires complique la solution du problème puisque de nombreux inconnus de vitesses et d'accélérations doivent être trouvés.

Etant donné les nombreuses contraintes imposées sur la trajectoire, certaines approximations doivent être faites. Les deux principales approximations consistent à calculer les vitesses et les accélérations aux points intermédiaires à l'aide d'une heuristique (section 3.2.2); et à choisir le type de fonction analytique qui relie les points. Ces approximations conduisent à une solution sous-optimale par rapport au temps.

2.3 - La méthode globale de solution (MGS):

Dans cette section, on présente les deux parties de la MGS en élaborant les principes de la méthode. Le fonctionnement de cette MGS est schématisé à la figure "2.1".

2.3.1 - La planification de la trajectoire:

L'opérateur spécifie la tâche désirée en donnant les positions de travail (points fixes) et la localisation des objets à éviter dans l'enveloppe du robot. Cela constitue l'entrée de la planification de la trajectoire qui se divise en deux parties: la planification globale de la tâche (PGT) et la planification locale de la trajectoire (PLT).

La PGT reçoit la tâche désirée et elle vérifie s'il est nécessaire d'ajouter des points libres pour éviter les objets ou pour rendre la trajectoire plus prévisible. La position de ces points peut être optimisée pour minimiser globalement le temps et/ou pour améliorer la courbe cartésienne. Le problème d'évitement d'objet n'étant pas le sujet de cette thèse, on assume que ces points sont connus.

L'entrée de la PLT est calculée en faisant la cinématique inverse de position pour les points fixes et libres. Puis, la PLT consiste à relier les positions articulaires de la trajectoire en utilisant des segments de fonctions analytiques sous l'influence de contraintes géométriques et dynamiques. Pour ce faire, la PLT se divise en six étapes; comme le suggère la figure "2.2".

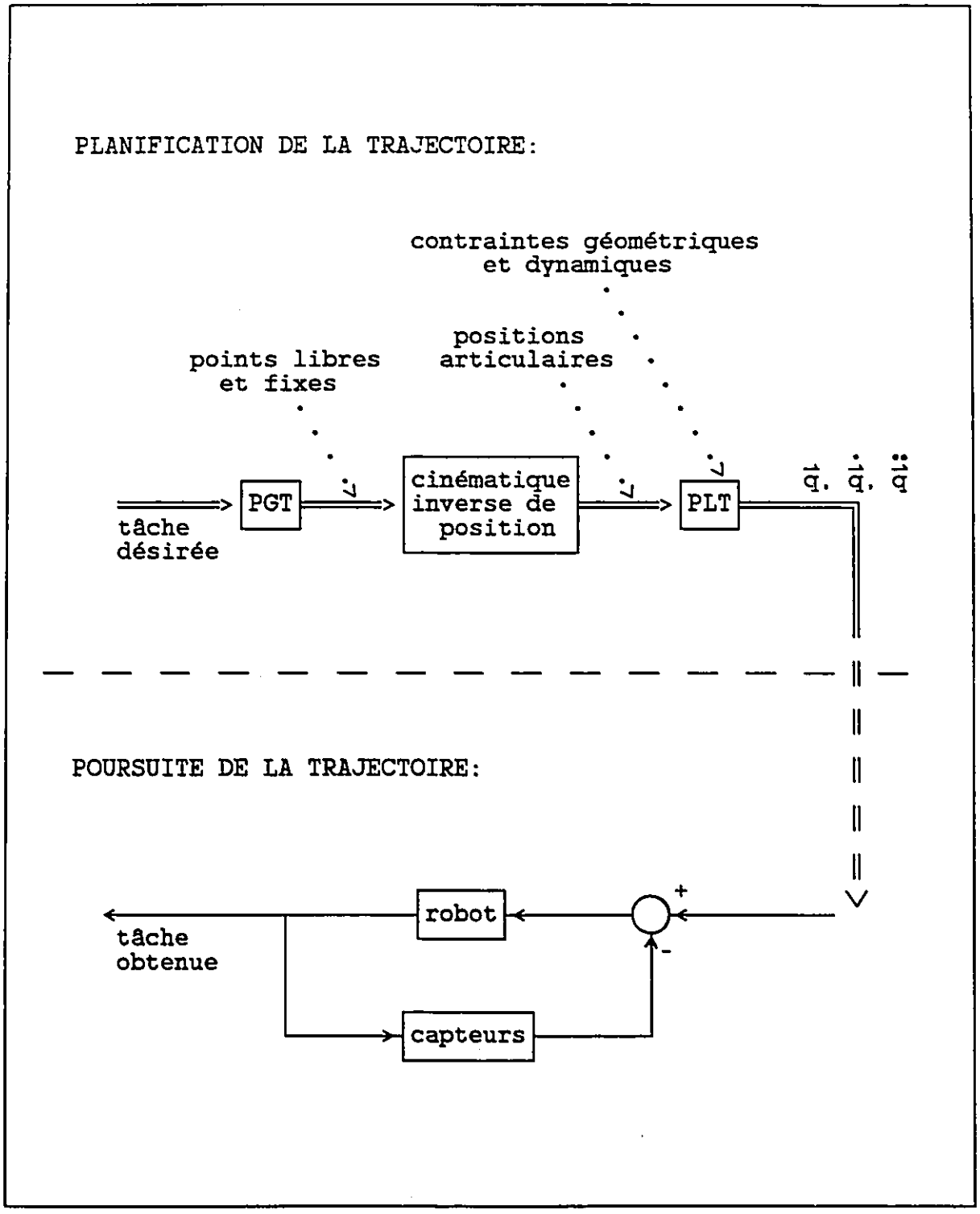


Fig. 1.1 - La méthode globale de solution.

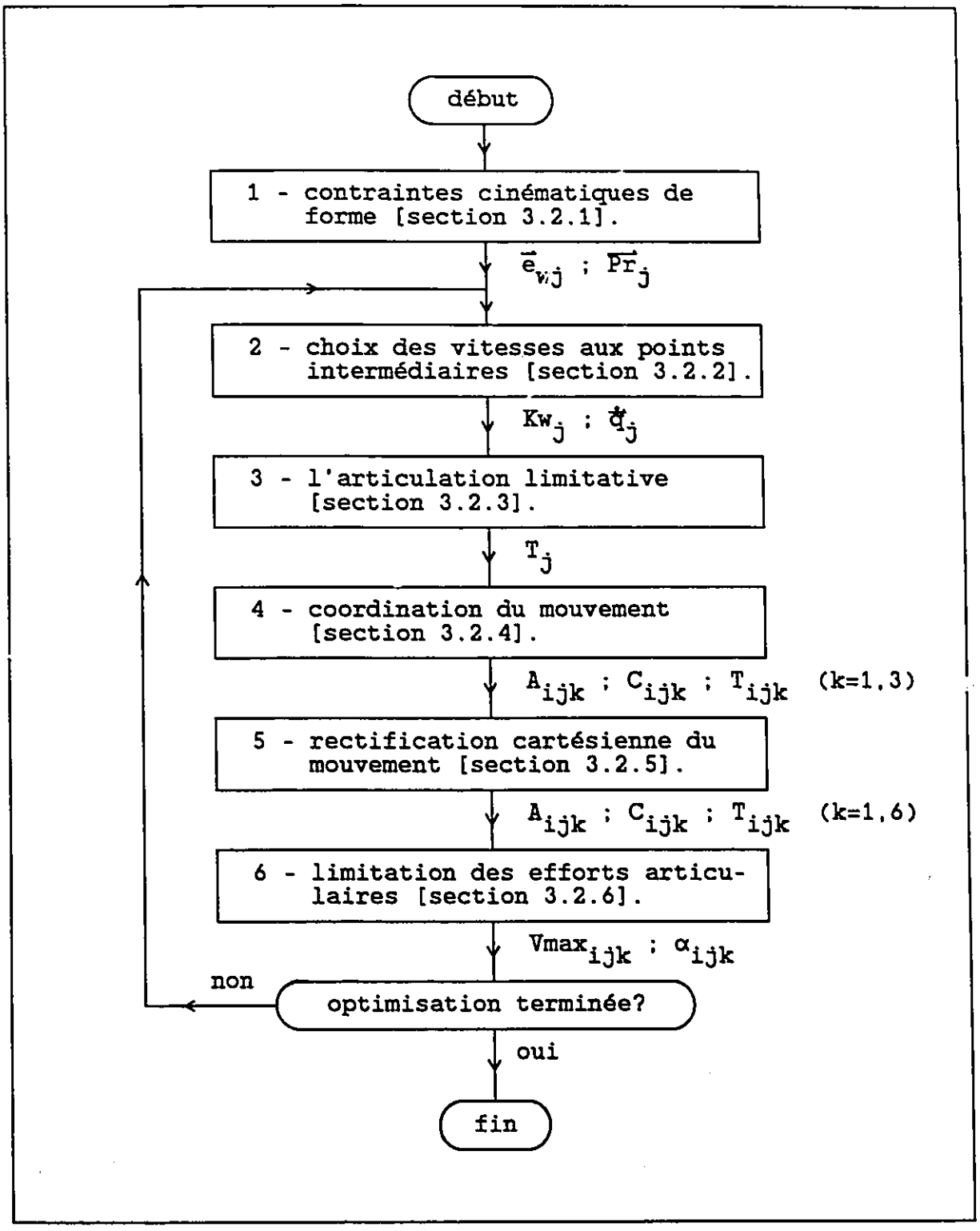


Fig.2.2 - La planification locale de la trajectoire.

L'étape "1" de l'algorithme calcule la position du point de rectification et permet de rendre la trajectoire plus prévisible en ajoutant une contrainte de forme. Cette contrainte consiste à imposer une orientation spécifique pour la vitesse cartésienne aux points intermédiaires. Puis, l'heuristique de l'étape "2" permet de trouver la grandeur des vitesses articulaires aux points intermédiaires. Ensuite, on calcule l'articulation limitative à l'étape "3" pour chaque segment de trajectoire. L'étape "4" permet de modifier le mouvement des autres liens pour coordonner le mouvement du manipulateur avec l'articulation limitative. A ce stade-ci, on calcule un point à la mi-temps de la courbe cartésienne générée. Si ce point est trop éloigné de la ligne qui relie le segment, alors l'étape "5" corrige le mouvement du manipulateur en ajoutant le pseudo-point de rectification calculé à l'étape "1". Cette étape "5" diminue la déviation cartésienne et n'affecte pas vraiment le temps. La dernière étape consiste à changer les paramètres d'optimisation (vitesses et accélérations articulaires maximales) sous l'effet de la dynamique. Le fondement mathématique de la PLT sera présenté au chapitre suivant.

2.3.2 - La poursuite de la trajectoire (temps réel):

La trajectoire articulaire sert d'entrée au système d'asservissement global du robot. Ce système doit commander les systèmes d'asservissement de chacun des moteurs (systèmes locaux) pour produire la trajectoire articulaire désirée.

2.4 - Les principes d'optimisation de la trajectoire:

La trajectoire ayant une composante de position et de temps, l'optimisation doit tenir compte de ces deux aspects. Les deux prochaines sections exposent les principes d'optimisation et leurs applications dans la PLT.

2.4.1 - L'optimisation géométrique:

Deux types de contraintes de forme sont imposées sur la trajectoire. La première consiste à choisir un ratio des vitesses articulaires de façon à obtenir une direction prévisible de la vitesse cartésienne de la main aux points intermédiaires. Ce critère s'avère inefficace pour des vitesses cartésiennes nulles ou faibles et n'a pas ou peu d'effet sur la prévisibilité de la courbe. Pour corriger ce problème, une contrainte de position est nécessaire. Avant de décrire cette seconde contrainte, on peut essayer de trouver une raison qui fait que la courbe est imprévisible. Soit un robot planaire qui déplace l'articulation "1" pendant que l'autre est au repos; ce qui génère un arc de cercle (figure "2.3"). Dans ce cas, un mouvement naturel (donc prévisible) de la main serait une ligne droite entre les deux points. Pour rectifier la courbe, il faudrait faire déplacer l'articulation "2" pendant le mouvement du lien limitatif "1". L'erreur sur la courbe cartésienne semblent provenir du non-mouvement et d'une mauvaise coordination des articulations non-limitatives. Dans l'exemple, l'articulation limitative "1" n'a pas besoin d'être

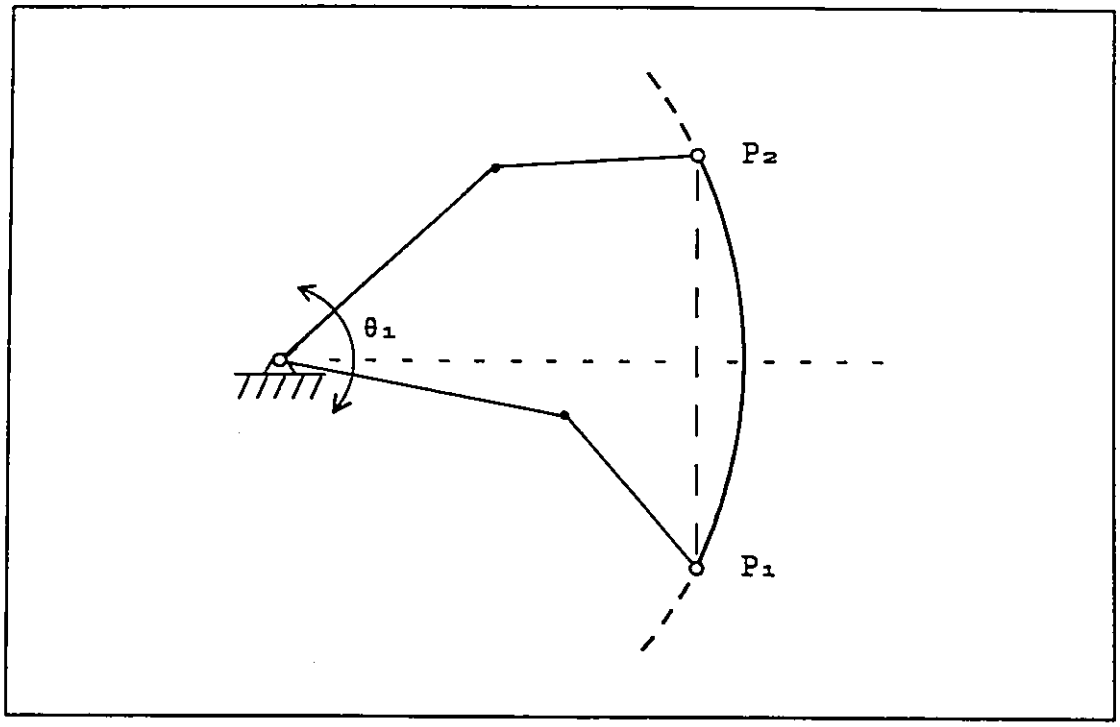


Fig.2.3 - Le mouvement symétrique d'un robot planaire.

modifié; mais pour l'autre, il faut la forcer à se déplacer pour corriger le mouvement cartésien. L'idée est donc d'ajouter un point pour modifier le mouvement des articulations non-limitatives seulement. Ces deux contraintes de forme sont calculées et appliquées aux étapes "1" et "5" de la planification locale de la trajectoire. Elles seront présentés à la section 3.2.1.

2.4.2 - L'optimisation du temps:

L'optimisation du temps se fait en utilisant la dynamique du manipulateur. Etant donné la nécessité de connaître la trajectoire pour calculer la dynamique, le processus d'optimisation du temps requiert une boucle d'itérations. Les

étapes "3 et 4" de la PLT permettent de faire le calcul de la trajectoire en utilisant des limites sur les vitesses et les accélérations articulaires. Ainsi, le problème global de minimisation du temps est ramené à un problème local d'optimisation puisque seulement les vitesses ou les accélérations maximales doivent être optimisées. De façon plus spécifique, l'étape "6" de la PLT vérifie et modifie ces limites articulaires pour la trajectoire générée aux étapes précédentes en utilisant la propriété de l'échelonnement dynamique du temps. Le processus d'itérations se termine lorsque la variation des limites articulaires est plus faible qu'une valeur donnée.

2.5 - Les problèmes rencontrés:

Plusieurs difficultés ont été rencontrées et les plus importantes sont traitées dans la présente section.

Le premier problème provient des vitesses articulaires inconnues aux points intermédiaires. Si le robot arrive à une vitesse très rapide au début d'un segment, alors le manipulateur peut générer un dépassement élevé pour atteindre la position désirée à la fin du segment. Par exemple, pour le dernier segment d'une trajectoire, le robot doit arrêter au point final. Si la vitesse initiale du segment est trop élevée, alors le robot ne sera pas capable d'arrêter à la position désirée et fera un dépassement pour revenir à sa position finale. Une façon d'éviter ce problème est de s'arrêter à chaque point pour avoir l'indépendance totale entre les segments de trajectoire, mais cela

est inacceptable dans un contexte d'optimisation du temps. Ce problème a été partiellement résolu par l'étape "2" de la PLT.

Le second problème a rapport avec la coordination du mouvement (étape 4): comment doit-on modifier le mouvement des articulations non-limitatives pour obtenir le mouvement uniforme du manipulateur? De fait, il existe généralement plusieurs façons d'augmenter le temps de ces articulations pour synchroniser le mouvement articulaire. Intuitivement, on peut s'imaginer qu'il existe toujours une solution lorsque les vitesses ou les accélérations articulaires sont réduites; mais cela n'est pas le cas. Pour certaines trajectoires, le manipulateur n'est pas capable de fournir les efforts nécessaires pour ralentir le mouvement de certaines articulations. On doit alors réduire les vitesses articulaires au début du segment.

Le dernier problème est la convergence de l'algorithme d'optimisation. Pour s'assurer de cela, il faut tout d'abord trouver l'articulation limitative (étape 3) en ignorant l'étape "5". Il faut noter que l'ajout d'un pseudo-point à l'étape "5" génère un segment divisé en six sous-segments pour les articulations non-limitatives; tandis que l'articulation limitative en possède trois. Même si la rectification peut changer l'articulation limitative à cause de la modification du mouvement, on gardera toujours la même articulation limitative pour éviter l'oscillation entre deux solutions.

Comme on a pu le constater, le nouvel algorithme respecte la majorité des caractéristiques d'une trajectoire idéale. En outre, il permet de rendre la trajectoire plus prévisible en ajoutant des contraintes géométriques de forme. L'optimisation par rapport au temps n'est pas optimale à cause de l'approximation de la trajectoire par une fonction analytique et à cause de l'heuristique choisi pour le calcul des vitesses articulaires aux points intermédiaires. Par contre, ces simplifications permettent d'augmenter l'efficacité de l'algorithme et d'appliquer l'algorithme aux robots avec plusieurs degrés de liberté.

CHAPITRE III

LA PLANIFICATION LOCALE DE LA TRAJECTOIRE (PLT)

La PLT permet de trouver les segments de trajectoire qui relient deux points. On présentera la façon de les modéliser en utilisant des multi-fonctions; et pour terminer, on exposera l'algorithme de la PLT.

3.1 - La modélisation mathématique des segments:

Un segment relie deux points consécutifs d'une trajectoire. L'expression de la fonction analytique optimale étant inconnue pour les segments, on assume un profil semblable à celui d'un robot planaire dans le domaine articulaire. On approxime le profil de la figure "1.2" par une multi-fonction à trois sous-segments (figure "3.1"): deux transitions et un mouvement à vitesse constante constituent le segment "j" d'une des articulations. La première transition permet d'établir la continuité avec le segment précédent et ajuste la vitesse articulaire pour le deuxième sous-segment à vitesse constante. La dernière transition anticipe le mouvement requis au segment suivant et modifie la vitesse articulaire pour générer une orientation désirée de la vitesse cartésienne.

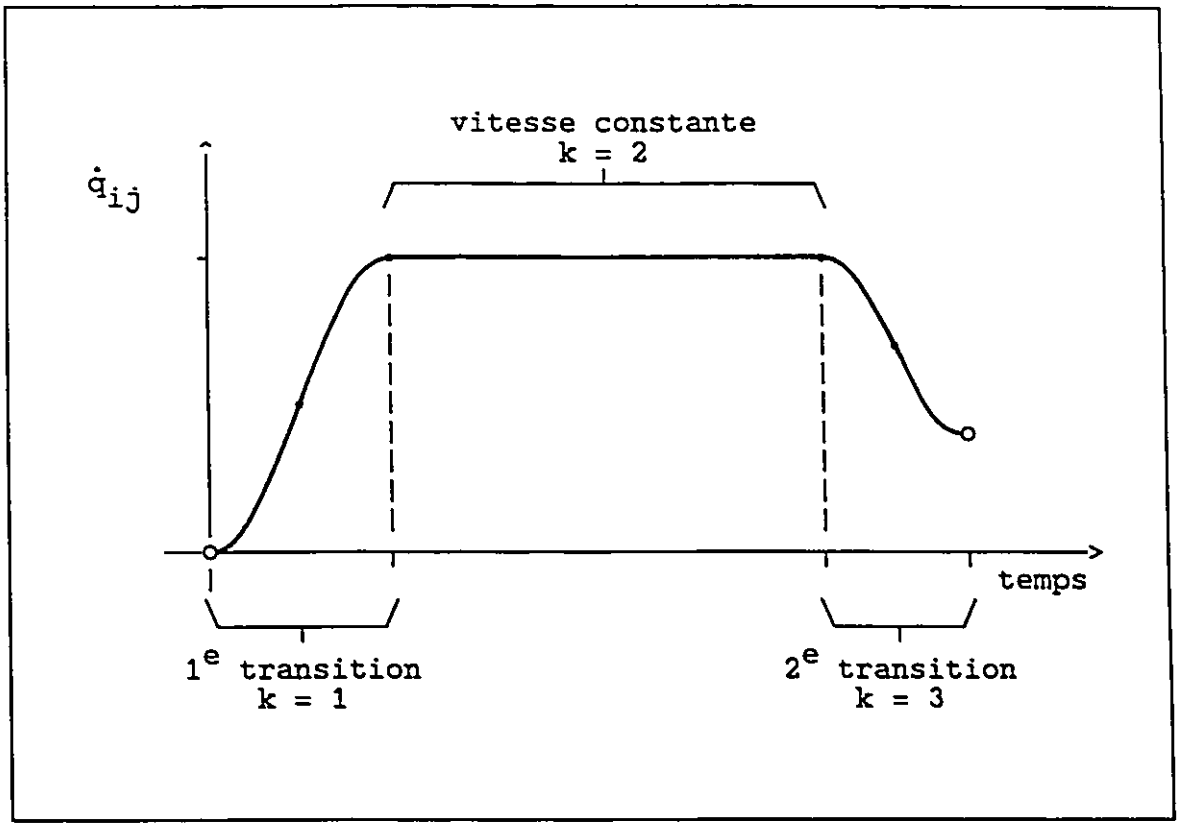


Fig.3.1 - Les sous-segments d'une trajectoire.

3.1.1 - Les transitions:

La modélisation mathématique des deux transitions est similaire; on fera surtout référence à la première transition, et cela facilitera la présentation de cette section.

La continuité du mouvement en position, en vitesse et en accélération est satisfaite en imposant six conditions aux bornes du sous-segment de transition. Sous forme mathématique, on obtient les contraintes de l'équation (3.1). À noter que l'intervalle de temps varie de "$-T_{ijk}$ à T_{ijk}" pour simplifier la solution.

$$
\begin{aligned}
q_{ijk}(-T_{ijk}) &= A_{ijk} = A & \qquad q_{ijk}(T_{ijk}) &= A_{ijk+1} = B \\
\dot{q}_{ijk}(-T_{ijk}) &= C_{ijk} = C & \qquad \dot{q}_{ijk}(T_{ijk}) &= C_{ijk+1} = D \\
\ddot{q}_{ijk}(-T_{ijk}) &= E_{ijk} = E & \qquad \ddot{q}_{ijk}(T_{ijk}) &= E_{ijk+1} = F
\end{aligned}
\qquad (3.1)
$$

$$(-T_{ijk} \leq t \leq T_{ijk})$$

Le problème consiste à relier ces conditions aux limites par une fonction analytique; on choisit les fonctions polynômiales, car elles sont plus simples à résoudre. Pour satisfaire les six contraintes, le polynôme du 5^e degré représente celui ayant le plus petit ordre qui puisse être utilisé. Le mouvement articulaire est alors donné par les équations suivantes:

$$
\begin{aligned}
q(t) &= a_5 \cdot t^5 + a_4 \cdot t^4 + a_3 \cdot t^3 + a_2 \cdot t^2 + a_1 \cdot t + a_0 \\
\dot{q}(t) &= 5 \cdot a_5 \cdot t^4 + 4 \cdot a_4 \cdot t^3 + 3 \cdot a_3 \cdot t^2 + 2 \cdot a_2 \cdot t + a_1 \\
\ddot{q}(t) &= 20 \cdot a_5 \cdot t^3 + 12 \cdot a_4 \cdot t^2 + 6 \cdot a_3 \cdot t + a_2
\end{aligned}
\qquad (3.2)
$$

Les coefficients dépendent des indices "i, j et k", tout comme "q(t)". Les indices ont été omis dans le but de simplifier la présentation. En utilisant les '6' conditions aux limites (3.1), on trouve les coefficients de l'équation (3.2). En posant $T = T_{ijk}$:

$$
\begin{aligned}
a_5 &= -\{3 \cdot (A-B) + T \cdot [3 \cdot (C+D) + T \cdot (E-F)]\} / (16 \cdot T^5) \\
a_4 &= [(C-D) + T \cdot (E+F) / (16 \cdot T^3) \\
a_3 &= \{5 \cdot (A-B) + T \cdot [5 \cdot (C+D) + T \cdot (E-F)]\} / (8 \cdot T^3)
\end{aligned}
\qquad (3.3)
$$

$$a_2 = -[3\cdot(C-D) + T\cdot(E+F)] \;/\; (8\cdot T)$$

$$a_1 = -\{15\cdot(A-B) + T\cdot[7\cdot(C+D) + T\cdot(E-F)]\} \;/\; (16\cdot T)$$

$$a_0 = \{8\cdot(A+B) + T\cdot[5\cdot(C-D) + T\cdot(E+F)]\} \;/\; 16$$

A noter que des coefficients de polynômes d'ordres différents sont présentées à l'appendice "A.1".

Habituellement, l'utilisateur connaît seulement les valeurs initiales de la position "A" et de la vitesse "C". Donc, il existe cinq inconnues dans les coefficients (3.3), il y a cinq inconnus: "B, D, E, F et T". Généralement, une transition côtoie un mouvement à vitesse constante; donc "E ou F" est nulle. Dans le but de simplifier la solution et de diminuer la dépendance d'un segment par rapport aux autres, on suppose que ces deux variables sont toujours nulles. Les autres variables sont déterminées en ajoutant d'autres contraintes sur la modélisation: la coordination du mouvement, la minimisation du temps local, la limitation de la vitesse maximale et la limitation de l'accélération maximale.

A - calcul de "D":

La vitesse finale "D" de l'articulation limitative est calculée de manière à minimiser le temps du segment et à respecter les limites imposées sur les vitesses et les accélérations. Cela est fait à l'étape "3" de la PLT. Pour les articulation non-limitatives, la vitesse "D" permet de coordonner le mouvement des articulations (étape "4" de la PLT).

B - calcul de "B et T":

Dans les transitions, l'accélération maximale articulaire "α_{ijk}" représente le paramètre à optimiser. Plus cette accélération est élevée, plus le temps de transition "T" est court puisque le temps est inversement proportionnel à l'accélération:

$$T_{ijk} \approx 1 / \alpha_{ijk} \qquad (3.4)$$

Le problème d'optimisation du temps de transition pour l'articulation limitative consiste, tout d'abord, à faire correspondre les extrémums de l'équation d'accélération (3.2) avec la limite permise par les moteurs. Cette équation possède, tout au plus, deux extrémums. Le changement de vitesse "D-C" se fait en un temps minimal seulement si l'accélération est toujours de même signe ou nulle. Cela implique que seulement un des deux extrémums doit être dans l'intervalle [-T,T]. Puis, le problème est complètement résolu en situant un des extrémums dans l'intervalle pour minimiser l'accélération maximale requise. D'un point de vue mathématique, on veut minimiser la valeur de l'extrémum en choisissant la position appropriée dans l'intervalle de temps [-T,T]. On peut montrer qu'il faut le positionner à "t = 0". Ce choix procure une accélération symmétrique par rapport au temps. Voici la présentation de cette preuve:

preuve:

Soit "A, C, D et T" données. A ce stade-ci, "B" est encore une variable inconnue. Puisque le temps "t" fait partie de l'intervalle [-T,T], on peut faire le changement de variable suivant:

$$t = n \cdot T \tag{3.5}$$

où $-1 \leq n \leq 1$

On cherche maintenant la valeur de "n", appelée "n^*", qui minimise l'accélération maximale. La valeur de l'accélération est donnée par l'équation (3.2):

$$\alpha_{ijk} = \ddot{q}_{ijk}(n^* \cdot T) \tag{3.6}$$

Pour minimiser l'accélération, il faut égaler la dérivée première de l'équation (3.6) à zéro:

$$\left.\frac{d\,\alpha_{ijk}}{d\,n}\right|_{n=n^*} = 0 \tag{3.7}$$

Dans l'équation (3.7), "B" est une variable inconnue; on trouve donc "n^*" en fonction de cette variable. En faisant correspondre le maximum d'accélération au temps "n^*", on obtient une seconde équation qui permet d'éliminer "B":

$$\dddot{q}(n^* \cdot T) = 0 \tag{3.8}$$

La subtitution de l'équation (3.8) dans l'équation (3.7) conduit au résultat suivant:

$$n^* = 0$$

De l'équation (3.5), on réalise que le maximum d'accélération doit être à "t=0" pour minimiser la valeur de ce maximum. De plus, on peut facilement démontrer que ce résultat minimise aussi le taux d'accélération (jerk); ce qui permet de réduire le niveau d'impact.

c.q.f.d

Des équations (3.6) et (3.8), on trouve maintenant les deux relations suivantes:

$$B = A + T \cdot (C+D) \qquad (3.9)$$

$$T = 3 \cdot (D-C) \; / \; (4 \cdot \alpha_{ijk}) \qquad (3.10)$$

Ces nouvelles équations permettent de simplifier l'expression des coefficients (3.3) aux valeurs suivantes:

$a_5 = a_3 = 0$ ----> procure la symétrie.

$a_4 = (C-D)/16 \cdot T^3$

$a_2 = \alpha_{ijk}/2$ ----> mi-accélération maximale.

$a_1 = (C+D)/2$ ----> la vitesse moyenne.

Le coefficient "a_5" étant nul, il existe seulement un maximum d'accélération; et l'ordre du polynôme se réduit à quatre degrés. Graphiquement, les équations (3.2) donnent les courbes de la figure "3.2".

En présentant différemment la solution de la modélisation des transitions, on peut facilement montrer qu'elle est identique à celle proposée par PAUL [50]. MUJTABA[2] a comparé différents types de modélisation d'un segment pour une articulation (robot avec un degré de liberté): bang-bang, polynôme (5^e degré), cosinus et amortissement critique. De toute évidence, la modélisation par bang-bang donne le temps minimum pour parcourir une distance donnée; les cosinus et les polynômes sont dix à vingt pour cent plus lents. La figure "3.3" montre la comparaison graphique des profils de vitesse entre ces différents types de modélisation.

---

[2] BRADY, M., et al. Basic of Robot Motion Planning and Control, Cambridge, Massachusetts, Etat-Unis, MIT press, p. 225.

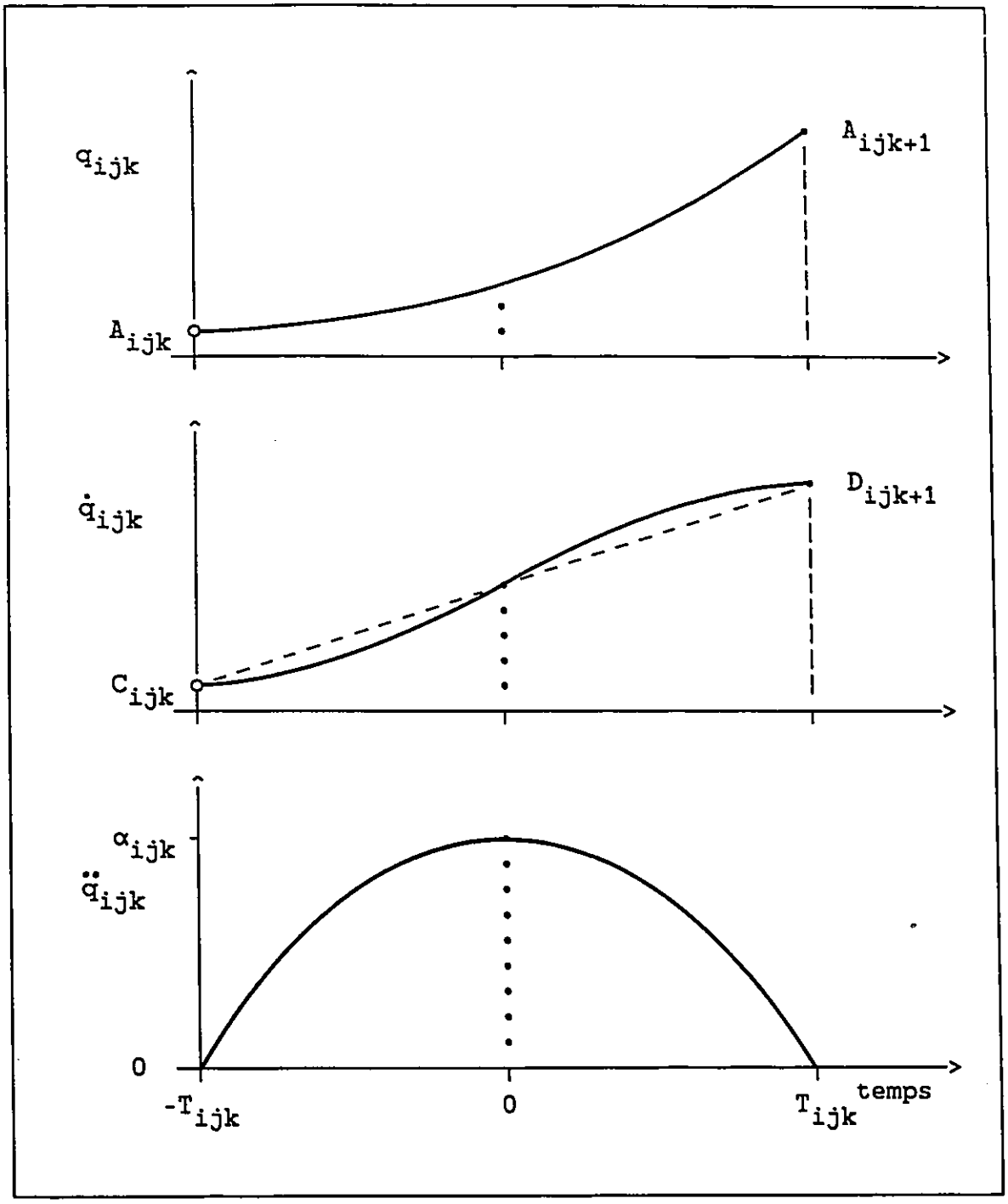


Fig.3.2 - La représentation graphique des transitions.

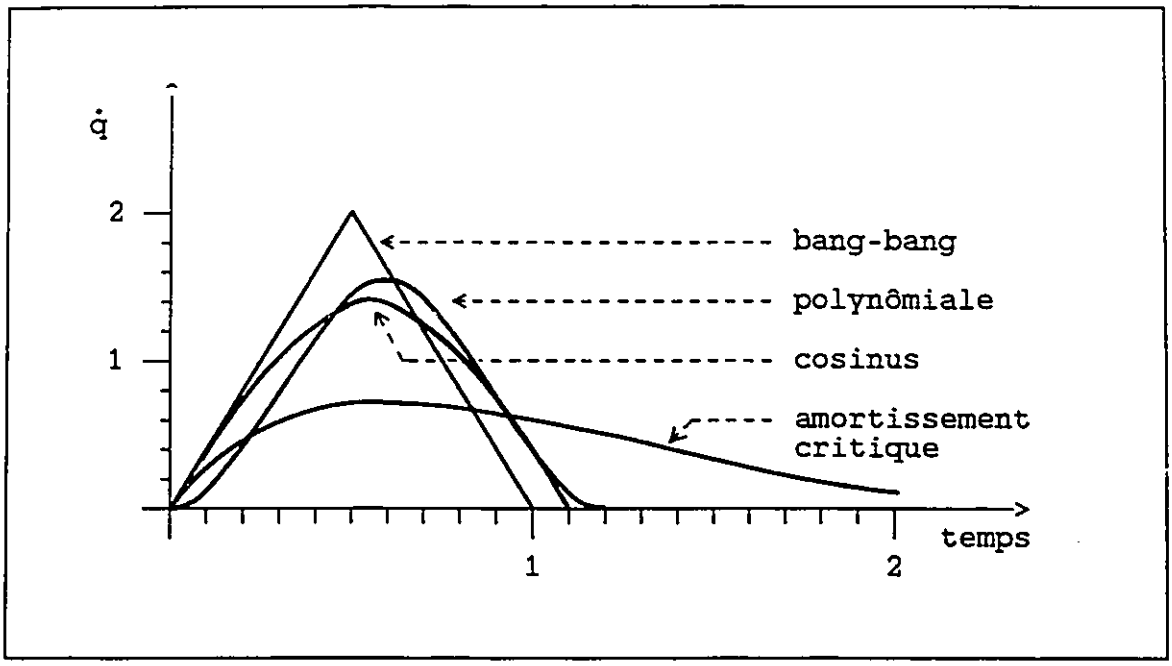


Fig.3.3 - différentes modélisations pour un segment (MUJTABA).

3.1.2 - La partie linéaire:

La modélisation de la partie linéaire se fait en utilisant un polynôme du premier degré sur l'intervalle de temps [-T,T]:

$$q(t) = a_1 \cdot t + a_o \qquad (3.11)$$

où,

$$a_1 = C$$

$$a_o = A + C \cdot T$$

3.2 - L'algorithme de la PLT:

L'algorithme de la PLT se présente en six étapes qui sont présentées dans les sections suivantes.

3.2.1 - Etape 1: les contraintes de forme.

Les contraintes de forme reposent sur la géométrie de la courbe cartésienne désirée; par conséquent, l'étape "1" doit être faite seulement une fois. Comme on l'a déjà mentionné, deux contraintes géométriques rendent la courbe cartésienne plus prévisible: l'orientation de la vitesse articulaire et l'ajout d'un point de rectification.

A - contrainte sur l'orientation de la vitesse cartésienne généralisée aux points intermédiaires:

Lorsque la vitesse est non-nulle aux points, un choix approprié de la vitesse articulaire produit une courbe cartésienne plus prévisible. On peut trouver les rapports de vitesses articulaires qui génèrent cette orientation en faisant la cinématique inverse pour une vitesse cartésienne unitaire $\vec{V}_j$ donnée. Cette vitesse est formée de la vitesse linéaire et de la vitesse d'orientation de la main:

$$\vec{V}_j = \begin{bmatrix} \vec{v}_j \\ \vec{w}_j \end{bmatrix} \qquad (3.12)$$

Soit l'orientation du segment "j" donnée par la ligne reliant le point "j" au point "j+1". "$\vec{V}_j$" est calculée en

choisissant l'orientation moyenne entre celle du segment "j-1" et celle du segment "j". Ce principe est illustré à la figure "3.4" pour le mouvement linéaire de la main "$\vec{v}_j$". Le même raisonnement s'applique pour la vitesse d'orientation de la main "$\vec{w}_j$".

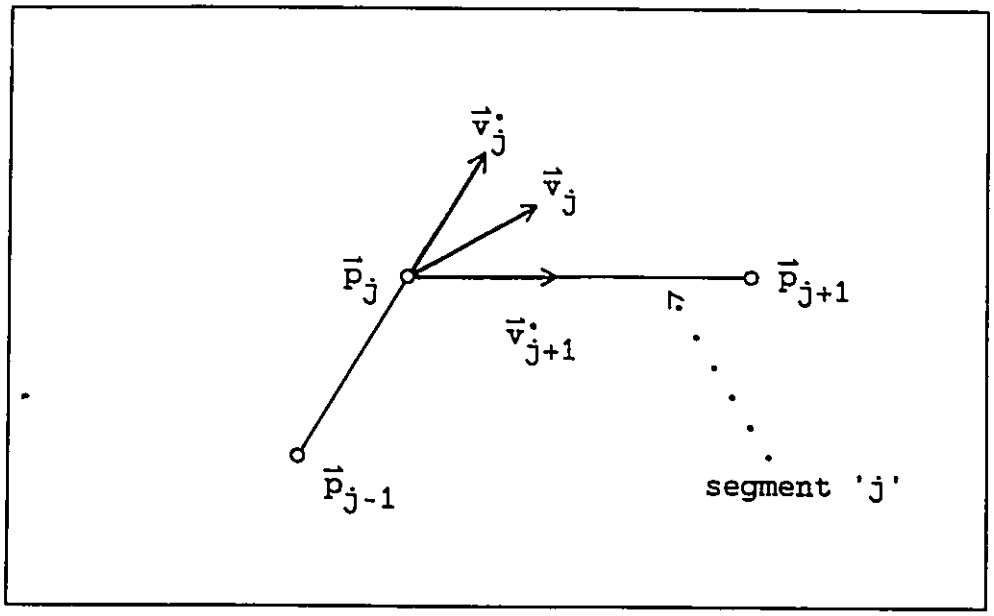


Fig.3.4 - L'orientation de la vitesse cartésienne linéaire.

De cette figure, on calcule "$\vec{v}_j$" en utilisant les positions cartésiennes des points:

$$\vec{v}_j = \vec{o} \qquad (j = 1)$$

$$= \frac{\vec{v}_j^{\cdot} + \vec{v}_{j+1}^{\cdot}}{\left| \vec{v}_j^{\cdot} + \vec{v}_{j+1}^{\cdot} \right|} \qquad (j = 2, m) \qquad (3.13)$$

$$\vec{v}_j = \frac{\vec{p}_j + \vec{p}_{j-1}}{\left| \vec{p}_j + \vec{p}_{j-1} \right|} \qquad (j \neq 1 \text{ et dénominateur} \neq 0)$$

$$= \vec{o} \qquad \text{(sinon)} \qquad (3.14)$$

Le calcul de "$\vec{w}_j$" se fait de la même façon en remplaçant "v" par "w" dans l'équation (3.13) et les "p" par "Φ" dans l'équation (3.14). Connaissant l'orientation de la vitesse cartésienne désirée "$\vec{V}_j$" par l'équation (3.12), on calcule la combinaison des vitesses articulaires en utilisant l'équation de la cinématique inverse de vitesse:

$$\vec{V}_{wj} = J^{-1} \cdot \vec{V}_j$$

$$\vec{e}_{wj} = \vec{V}_{wj} \,/\, \left| \vec{V}_{wj} \right| \qquad (3.15)$$

Pour les points singuliers, on fixe le vecteur vitesse articulaire unitaire "$\vec{e}_{wj}$" égale à zéro.

B - L'ajout d'un pseudo-point de rectification:

La rectification s'applique aux articulations non-limitatives; d'où son appellation de pseudo-point de rectification. Ce point permet de rectifier la courbe cartésienne si la déviation du point à la mi-temps de la courbe générée est trop élevée par rapport à un point d'une courbe cartésienne désirée. Ce point est calculé pour chaque segment et est ajouté à l'étape "5". A noter que la rectification de la position et de l'orientation cartésienne de la main se fait séparément. La position généralisée du

point de rectification est donné par:

$$\overrightarrow{Pr}_j = \begin{bmatrix} \overrightarrow{pr}_j \\ \overrightarrow{\Phi r}_j \end{bmatrix} \qquad (3.16)$$

Pour calculer la position de ce point, il faut se donner une "genre" de courbe cartésienne désirée; cette courbe doit avoir une forme prévisible par l'utilisateur. Le point de rectification est ensuite choisi au centre de cette courbe puisque le maximum de déviation est près du point milieu (TAYLOR [70]).

Ici, on présente seulement la solution pour la position cartésienne, car le calcul de "$\overrightarrow{\Phi r}_j$" est similaire à "$\overrightarrow{pr}_j$". En prenant un pourcentage "ϵ_j" de la longueur du segment "j" le long de l'orientation des segments "j-1" et "j+1", on définit deux nouvelles positions: "$\vec{p}^{\,\cdot}_j$" et "$\vec{p}^{\,\cdot}_{j+1}$". Cela est illustré sur la figure "3.5". Le point de rectification représente la position médiane de ces deux points. Ce qui donne les équations mathématiques suivantes:

$$\begin{aligned} d_r &= \left| \vec{p}_{j+1} - \vec{p}_j \right| \\ \vec{p}^{\,\cdot}_j &= \vec{p}_j + d_r \cdot \epsilon_j \cdot \vec{v}_j \\ \vec{p}^{\,\cdot}_{j+1} &= \vec{p}_{j+1} - d_r \cdot \epsilon_j \cdot \vec{v}_{j+1} \\ \overrightarrow{pr}_j &= \frac{\vec{p}^{\,\cdot}_j + \vec{p}^{\,\cdot}_{j+1}}{2} \end{aligned} \qquad (3.17)$$

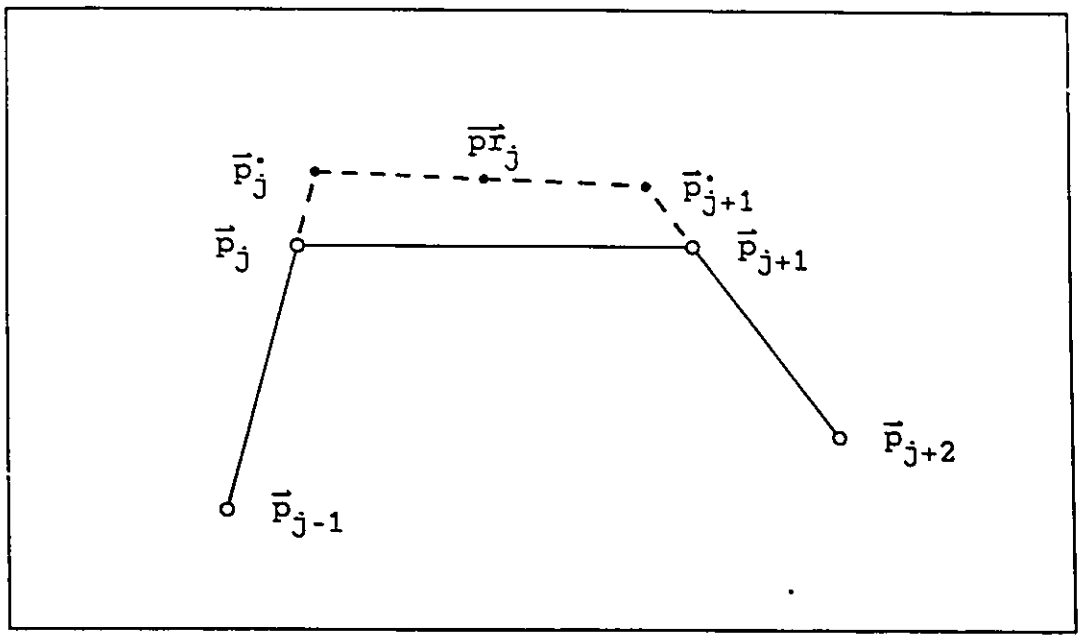


Fig.3.5 - Le point de rectification.

On appelle "ϵ_j" le facteur de forme. Plus ce facteur est petit, plus la distance minimale entre le point de rectification et la droite reliant les points du segment "j" est petite.

3.2.2 - Etape 2: la vitesse aux points.

Cette étape fait partie de l'optimisation géométrique et elle utilise la première contrainte de forme déterminée à l'étape précédente. Le choix des vitesses articulaires aux points intermédiaires a beaucoup d'effet sur l'ensemble de la trajectoire; cela affecte la courbe cartésienne et le temps. Dans les mouvements linéaires du manipulateur, les vitesses articulaires aux points doivent être nulles pour obtenir des dépassements nuls; pour les autres types de trajectoires, la

vitesse est limitée par la forme de la courbe cartésienne désirée et la dynamique du manipulateur.

Basé sur une discussion de CRAIG [8], il existe trois façons de spécifier les vitesses articulaires aux points intermédiaires: donner la vitesse cartésienne généralisée, choisir des vitesses articulaires nulles ou appliquer une heuristique. Dans la première approche, l'utilisateur donne la vitesse cartésienne généralisée; ce qui complique la description de la tâche. Ces vitesses cartésiennes sont ensuite ramenées dans l'espace articulaire en faisant la cinématique inverse de vitesse. La seconde approche est une solution triviale qui fonctionne pour toutes les configurations du robot, mais elle élimine la possibilité de rectifier la courbe cartésienne en utilisant une combinaison appropriée des vitesses articulaires. Dans la dernière approche, l'algorithme choisi automatiquement les vitesses en applicant une heuristique dans l'espace cartésien ou articulaire.

Dans cette thèse, on a choisi la troisième approche; car elle permet l'application simple de la première contrainte de forme géométrique. L'orientation du vecteur vitesses articulaires constitue le premier élément de l'heuristique; ce qui a été calculé à l'étape "1". Maintenant, seule la norme "Kw_j" de ce vecteur est inconnue. Les vitesses articulaires aux points sont donc données par l'équation suivante:

$$\dot{\vec{q}}_j = Kw_j \cdot \vec{e}_{wj} \qquad (3.18)$$

Cette grandeur est calculée en utilisant l'heuristique suivante. Pour les deux segments "j-1 et j" qui côtoient le point "j", on calcule le plus court temps théorique de chaque articulation en assumant un mouvement articulaire à vitesse constante:

$$T^{*}_{ij} = \left| \frac{A_{ij+1,1} - A_{ij1}}{Vmax_{ij2}} \right| \qquad (i=1,n)$$

Ce qui permet de trouver le temps le plus long et le vecteur vitesses articulaires constantes "$\dot{\vec{x}}_j$" associées aux deux segments:

$$Tmax_j = MAX \left[T^{*}_{ij} \right]_{i=1}^{i=n}$$

$$\dot{\vec{x}}_{ij} = \frac{A_{ij+1,1} - A_{ij1}}{Tmax_j} \qquad \begin{matrix} (i = 1, n) \\ (j \text{ et } j-1) \end{matrix}$$

En minimisant l'erreur au carré entre la vitesse de l'équation (3.18) et les vitesses articulaires "$\dot{\vec{x}}$", on obtient le résultat suivant:

$$kw_j = \frac{\sum_{i=1}^{n} \left[e_{wi} \cdot (\dot{x}_{ij} + \dot{x}_{ij-1}) \right]}{2} \qquad (3.19)$$

Ce qui revient à faire la somme des vitesses moyennes de chaque articulation décomposée selon la direction "e_{wi}". Il faut noter que la présente thèse n'étudie pas en profondeur le

choix approprié de la grandeur des vitesses articulaires aux points intermédiaires.

3.2.3 - Etape 3: l'articulation limitative.

Chaque articulation a besoin d'un temps minimum pour faire le mouvement. De par la modélisation mathématique des segments, il existe quatres types de solutions possibles qui permettent de modéliser un segment "j"; comme illustrées à la figure "3.6".

Le problème est résolu en établissant la logique nécessaire qui mène au bon type de solution. En utilisant la modélisation mathématique précédemment développée, on calcule le temps minimum associé à chaque articulation en respectant les limites physiques du robot. Ce temps résulte de l'addition de celui des trois sous-segments:

$$Tmin_{ij} = 2 \cdot \sum_{k=1}^{3} T_{ijk} \qquad (3.20)$$

Puis, le temps du segment "T_j" correspond au plus grand de ces temps:

$$T_j = MAX \left[Tt_{ij} \right]_1^n \qquad (3.21)$$

Le lien ayant ce temps correspond à l'articulation limitative.

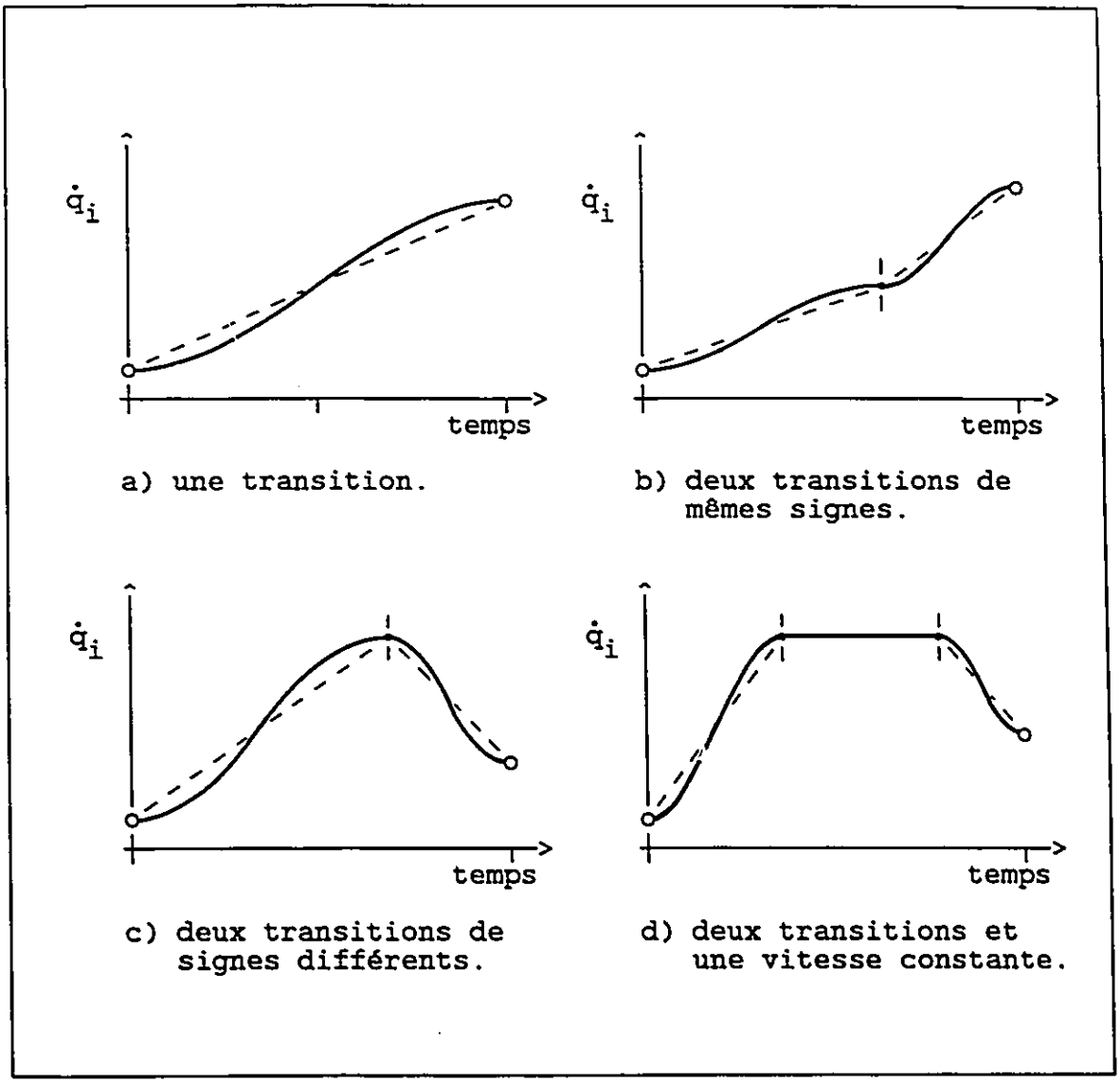


Fig.3.6 - Les solutions pour le lien limitatif.

3.2.4 - Etape 4: la coordination du mouvement.

Maintenant, on désire coordonner le mouvement du manipulateur en augmentant le temps des articulations non-limitatives. Il existe deux façons de le faire: réduire les vitesses ou les accélérations articulaires. La première façon

augmente le temps passé durant le segment à vitesse constante; la seconde, le temps passé durant les transitions. D'un point de vue dynamique, il est plus intéressant d'utiliser la seconde option car elle diminue le couplage dynamique sur le lien limitatif. De fait, les accélérations ont plus d'effet sur les efforts articulaires que les vitesses.

La modélisation des segments se fait en utilisant les mêmes types de solutions (figure "3.6") que dans l'étape précédente. Il suffit d'établir la logique nécessaire pour choisir le bon type de solution en respectant les limites physiques du robot et le temps alloué pour le segment.

La logique associée à l'étape "3" est beaucoup plus simple comparativement à l'étape "4". A l'étape "3", on connaît les valeurs des vitesses et des accélérations dans les équations; elles correspondent aux valeurs maximales. Cela permet de calculer facilement le temps associé à chacune des articulations. Par contre, la coordination du mouvement demande une réduction des accélérations pour les deux transitions à des valeurs inconnues dans un temps donné.

3.2.5 - Etape 5: l'ajout du point de rectification.

Cette étape permet d'appliquer la seconde contrainte de forme: l'ajout d'un point de rectification. Cette étude se fait en considérant séparément la position et l'orientation cartésienne de la main. Etant donné la similitude entre la position et l'orientation, on réfèrera seulement à la position cartésienne de la main.

Tout d'abord, il faut se demander où est situé le point de rectification dans l'espace temps; la seule limite étant l'intervalle de temps $[0,T_j]$. A l'étape "1", on a positionné ce point près du centre du segment "j"; ce qui correspond approximativement au temps "$T_j/2$". Il faut noter que la localisation précise de ce point n'est pas nécessaire puiqu'il sert seulement à forcer le mouvement des articulations non-limitatives.

Pour vérifier la nécessité d'ajouter un point, on se donne un critère: comparer la position cartésienne obtenue au temps "$T_j/2$" avec la position du point de rectification.

Au temps "$T_j/2$" de la trajectoire obtenue à l'étape "4", la main possède une position cartésienne pouvant être calculée par la cinématique directe. Pour chacun des deux points (rectification et obtenu), on calcule la distance minimale les séparant de la droite qui relie le point "j" au point "j+1"; ce qui donne la distance obtenue et la distance de rectification. Si la distance obtenue est supérieure à celle de rectification, alors on ajoute le point.

Le point de rectification sépare le segment "j" en deux nouveaux segments de temps "$T_j/2$": $T1_j$ et $T2_j$. Les deux segments ainsi créés sont modélisés par le même type de solution qu'auparavant; soit un segment composé de 3 sous-segments. Le segment "j" possède maintenant six sous-segments (figure "3.7").

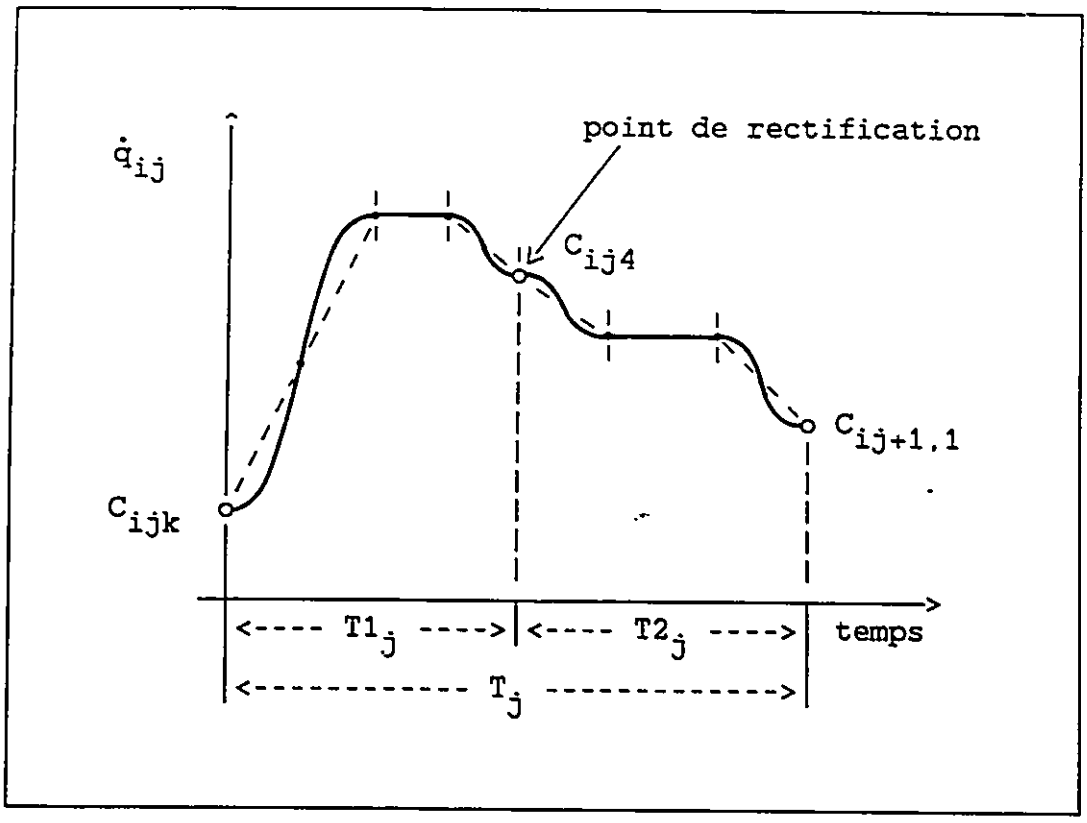


Fig.3.7 - Un segment constitué de six sous-segments.

Il existe un dernier inconnu: la vitesse "C_{ij4}" à ce point. De fait, on cherche la combinaison de vitesses articulaires qui aide à rectifier le mouvement et qui est réalisable dynamiquement. Une première façon consiste à calculer les vitesses articulaires en faisant la cinématique inverse d'une orientation cartésienne donnée. Cette approche doit être rejetée étant donné les deux inconvénients suivants: la grandeur de la vitesse cartésienne inconnue et la possibilité d'une singularité mathématique au point de rectification. Il serait aussi difficile de donner une solution dynamiquement faisable. Ces problèmes sont évités en gardant la même com-

binaison de vitesses articulaires qu'on avait au temps "$T_j/2$" à l'étape 4. Cela permet de s'assurer que cette combinaison de vitesses articulaires est possible d'un point de vue dynamique. Ces combinaisons de vitesses articulaires améliorent aussi la rectification de la courbe cartésienne en imposant une contrainte sur l'orientation de la vitesse.

Se basant sur des courbes cartésiennes obtenues de l'étape "4", on s'est aperçu qu'elles avaient une courbure dans un seul sens. Pour un segment "j", ces courbes ressemblent à celle de la figure "3.8". Cette courbe indique qu'il serait intéressant de garder la même orientation de vitesse cartésienne au centre de la courbe. Au temps "$T_j/2$", le vecteur vitesse cartésienne "$\vec{v}_j$" obtenu (de l'étape "4") correspond approximativement à celle obtenue au point milieu. Donc, si l'on garde la même combinaison de vitesses articulaires que pour la trajectoire obtenue à l'étape "4", alors la vitesse cartésienne ainsi générée au point de rectification aura une orientation semblable à "$\vec{v}_j$".

Il faut noter que le point de rectification a été trouvé en considérant des critères géométriques seulement. Donc, il est peut-être inaccessible dynamiquement. Si cela se produit, on fera du mieux qu'on peut: l'important, ce n'est pas de passer par le point de rectification mais de tendre vers ce point. La rectification doit donc faire partie du processus d'optimisation, car elle dépend de la dynamique du manipulateur.

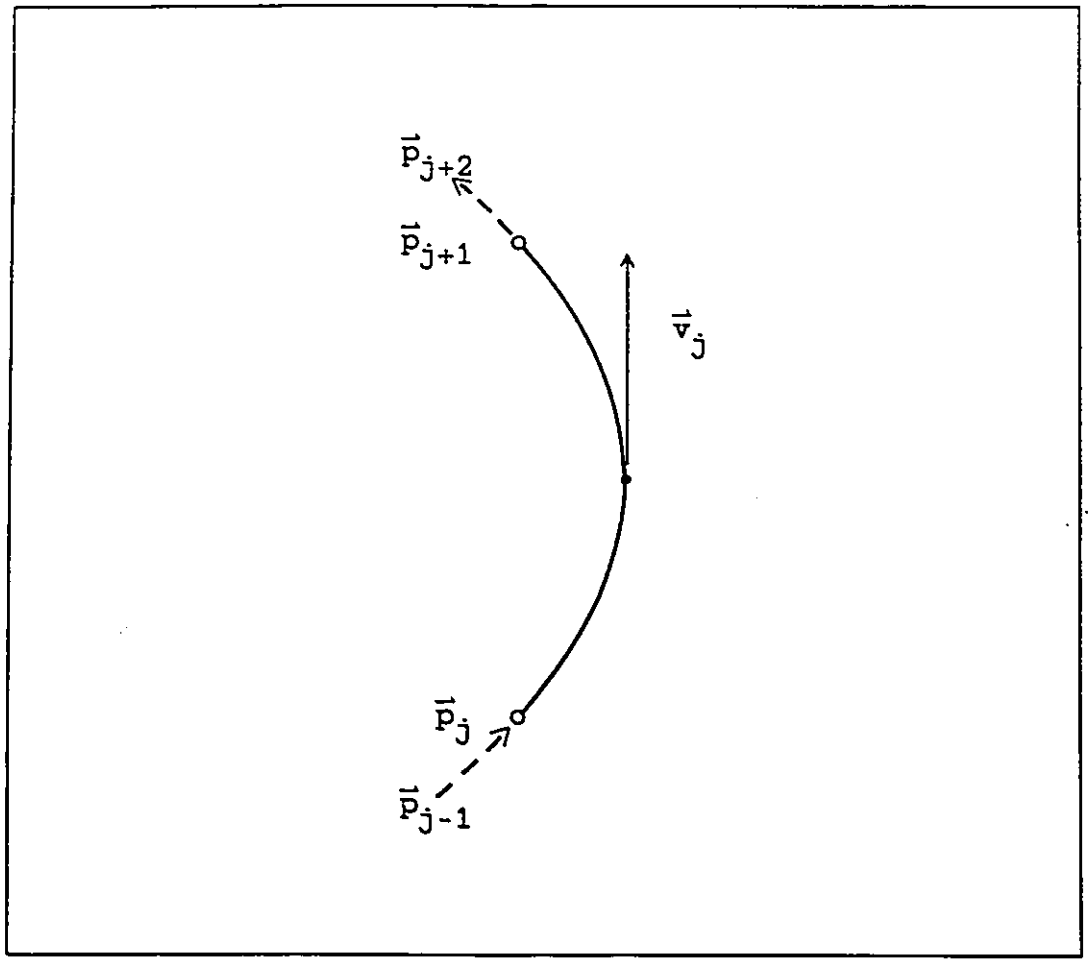


Fig.3.8 - Courbe cartésienne typique pour un segment "j".

3.2.6 - Etape 6: la contrainte dynamique.

Puisque la planification de la trajectoire est naturellement reliée au mouvement articulaire, il est préférable de convertir les limites d'efforts en limites de vitesses et d'accélérations articulaires. La propriété d'échelonnement dynamique du temps développée par HOLLERBACH [22] (annexe "B.3") facilite ce changement et permet de modifier linéairement l'échelle du temps d'une trajectoire existante:

$$t^* = t / c \qquad (3.22)$$

où l'intervalle de "c" sera donné par $[c^-, c^+]$.

Cet intervalle est calculé en tenant compte des limites des efforts articulaires. Le temps est minimisé en choisissant la limite supérieure "c^+".

Pour trouver cette valeur de "c", la dynamique de la trajectoire est vérifiée à intervalle de temps régulier "t" en suivant l'organigramme présenté à la figure "3.9". On calcule, tout d'abord, les moments de torsion des articulations par la méthode de Newton-Euler en séparant la gravité "G(t)" des autres termes [équation (B.1) de l'annexe "B.3"]. Cela permet de modifier les limites d'effort articulaire en utilisant l'équation (B.7). Puis, la variable d'échelonnement est calculée à l'aide du tableau "A.1" (annexe "B.3").

La trajectoire est modélisée avec deux types de mouvement: les transitions et les vitesses constantes. Les transitions demandent l'optimisation de la limite d'accélération maximale tandis que la partie linéaire requière l'optimisation de la limite de vitesse. De l'équation (3.10), on peut isoler l'accélération maximale:

$$\alpha_{ijk} = \frac{3 \cdot (D-C)}{4 \cdot T}$$

La nouvelle "α_{ijk}" est obtenue en divisant le temps par "c^+":

$$\alpha_{ijk} = \frac{3 \cdot (D-C)}{4 \cdot T/c^+} \qquad (3.23)$$

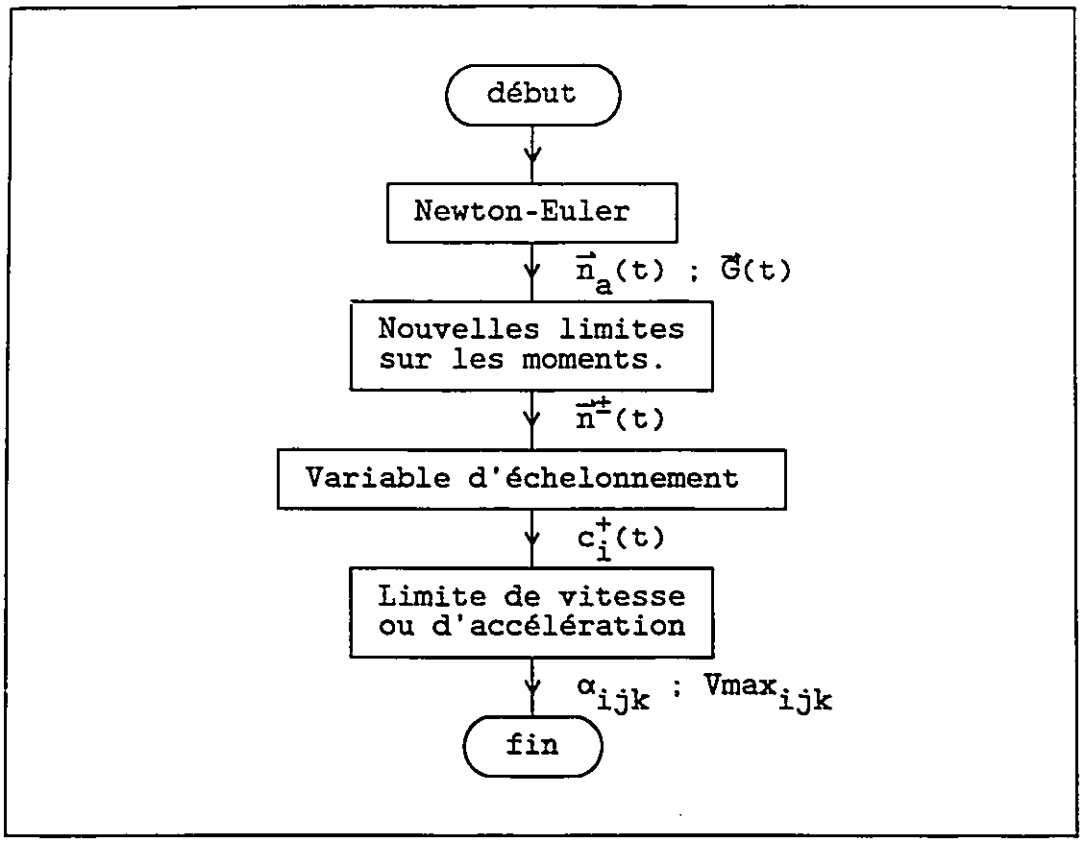


Fig.3.9 - L'introduction de la contrainte dynamique.

Pour la partie linéaire de la trajectoire calculée à l'étape "4", la vitesse constante peut être donnée par:

$$C_{ij2} = \frac{\text{distance}}{\text{temps}} = \frac{A_{ij3} - A_{ij2}}{2 \cdot T_{ij2}} \qquad (3.24)$$

La nouvelle limite sur la vitesse maximale est calculée en divisant le temps par "c^+":

$$Vmax_{ij2} = \frac{\text{distance}}{\text{temps}/c^+} \qquad (3.25)$$

CHAPITRE IV

LA SIMULATION NUMERIQUE

L'algorithme proposé a été utilisé pour simuler numériquement et graphiquement plusieurs tâches; une seule d'entre elles sera présentée à la section 4.1. Cette tâche a été accomplie par le modèle d'un robot ASEA IRB/6 (appendices "B à D") à six degrés de liberté; cela permet une simulation plus réaliste des trajectoires. La programmation de l'algorithme et la modélisation du robot en langage fortran (appendice "E") demande 285 K-octets de programme et de données; ce qui permet l'utilisation d'un micro-ordinateur.

Les performances de l'algorithme proposé seront comparées aux résultats de L'ILA. L'ILA a été choisie pour plusieurs raisons: sa simplicité de mise en oeuvre, sa modélisation mathématique comparable au nouvel algorithme et son utilisation fréquente par les intervenants du milieu. De fait, l'ILA est souvent considérée comme un standard de comparaison. Cela permet d'étudier les temps et les courbes obtenus par les deux algorithmes. Les résultats des deux algorithmes seront présentés à la section 4.2.

4.1 - La présentation de la tâche simulée:

Les deux algorithmes utilisent une représentation implicite de la tâche; la trajectoire est donc décrite par une suite de points donnés dans l'espace cartésien. La tâche considérée possède quatre points.

On désire usiner une pièce sur une machine-outil (point "3"). Pour se faire, le robot doit prendre une pièce au point

"1" et la déposer sur la machine-outil. Le point "2" est ajouté pour modifier l'approche de la main du robot et ainsi éviter la collision avec la machine. Une fois la pièce terminée, elle est déposée dans un magasin au point "4". Dans l'espace cartésien, cela équivaut aux points du tableau "4.1" et de la figure "4.1".

TABLEAU 4.1

LES QUATRES POINTS DE LA TACHE SIMULEE.

point #	p_x (m.)	p_y (m.)	p_z (m.)	Φ_x (deg.)	Φ_y (deg.)	Φ_z (deg.)
1	0.72	0.00	0.85	0.	135.0	0.
2	0.80	0.00	1.74	0.	79.5	0.
3	1.02	0.00	1.22	0.	135.0	0.
4	0.00	1.02	1.22	0.	135.0	90.

Les trois premiers points sont dans le plan "X-Z" tandis que le point "4" est dans le plan "Y-Z". La hauteur des points "3 et 4" est la même. Le système d'axes montré à la figure "4.1" représente le système de base du robot. Le plan "X-Y" représente la surface horizontale.

Cette tâche a été simulée sur un micro-ordinateur IBM-386 compatible qui possède une horloge à 20 MHz et un processeur mathématique 80387. Les résultats obtenus sont présentés à la section suivante.

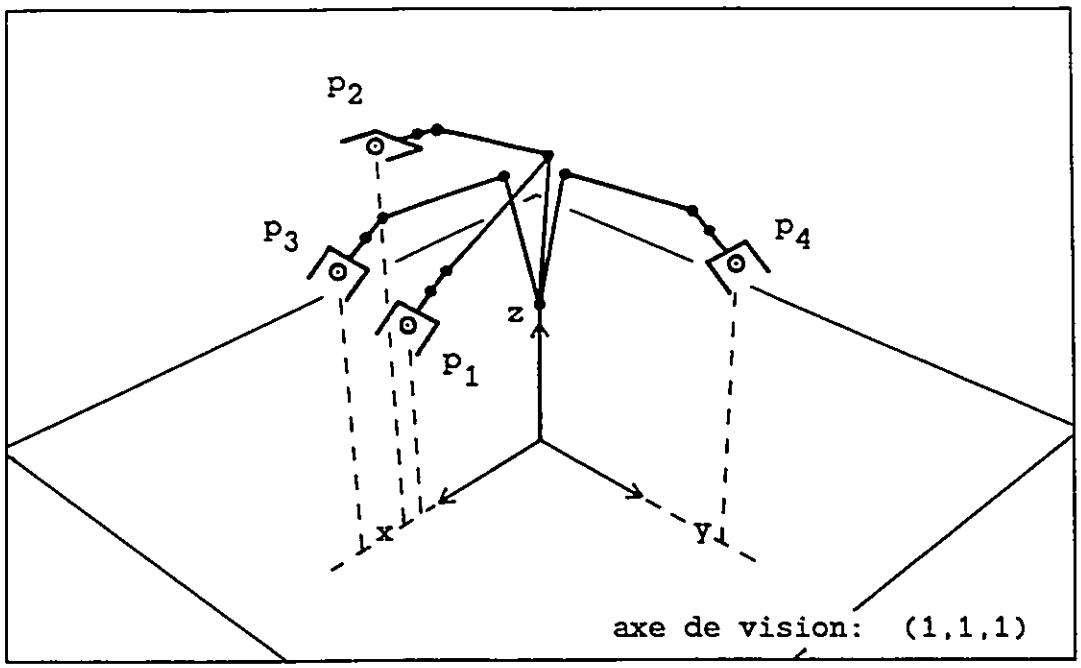


Fig.4.1 - Les quatres points de la tâche à accomplir.

4.2 - Les résultats:

4.2.1 - L'interpolation linéaire articulaire (ILA):

L'ILA nécessite la connaissance de certains paramètres: le temps de transition "T_{acc}" et les vitesses maximales articulaires. Généralement, ces valeurs sont déterminées expérimentalement à partir de la pire configuration du robot. Ces valeurs étant inaccessibles pour le robot ASEA, elles ont été approximées à partir des résultats de l'algorithme proposé; ce qui permet une comparaison plus réaliste. "T_{acc}" est choisi en prenant le plus court temps de transition possible pour la tâche considérée. En fait, le "T_{acc}" utilisé dans la simulation de l'ILA est inférieur à celui de la pire con-

figuration. Une discussion semblable s'applique pour les vitesses maximales. Donc, les temps totaux obtenus dans la simulation de l'ILA devrait être plus élevés.

L'analyse de l'ILA se fait en simulant la tâche dans les trois cas suivants:

A - pas d'arrêt aux points intermédiaires.

B - arrêt aux points intermédiaires et ajout d'un point de rectification dans le dernier segment.

C - arrêt aux points intermédiaires et arrêt à un point de rectification dans le dernier segment.

Le cas "A" représente l'utilisation normale de l'ILA. Une meilleure comparaison avec l'algorithme proposé est permise par les cas "B et C". Ces deux cas forcent le robot à visiter les points intermédiaires et utilisent une contrainte de forme (ajout d'un point dans le dernier segment). Les courbes cartésiennes de ces trois cas sont présentées ci-dessous. La simulation numérique requiert "7 à 8" secondes en temps de calcul.

A - Pas d'arrêt aux points intermédiaires:

La figure "4.2" montre différentes prises de vue de la courbe cartésienne générée par le modèle du robot ASEA. D'une distance de huit mètres, l'observateur regarde l'origine du système de base du robot. Les trois schémas sont obtenus par la projection de la scène tridimensionnelle sur un plan; cela produit un effet de perspective. En faisant varier la distance focale, on obtient des figures de grandeurs appropriées. Donc, les dimensions sur les figures ne doivent pas être comparées entre elles.

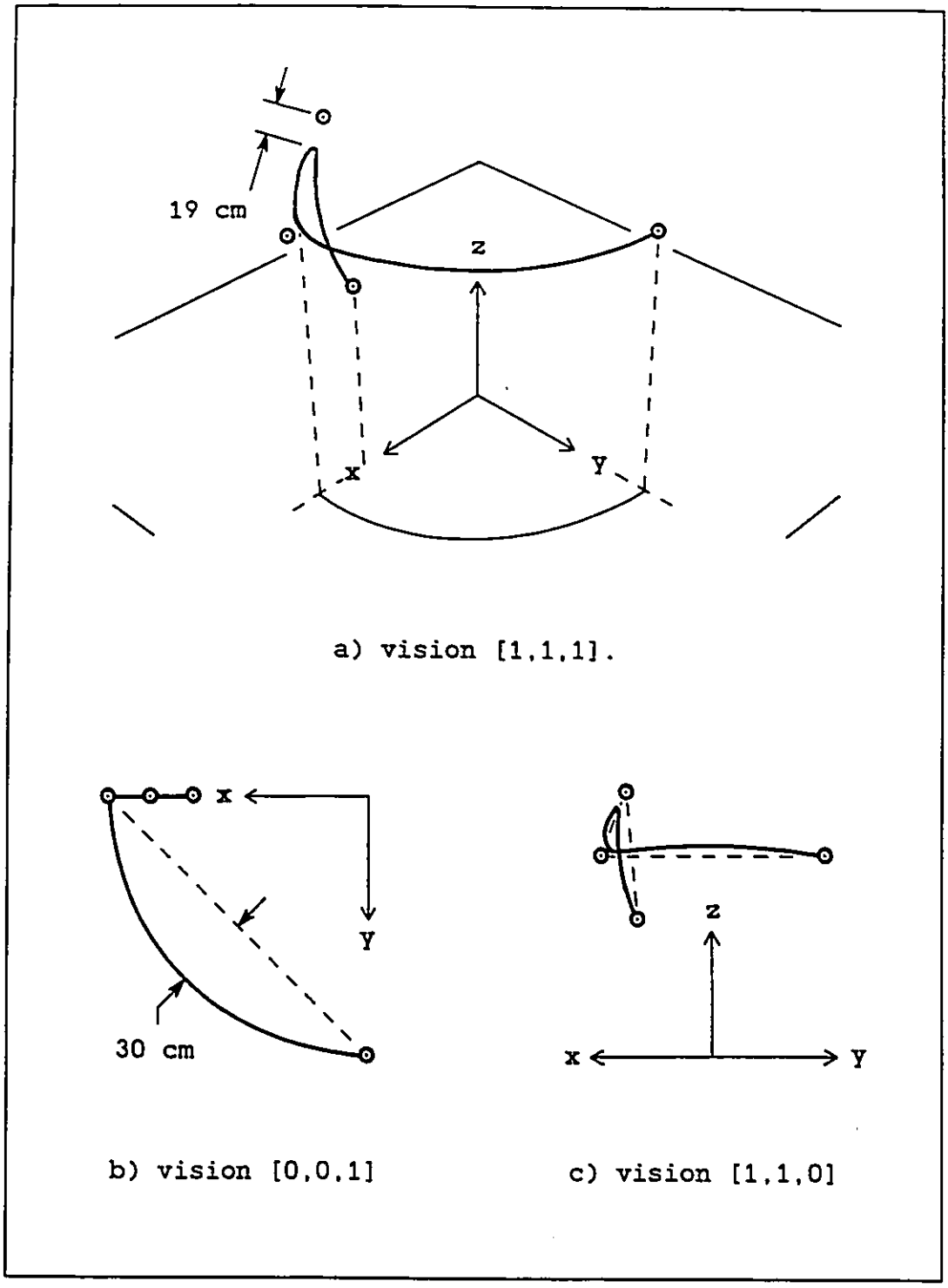


Fig.4.2 - ILA: pas d'arrêt aux points intermédiaires.

En regardant la figure "4.2a", on remarque que le manipulateur ne visite pas ses points intermédiaires; cela provient de l'interpolation faite à ces points. Cette erreur est relativement élevée au point "2" et est d'environ dix-neuf centimètres (19 cm). Sur la figure "4.2b", on s'aperçoit aussi d'une déviation importante par rapport à une ligne droite. La déviation maximale arrive au centre du dernier segment et est d'environ trente centimètres (30 cm) dans le plan XY. Cela représente une erreur de vingt pour cent (20%) par rapport à la distance entre les deux derniers points. La figure "4.2c" montre ces déviations selon un autre axe de vision.

Du côté temporel, cette tâche s'accomplit en un temps total de "1,83" seconde.

B - Arrêt aux points et point de rectification:

L'ajout d'un point de rectification se fonde sur l'idée de TAYLOR [70]: diminuer la déviation d'une courbe cartésienne en ajoutant des points intermédiaires. Un seul point est ajouté au centre du dernier segment du cas "B" de l'ILA. Ici, le robot atteint ses positions intermédiaires, car on a demandé un arrêt aux points intermédiaires. Cela permettra une comparaison plus juste entre les deux algorithmes: ils visitent leurs points et ont au moins une contrainte de forme. La figure "4.3" montre la courbe cartésienne obtenue selon des axes de vision différents.

L'arrêt aux points intermédiaires permet d'éliminer la déviation à la jonction des segments (figure "4.3a"), et

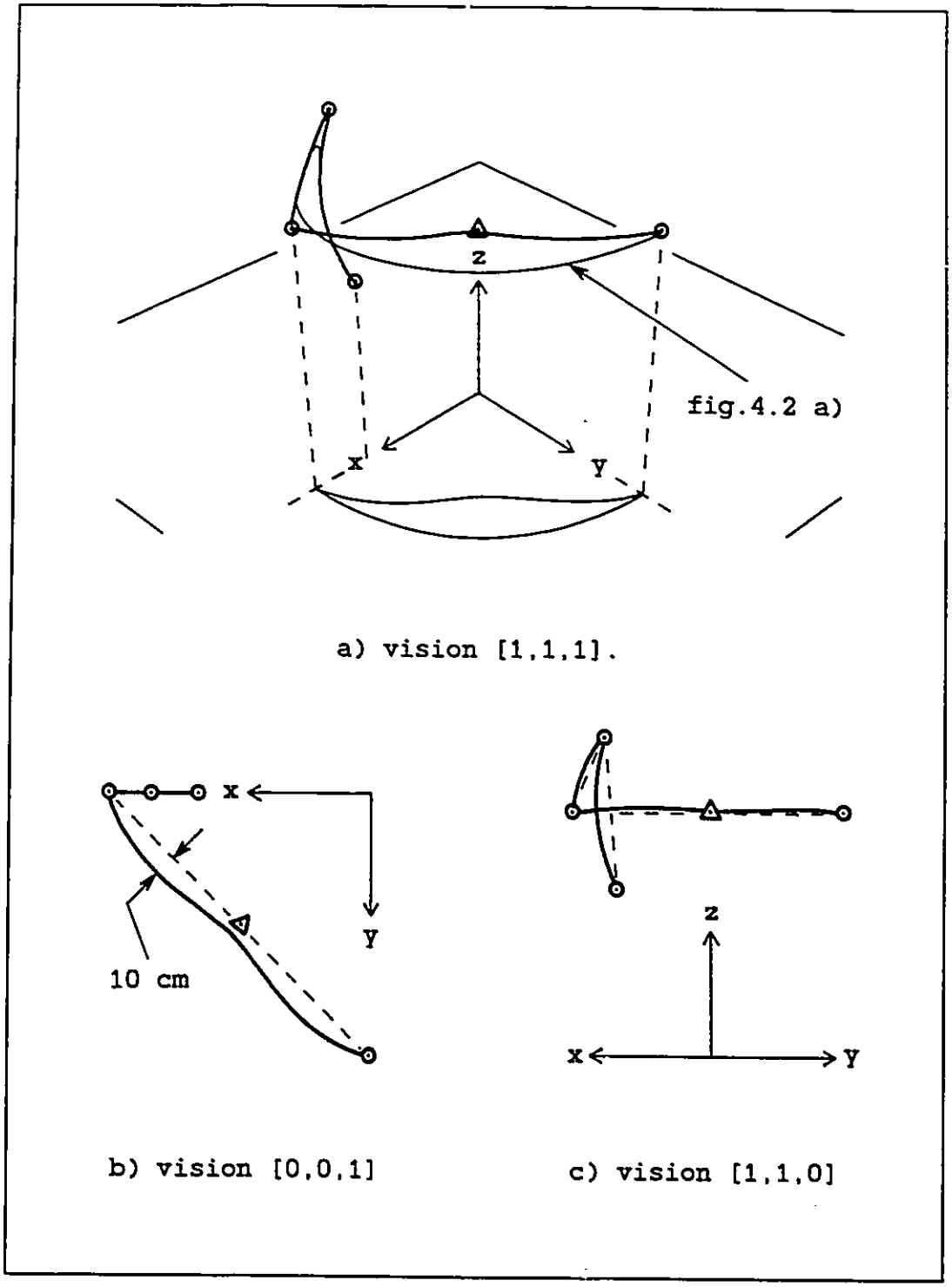


Fig.4.3 - ILA: arrêt aux points intermédiaires et ajout d'un point au dernier segment.

l'ajout du point de rectification réduit la déviation par rapport à une droite dans le dernier segment (figure "4.3b"). Par contre, la déviation maximale demeure élevée (10 cm); et l'ajout d'autres points serait nécessaire pour réduire cette erreur. Cette déviation représente une erreur de sept pour cent (7%) par rapport à la longueur du dernier segment. La figure "4.3c" montre que la déviation dans ce segment est diminuée selon l'axe "Z" comparativement à la figure "4.2c".

Dans cette simulation, le robot a besoin d'un temps total de "3,05" secondes; ce qui représente un accroissement absolu de "1,22" seconde et un accroissement relatif de 67% par rapport à la simulation précédente:

$$\frac{3,05 - 1,83}{1,83} = 67\%$$

Cette augmentation du temps provient de l'arrêt aux points intermédaires (2 points) et de l'ajout du point de rectification (1 point). Chacun de ces points introduit une nouvelle transition; donc, chaque point augmente le temps total de "$2 \cdot T_{acc}$" (T_{acc} = 0,2 sec.):

$$3 \text{ points} \cdot 2 \cdot T_{acc} = 1,2 \text{ secondes.}$$

Cela montre que l'algorithme de TAYLOR [70] a une faible performance par rapport au temps, et cela est inadéquat dans un contexte d'optimisation.

C - Arrêt aux points et arrêt au point de rectification:

Dans le but de diminuer la déviation au dernier segment, on fait la même simulation que précédemment en demandant au robot d'arrêter au point de rectification. Les résultats suivants sont obtenus (figure "4.4").

L'arrêt au point de rectification n'a pas réduit la déviation par rapport à une droite dans le dernier segment (figures "4.4b et 4.4c"); la déviation maximale demeure à 10 cm (7% d'erreur par rapport à la distance). Seule la déviation au point de rectification a été réduite.

Cette tâche nécessite un temps de "3,45" secondes; ce qui représente une augmentation "0,4" seconde par rapport à la simulation précédente. Cette augmentation du temps provient de l'arrêt au point de rectification:

$$2 \cdot T_{acc} = 0,4 \text{ seconde.}$$

4.2.2 - L'algorithme proposé: AP.

L'algorithme proposé est comparé avec l'ILA en étudiant les deux cas suivants:

A - pas de points de rectification.

B - ajout de points de rectification.

Le processus d'itérations se termine lorsque la variation la plus élevée des limites de vitesses et d'accélérations articulaires ne dépasse pas "5%" par rapport à la valeur précédente. Ce qui conduit à une erreur moyenne bien inférieure à "5%".

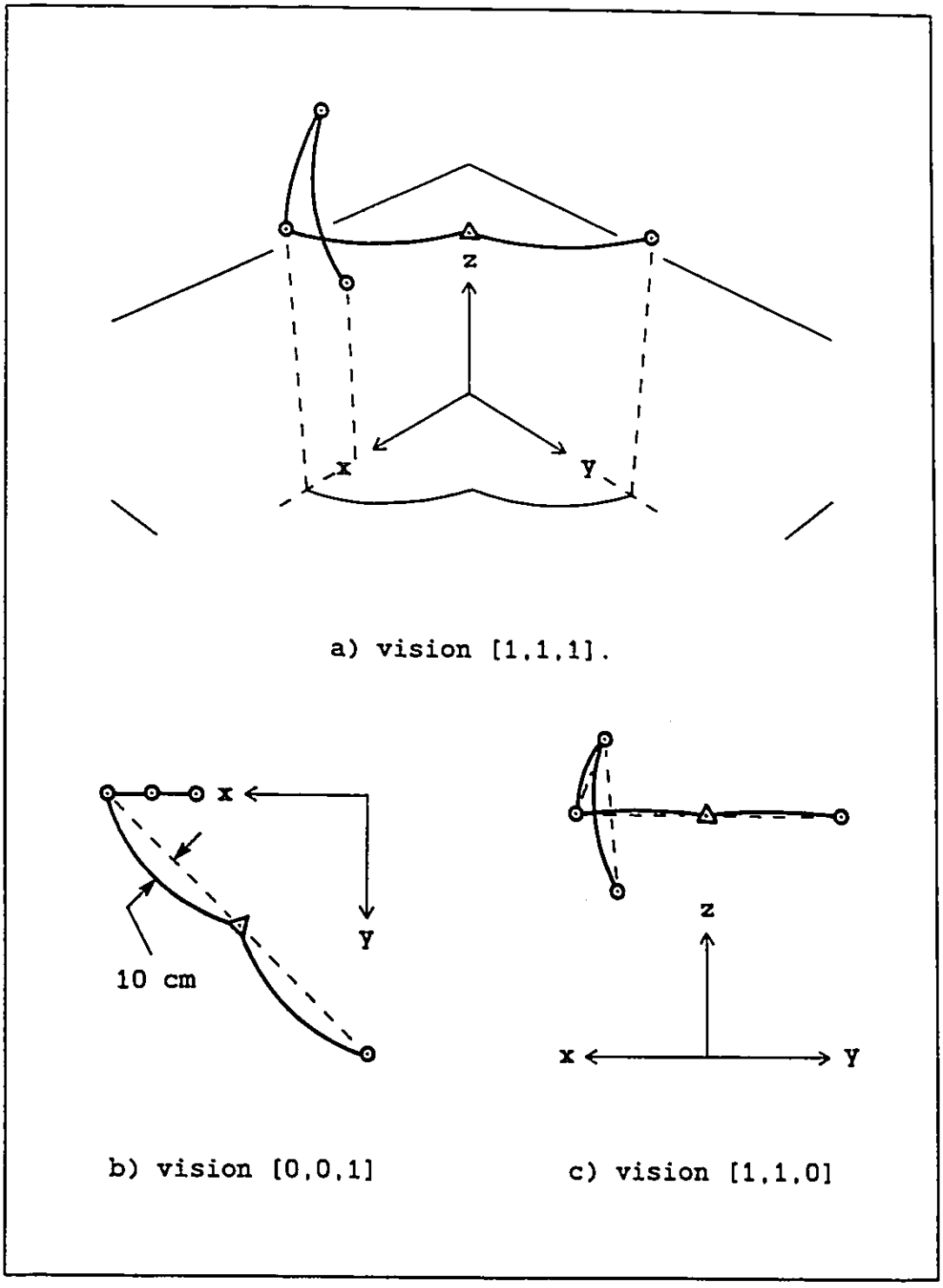


Fig.4.4 - ILA: arrêt aux points intermédiaires et au point de rectification.

A - Pas de points de rectification:

Cette simulation requiert deux minutes et demie en temps de calcul; soit huit itérations. La courbe cartésienne obtenue est illustrée à la figure "4.5".

En regardant la figure "4.5a", on remarque que la déviation cartésienne est réduite pour le premier segment par rapport aux figures "4.2a et 4.4a"; cette amélioration provient de la contrainte d'orientation de la vitesse cartésienne au deuxième point. Par contre, cela contribue à l'augmenter au deuxième segment puisque la vitesse est non-nulle au point "2". Dans le dernier segment, la contrainte d'orientation de la vitesse cartésienne n'a aucun effet (figures "4.5b et 4.5c"). Sur la forme géométrique de la courbe cartésienne (figure "4.5b"), on s'aperçoit de la même déviation par rapport à une ligne droite que sur la figure "4.2b"; soit environ trente centimètres (30 cm) dans le dernier segment.

Afin de visualiser le principe d'optimisation du temps de l'algorithme, les courbes d'efforts articulaires des six moteurs sont présentées dans la figure "4.6". A noter que les pointillés sur ces figures représentent la localisation des points intermédiaires "2 et 3" sur l'échelle du temps.

Les limites sur les efforts articulaires des moteurs "1 à 6" sont respectivement "150, 300, 300, 50, 25 et 25 N-m". La figure "4.6c" indique que le lien "3" représente l'articulation limitative pour les deux premiers segments. Au dernier segment, la figure "4.6a" montre que l'articulation

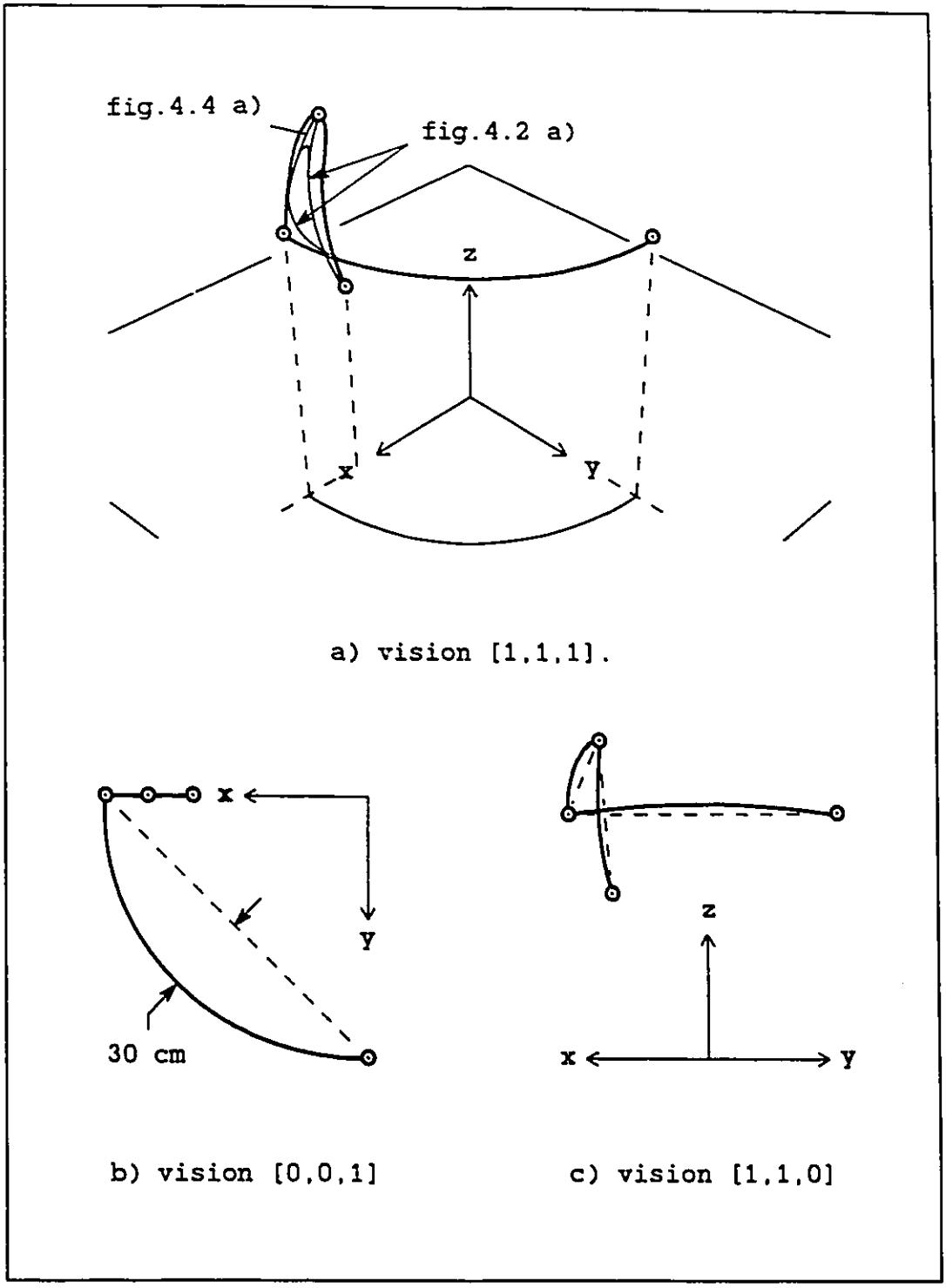
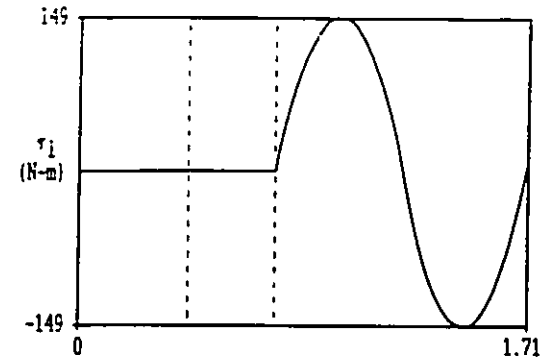
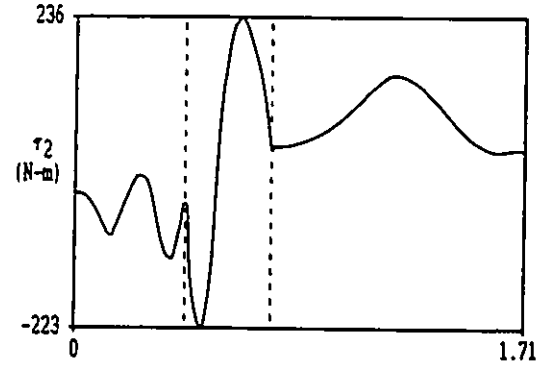


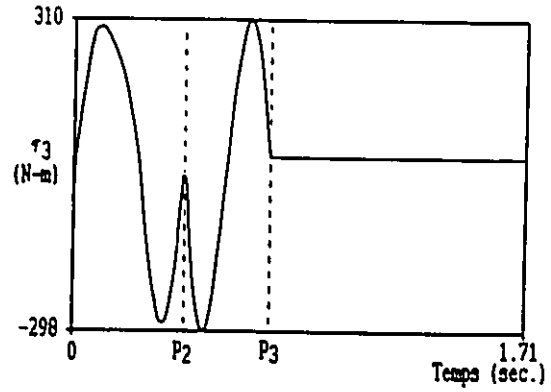
Fig.4.5 - AP: pas de points de rectification.



a) moteur "1".



b) moteur "2".



c) moteur "3".

Fig.4.6 - Les moments de torsion.

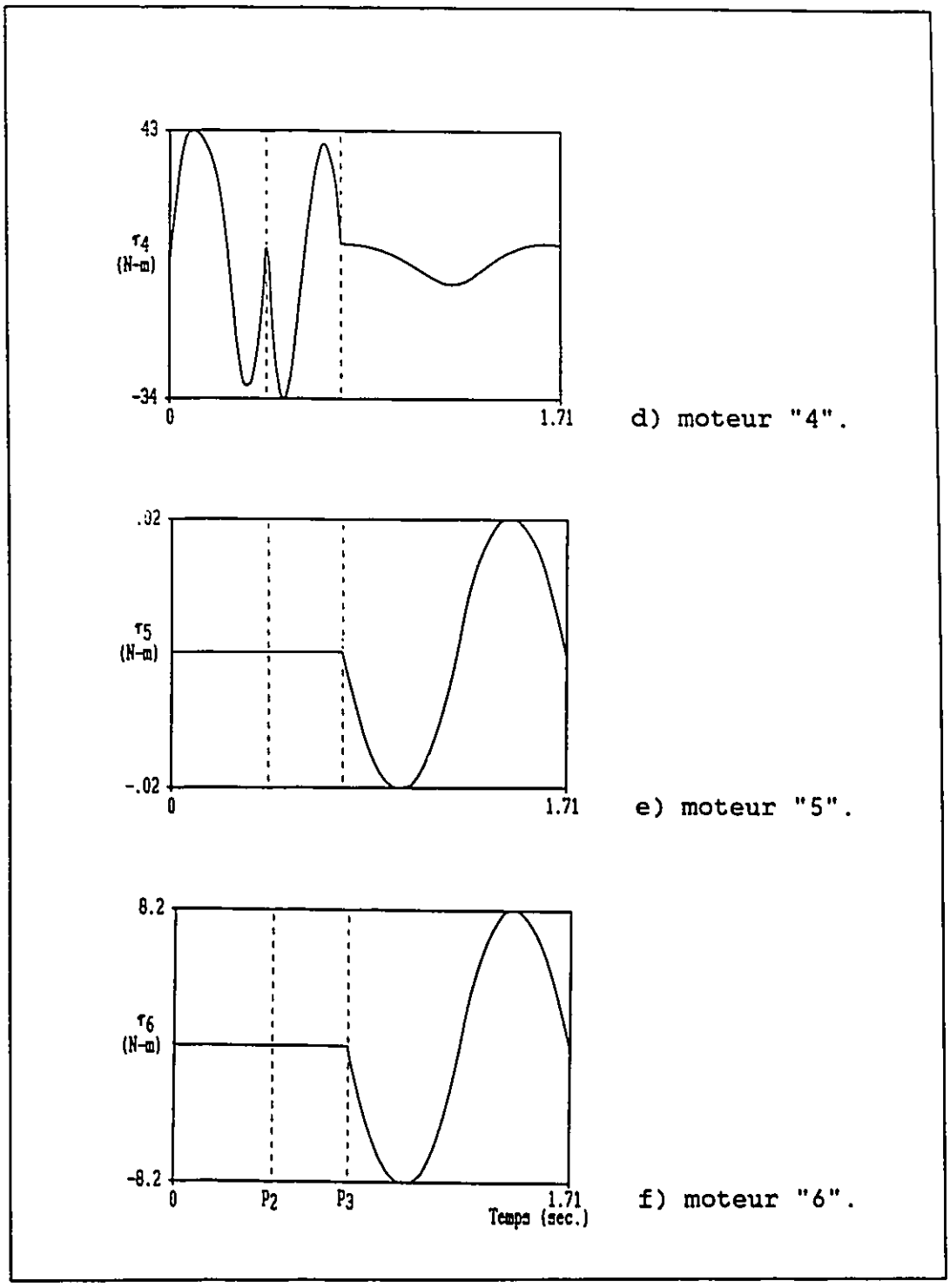


Fig.4.6 - Les moments de torsion (suite).

limitative est le lien "1". Dans un même segment, les articulations limitatives atteignent deux fois la limite articulaire imposée. L'effort articulaire des liens limitatifs s'apparente donc à un bang-bang: effort maximal dans un sens suivi d'un maximum dans l'autre sens. Donc, la modélisation mathématique des segments par deux transitions et une vitesse constante ressemble à une approximation du "bang-coast-bang".

La différence entre la valeur maximale des articulations limitatives et la valeur maximale imposée est inférieure à "5%" pour cette tâche. Cela suggère qu'une erreur maximale de "5%" sur la variation des limites de vitesses et d'accélérations articulaires produit une erreur inférieure à cinq pour cent pour les maximums d'efforts des articulations limitatives.

En regardant de plus près la figure "4.6c", on s'aperçoit que l'articulation limitative (lien "3") réduit son effort de torsion au point "2". Une façon de réduire le temps serait d'éviter cette réduction de l'effort en ne choisissant pas des valeurs nulles pour les contraintes d'accélération "E et F" aux bornes des transitions.

Comme indiqué sur la figure "4.6", le robot exécute cette tâche en un temps total de "1,71" seconde comparativement à "1,83" seconde pour la première simulation de l'ILA. Cela représent une diminution de sept pour cent (7%). Par contre, cette réduction est beaucoup plus significative par rapport à la deuxième simulation de l'ILA:

$$\frac{3,05 - 1,71}{3,05} = 44\%$$

Cette dernière comparaison est plus juste, car les deux algorithmes produisent des trajectoires où le robot passe par les points intermédiaires.

B - Ajout de points de rectification:

La déviation au dernier segment étant très élevée (figure "4.5b"), on voit la nécessité d'ajouter une contrainte de rectification sur la position. On a utilisé un facteur de forme de dix pour cent (ϵ = 0,10) à chaque segment. L'algorithme trouve l'articulation limitative et son temps en prenant neuf itérations (2,6 min.). Puis, les points de rectification sont ajoutés en faisant seize itérations (5,3 min.). Le temps total de calcul est donc inférieur à huit minutes. La courbe cartésienne obtenue est montrée à la figure "4.7".

Sur la figure "4.7a", on remarque une amélioration considérable par rapport à la simulation de la figure "4.5a"; les déviations cartésiennes sont réduites par la rectification. La déviation maximale au dernier segment est de trois centimètres (3 cm); soit une erreur de deux pour cent (2%) par rapport à la distance entre les points (figure "4.7b"). Cette erreur est plus faible que celle obtenue pour l'ILA avec un point de rectification dans le dernier segment; soit "2%" par rapport à "7%". Cette différence est attribuable aux choix de vitesse au point de rectification. L'utilisa-

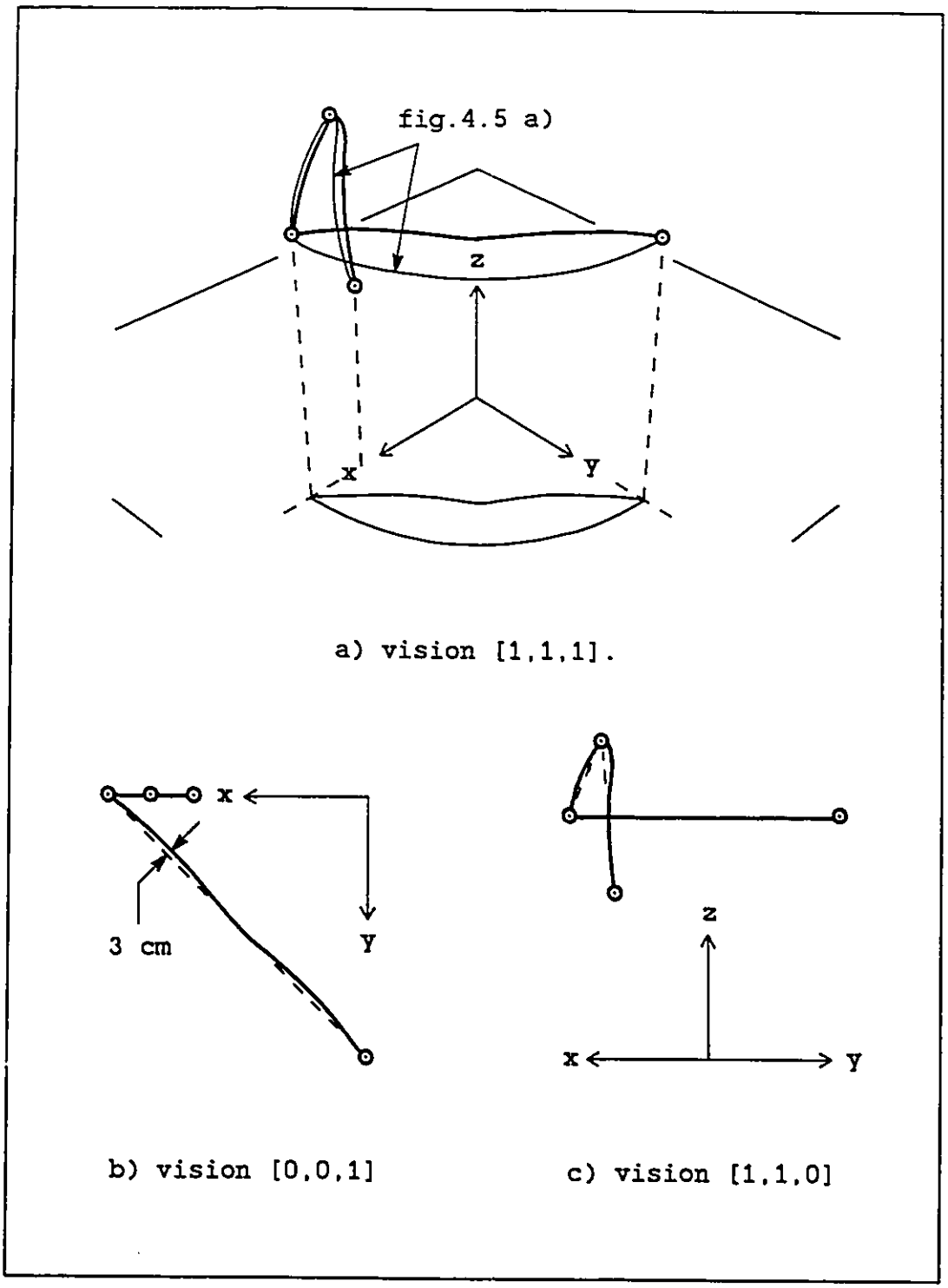


Fig.4.7 - AP: ajout de points de rectification.

tion d'une contrainte sur la vitesse aux points intermédiaires et aux points de rectification réduit le nombre de points requis dans l'approximation d'une courbe cartésienne. L'algorithme proposé est donc plus performant que celui proposé par TAYLOR [70].

Du côté temporel, le robot a besoin d'un temps de "1,60" seconde pour faire cette tâche; ce qui représente une légère diminution de "0,11 seconde" par rapport à la simulation précédente. La rectification affecte peu le temps, car l'articulation limitative n'a pas à exécuter une distance supplémentaire; seul son temps est modifié par les effets de couplage dynamique. Comparativement au troisième cas étudié dans l'ILA (3,45 sec.), ce temps de "1,60" seconde représente une réduction de plus de la moitié; soit 54%.

En regroupant l'ensemble des résultats de l'ILA et de l'algorithme proposé, on obtient le tableau "4.2". L'ILA permet d'obtenir des temps de calculs courts, mais cet algorithme offre certains inconvénients. De fait, le robot ne visite pas les points intermédiaires; et les courbes cartésiennes générées sont imprévisibles. Une façon de réduire ces déviations est de faire arrêter le robot aux points intermédiaires; mais cela augmente le temps total de "$2 \cdot T_{acc}$" pour chaque arrêt. TAYLOR [70] a rectifié la courbe cartésienne en ajoutant des points intermédaires. Cela augmente aussi le temps d'une trajectoire de "$2 \cdot T_{acc}$" pour chaque point de rectification ajouté.

TABLEAU 4.2

LES RESULTATS DES DEUX ALGORITHMES

Caractéristiques	ILA			AP	
	A	B	C	A	B
Temps de calcul (sec.)	7	8	8	150	474
Robot visite les points	non	oui	oui	oui	oui
Robot arrête aux points	non	oui	oui	non	non
Déviation maximale (cm)	30	10	10	30	3
Temps total du déplacement (sec.)	1,83	3,05	3,45	1,71	1,60

Le nouvel algorithme corrige les lacunes de l'ILA. Le manipulateur visite les points intermédiaires; ce qui diminue la déviation à ces points. La trajectoire est aussi rectifiée par l'ajout de contraintes géométriques. Ces deux corrections n'augmentent pas le temps d'une trajectoire. De plus, le temps d'un segment est minimisé sous l'influence de la dynamique. Le présent algorithme possède donc de meilleures performances que l'algorithme de TAYLOR [70] pour approximer une courbe cartésienne.

Conclusion:

Comme on a pu le constater tout au long de ce travail, la planification optimale de trajectoire est un problème compliqué; cela est dû aux caractéristiques souhaitées pour une trajectoire. Les premières tentatives de planification de trajectoires reposent sur une approche purement cinématique qui utilise des contraintes globales sur la trajectoire; ce qui conduit à une solution avec de faibles performances.

L'introduction de la dynamique dans la planification de la trajectoire améliore ces performances. Evidemment, cela contribue à compliquer la solution. Dans le but de trouver la trajectoire optimale, la recherche actuelle se concentre sur l'utilisation de la théorie de l'asservissement optimal et de la discrétisation de l'espace solution. Ces deux approches utilisent un indice de performance unique (généralement, le temps) et ne tiennent pas compte de certaines caractéristiques souhaitables: la prévisibilité de la courbe cartésienne, la généralité de l'algorithme et l'efficacité de calcul. De fait, aucune contrainte cartésienne autre que la position du point initial et final est utilisée. Ces algorithmes permettent seulement la planification d'un mouvement rapide du manipulateur (optimisation du temps). Le temps de calcul associé à un robot avec seulement deux degrés de liberté est très long et la solution semble possible pour un robot avec trois articulations. La génération de trajectoire avec des robots à six degrés de liberté demande un compromis entre l'optimalité et la facilité de solution.

L'algorithme proposé apporte certains éléments de réponse aux problèmes associés aux précédents algorithmes. Cet algorithme respecte la majorité des caractéristiques d'une trajectoire idéale. En outre, il permet d'améliorer la prévisibilité de la courbe cartésienne en ajoutant des contraintes géométriques. La modélisation mathématique des segments en deux transitions et une vitesse constante facilite l'introduction des contraintes géométriques et permet d'appliquer l'algorithme aux robots avec plus de trois degrés de liberté. Cette formulation mathématique ramène le problème global de minimisation du temps en un problème local d'optimisation. Au lieu de minimiser le temps, on optimise des vitesses et des accélérations maximales articulaires. On a constaté que le temps n'est pas optimal à cause de l'approximation de la trajectoire par une fonction analytique et à cause de l'heuristique choisie pour le calcul des vitesses articulaires aux points intermédiaires. Dans certains cas, le fait de donner des valeurs nulles à "E et F" affecte aussi l'optimalité temporelle; car il n'est pas toujours nécessaire d'avoir ces conditions aux bornes des transitions. Par contre, ces simplications augmentent l'efficacité de calcul et rendent possible l'application de l'algorithme aux robots avec six degrés de liberté.

Comparativement à l'ILA, l'algorithme proposé force le robot à passer par les points intermédiaires en affectant peu le temps total de la trajectoire. L'utilisation de contraintes de forme donne à l'algorithme un certain contrôle sur la forme de la courbe cartésienne. La contrainte sur la vitesse aux points intermédiaires et aux points de rectification réduit le nombre de points

requis pour l'approximation d'une courbe cartésienne sans vraiment affecter le temps. L'algorithme proposé est, par conséquent, plus performant que celui proposé par TAYLOR [70]. De plus, il permet d'optimiser le temps sous l'influence de la dynamique.

La différence de l'algorithme présenté dans cette thèse avec les autres algorithmes utilisant la dynamique, c'est qu'il ne cherche pas à résoudre un problème spécifique (ex.: optimisation du temps entre deux points); mais bien le problème complet de la planification: satisfaction des caractériques idéales, spécification de la tâche et présence de points intermédiaires.

Dans un travail futur, il y aurait quelques améliorations à apporter à l'algorithme. Premièrement, il faudrait améliorer l'heuristique de l'étape "2" de la PLT. La norme de la vitesse articulaire aux points intermédiaires devrait être limitée en fonction de la dynamique du manipulateur et choisie de façon à minimer le temps global de la trajectoire. Deuxièmement, on pourrait choisir des accélérations non-nulles aux limites des transitions. Une combinaison appropriée d'accélérations articulaires pourrait améliorer la forme de la courbe cartésienne. Dans certain cas, cela contribuerait aussi à optimiser le temps puisque l'articulation limitative n'aurait pas besoins de décélérer à la jonction de deux segments; et par conséquent, elle pourrait fournir un effort plus élevé et plus constant. Par exemple, si un des moteurs représente l'articulation limitative pour deux segments consécutifs et si le mouvement doit se faire dans la même direction, alors il est évident qu'une accélération non-nulle à la jonction des segments réduirait le temps. Troisièmement, il serait possible de limiter

la déviation maximale par l'ajout d'autres points de rectification lorsque cela s'avèrerait nécessaire. En terminant, l'introduction du problème d'évitement d'objets permettrait d'ajouter des points libres pour éviter les collisions.

ANNEXES

L'ETAT DES CONNAISSANCES POUR LES SUJETS RELATIFS

ANNEXE A: La cinématique des robots.

La cinématique est l'étude géométrique du mouvement et consiste à calculer le mouvement du manipulateur sans regard aux efforts nécessaires pour le produire. Ce problème se divise en deux types de problème: la cinématique directe et la cinématique inverse.

A.1 - La cinématique directe:

La résolution de ce problème s'énonce comme suit: Connaissant le mouvement dans le domaine articulaire, on est intéressé par le mouvement cartésien correspondant. Pour résoudre ce problème, on se base sur le calcul matriciel.

- La position directe: de façon à représenter la position généralisée de l'organe terminal par rapport au système de référence (X), on fixe un système de coordonnées sur chaque articulation en utilisant la notation de Denavit-Hartenberg [50]. La position généralisée est donc obtenue en faisant la multiplication successive des matrices de transformation. En utilisant la notation employée par PAUL [50]:

$$T = A_1 \cdot A_2 \cdot \cdot \cdot A_n \qquad (A.1)$$

$$X = Z \cdot T \cdot G \qquad (A.2)$$

- La vitesse directe: WHITNEY [74] a étudié la relation entre le domaine articulaire et le domaine cartésien. La relation mathématique se présente comme suit:

$$\dot{X} = J \cdot \dot{\vec{q}} \qquad (A.3)$$

où "J" représente la transformation jacobienne. Cette relation relie les vitesses cartésiennes généralisées par rapport aux vitesses articulaires (voir la figure "1.0"). Une méthode systématique pour développer la matrice jacobienne est présentée ASADA et SLOTINE [2].

- L'accélération directe: Le calcul de l'accélération cartésienne est dérivée de l'équation (A.3):

$$\ddot{X} = J \cdot \ddot{\vec{q}} + \dot{J} \cdot \dot{\vec{q}} \qquad (A.4)$$

A.2 - La cinématique inverse:

La cinématique inverse consiste à résoudre le problème suivant: connaissant le mouvement dans le domaine cartésien, on est intéressé par le mouvement articulaire correspondant.

- La position inverse: mathématiquement, on peut utiliser la transformation inverse de l'équation (A.2):

$$T = Z^{-1} \cdot X \cdot G^{-1} \qquad (A.5)$$

Pour chaque position "X", il existe généralement plusieurs configurations. Cela contribue à rendre la planification de la trajectoire plus complexe à résoudre, car elle requiert un choix parmi les solutions. Ce choix affectant les caractéristiques de la trajectoire, il est important de choisir la configuration appropriée. Généralement, si on choisit une solution particulaire pour le premier point, on garde la même pour les autres points.

- La vitesse inverse: de l'équation (A.3), on obtient:

$$\dot{\vec{q}} = J^{-1} \cdot \dot{X} \qquad (A.6)$$

L'inverse de la matrice jacobienne fait apparaître le problème de singularité mathématique. Les singularités indiquent qu'aucune combinaison des vitesses articulaires ne peut donner la vitesse cartésienne selon l'une des articulations. Ces singularités peuvent être enlevées en ajoutant des degrés de mobilité supplémentaires. Selon BRADY [5], la redondance du bras humain est utilisée pour éviter les positions singulières dans l'espace de travail. Cette redondance peut être manipulée par l'utilisation de la pseudo-inverse de la matrice jacobienne.

- L'accélération inverse: de l'équation (A.4), on peut développer la relation suivante:

$$\ddot{\vec{q}} = J^{-1} \cdot [\, \ddot{X} - \dot{J} \cdot \dot{\vec{q}} \,]^{-1} \qquad (A.7)$$

ANNEXE B: La dynamique des robots.

L'étude de la dynamique permet l'analyse des forces pour générer le mouvement du manipulateur. La dynamique exprime donc la relation entre le mouvement du manipulateur et les efforts fournis par les différents moteurs. Cette relation s'appelle les équations du mouvement du robot.

Dans cette section, on étudiera les trois sujets suivants: le développement des équations de mouvement, la dynamique des robots avec une chaîne fermée et la propriété de l'échelonnement dynamique du temps.

B.1 - Les équations du mouvement:

Deux méthodes de base peuvent être utilisées pour développer les équations du mouvement: la formulation de Newton-Euler et la formulation de Lagrange. Certains auteurs considèrent la formulation de Kane comme étant la troisième façon de développer les équations de mouvement. L'équation de Kane se base sur des principes d'énergie et d'efforts généralisés; cette équation est similaire à l'équation de Lagrange.

L'approche de Newton-Euler se base sur le principe d'équilibre en tenant compte des efforts internes et externes qui agissent sur un lien; ce qui permet de développer la forme implicite des équations de mouvement puisque les efforts articulaires ne sont pas décrits explicitement. Donc, d'autres transformations sont nécessaires pour obtenir les équations explicites.

La formulation de Lagrange est décrite en terme d'énergie et de travail en utilisant des coordonnées généralisées. Seulement les efforts produisant du travail apparaissent; ce qui résulte à une forme explicite des efforts articulaires dans les équations de mouvement.

SILVER [68] a montré qu'il n'y a pas de différence fondammentale entre la formulation de Newton-Euler et la formulation de Lagrange: en choisissant un choix approprié de coordonnées pour la formulation de Lagrange, les équations de mouvement des deux méthodes sont identiques.

Pour permettre le calcul rapide de la dynamique lors de l'asservissement, des algorithmes récursifs ont été développés pour les manipulateurs avec des chaînes cinématiques ouvertes. En utilisant des contraintes cinématiques, l'extension de ces algorithmes est possible pour les robot ayant des chaines cinématiques fermées (ex. robot ASEA). Historiquement, les algorithmes de calcul récursif sont basés sur la façon d'obtenir les équations du mouvement. LUH et al. [41] ont présenté un algorithme récursif qui utilise la formulation de Newton-Euler. HOLLERBACH [21] a développé une méthode basée sur la formulation de Lagrange et a démontré que la méthode proposée par LUH et al. [41] était plus rapide à calculer. Comme les deux formulations peuvent être identiques (SILVER [68]), c'est le mauvais choix de coordonnées

qui rend l'algorithme d'HOLLERBACH [21] plus lent. De plus, l'algorithme de Kane et Levinson [25] est encore plus rapide que celui de LUH et al. [41]; car ils ont évité le calcul répétitif inutile de certaines variables. Ces trois algorithmes utilisent un calcul récursif en deux étapes: une étape cinématique pour calculer le mouvement du manipulateur et une étape dynamique pour calculer les efforts nécessaires pour produire le mouvement.

B.2 - La dynamique des chaînes fermées:

Certains robots possèdent une ou plusieurs chaînes cinématique fermées; le robot ASEA IRB/6 en est un exemple. Pour ces manipulateurs, les algorithmes récursifs précédemment présentés ne peuvent pas être directement appliqués.

Plusieurs algorithmes se basent sur la technique des multiplicateurs de Lagrange pour résoudre ce problème. En autres, LUH et ZHENG [42] ont utilisé cette méthode. En ouvrant la chaine de façon virtuelle, ils calculent la dynamique pour chaque branche du manipulateur en utilisant un algorithme récursif connu (Newton-Euler). Puis, en applicant des contraintes cinématiques, ils trouvent les efforts réels. MEGAHED et FAESSLER [24] ont développé une façon semblable de résoudre le problème, mais ils ont basé leur algorithme sur la formulation de Lagrange. Ces façons de procéder ont le désavantage d'augmenter le temps de calcul à cause de l'introduction et de l'élimination des multiplicateurs de Lagrange.

WANG [72] et KANE [24] ont présenté des algorithmes basés sur la formulation de Kane. Ces algorithmes n'ont pas besoin d'introduire et d'éliminer les multiplicateurs inconnus puisque les contraintes cinématiques sont utilisées pour développer les équations du mouvement.

B.3 - L'échelonnement dynamique du temps:

Cette propriété fut identifiée par HOLLERBACH [22] et permet de modifier la vitesse du mouvement sans recalculer entièrement la dynamique si l'algorithme est appliquée pour l'ensemble de la trajectoire. Cela permet de vérifier si une trajectoire est faisable; et sinon, comment la modifier pourqu'elle le soit. Les vitesses sont ainsi modifier sans changer le profil de la courbe cartésienne. Cette sous-optimisation du temps est généralement faite lors de l'asservissement. La trajectoire ainsi optimisée n'est pas optimale par rapport au temps puisque le maximum d'effort est atteint seulement une fois pour une trajectoire donnée.

Cette propriété peut être utilisée pour simplifier l'optimisation des trajectoires. Elle fut utilisée dans un contexte de recherche intensive [56,57] pour l'optimisation du temps. Cette propriété a été utilisée pour le nouvel algorithme présenté dans cette thèse de maîtrise. Voici les fondements de cette propriété:

Les équations de mouvement d'un manipulateur sont données par la relation suivante:

$$\vec{n}(t) = I(t)\cdot\ddot{\vec{q}}(t) + \dot{\vec{q}}(t)\cdot C(t)\cdot\dot{\vec{q}}(t) + \vec{G}(t) \qquad (B.1)$$
$$= \vec{n}_a(t) + \vec{G}(t)$$

Soit une nouvelle trajectoire "$\vec{q}^{\,\bullet}[r(t)]$" définie par rapport à la trajectoire existante "$\vec{q}(t)$":

$$\vec{q}^{\,\bullet}[r(t)] = \vec{q}(t) \qquad (B.2)$$

où "r(t)" représente une fonction croissante par rapport au temps. Cette fonction permet de faire un nouvel échelonnement du temps.

En remplaçant l'équation (B.2) dans (B.1), on obtient les équations de mouvement suivant pour la nouvelle trajectoire:

$$\vec{n}^{\,\bullet}(t) = \vec{n}_a^{\,\bullet}(t) + \vec{G}(t) \qquad (B.3)$$

et,

$$\vec{n}_a^{\,\bullet}(t) = \dot{r}^2\cdot\vec{n}_a(r) + r\cdot I(r)\cdot\frac{d\vec{q}(r)}{dr} \qquad (B.4)$$

Donc, les nouveaux efforts articulaires "$\vec{n}_a^{\,\bullet}(t)$" sont fonctions des anciens "$\vec{n}_a(r)$ par le facteur d'échelle $\dot{r}^2$ plus un autre terme.

Le facteur d'échelonnement du temps est une fonction croissante du temps. La fonction la plus simple correspond à une rampe:

$$r(t) = c\cdot t \qquad (B.5)$$

où "c" est une constante quelconque.

Pour ce cas particulier, l'équation (B.4) se réduit à l'expression suivante:

$$\vec{n}_a^{\,\bullet}(t) = c^2\cdot\vec{n}_a(c\cdot t)$$

Cette équation étant valide pour tout temps, on peut diviser le temps par "c":

$$\vec{n}_a^{\,\bullet}(t/c) = c^2\cdot\vec{n}_a(t) \qquad (B.6)$$

Donc, si "c" est supérieur à "1", alors le temps alloué pour la trajectoire est réduit. Puisque "c" est compris dans l'interval $[c^-, c^+]$, alors on choisit la valeur "c^+" pour minimiser le temps.

Supposons que "$\vec{n}_a(t)$" a été calculé séparément de l'effet de gravité "$\vec{G}(t)$" et que les extrémums des efforts articulaires sont constants et donnés par:

$$\vec{n}^- \leq \vec{n}(t) \leq \vec{n}^+$$

(note: il est possible de tenir compte de limites variables sur les efforts articulaires).

Les effets de gravité n'étant pas affectés par l'échelonnement dynamique du temps, on peut inclure ces effets dans les limites des efforts articulaires; ce qui donne de nouvelles limites:

$$\begin{aligned} \vec{n}^+(t/c) &= \vec{n}^+ - \vec{G}(t) \\ \vec{n}^-(t/c) &= \vec{n}^- - \vec{G}(t) \end{aligned} \qquad \text{(B.7)}$$

Pour calculer le "c" qui minimise le temps, on utilise les équations (B.6) et (B.7) pour chaque temps et pour chaque articulation. La valeur recherchée est la plus petite valeur sur l'interval de temps et pour chaque articulation:

$$c^{i+} = \min \left[c_i^{i+}(t) \right]_{i=1,\ t=0}^{i=n,\ t=t_f} \qquad \text{(B.8)}$$

Pour le calcul de $c_i^{i+}(t)$, les trois cas suivants doivent être considérés:

TABLEAU A.1

LE CALCUL DE LA CONSTANTE "c^+" D'ECHELONNEMENT.

conditions	$c_i^{i+}(t)$		
	cas 1	cas 2	cas 3
	$n_i^-(t) < 0$ $n_i^+(t) > 0$	$n_i^-(t) > 0$ $n_i^+(t) > 0$	$n_i^-(t) < 0$ $n_i^+(t) < 0$
$n_{ai}(t) > 0$	$n_i^+(t)/n_{ai}(t)$	$n_i^+(t)/n_{ai}(t)$	infaisable
$n_{ai}(t) = 0$	∞	infaisable	infaisable
$n_{ai}(t) < 0$	$n_i^-(t)/n_{ai}(t)$	infaisable	$n_i^-(t)/n_{ai}(t)$

En applicant l'équation (B.8) sur chaque segment de trajectoire, on peut calculer les nouvelles limites sur les accélérations et les vitesses maximales en utilisant les équations (3.23 et 3.25).

ANNEXE C: L'asservissement.

L'asservissement d'un manipulateur consiste à suivre le mouvement désiré le long d'une trajectoire déjà générée. Pour se faire, le système d'asservissement produit une action compensative en faisant varier les efforts articulaires pour corriger la déviation du bras par rapport à la trajectoire désirée.

De façon générale, il existe deux niveaux d'asservissement: le système d'asservissement local et le système d'asservissement global. Communément, chaque moteur est asservi par un régulateur de type PID; ce qui constitue l'asservissement local ou articulaire. Le système d'asservissement global commande les systèmes d'asservissement locaux en leur fournissant les trajectoires articulaires pour produire la tâche désirée.

Habituellement, la résolution du problème d'asservissement global consiste à:

1° obtenir des modèles dynamiques du robot en utilisant les formulations de Newton-Euler, de Lagrange ou de Kane.
2° utiliser ces modèles pour déterminer les lois ou les tactiques d'asservissement qui produiront les performances et la réponse désirées.

Les références [23], [34], [40], [48], [49], [54], [58], [59], [64], [73], [74] et [75] traitent de ce sujet.

ANNEXE D: L'évitement d'objets.

L'espace de travail d'un robot contient de nombreux objets (machines numériques, autres robots, objets à manipuler, convoyeurs, ...). Certains objets sont statiques; d'autres se déplacent dans la cellule de travail. Cette présence d'objets fait apparaître le problème des collisions lors de la planification de la trajectoire. Etant donné le coût relié aux collisions, il est très important de les éviter.

LOZANO-PEREZ [39] a développé un algorithme d'évitement d'objets en utilisant des considérations géométriques. De plus, il assume que les objets sont statiques dans l'espace de travail du robot. Pour tenir compte du mouvement de certains objets, KANT et ZUCKER [26] ont séparé le problème en deux parties:
planifier la courbe pour éviter les objets statiques et planifier les vitesses pour éviter les objets en mouvement.

ANNEXE E: L'interpolation lineaire articulaire.

Cet algorithme fut proposé par Paul [3]. Le système d'asservissement du robot doit calculer le temps pris par chaque moteur pour atteindre sa destination à la vitesse commandée. Puis, il choisit le temps le plus long "T_j" et utilise cette valeur pour réduire les vitesses des autres articulations. De façon à établir la continuité du mouvement entre les segments à vitesse constante, une transition est ajoutée aux points.

Voici l'algorithme de calcul pour l'interpolation linéaire articulaire:

Posons:

- i : articulation 'i' (i = 1, n).
- j : numéro du segment.
- q_{ij} : position de l'articulation au début du segment 'j'.
- q_{ij+1} : position de l'articulation à la fin du segment 'j'.
- T_{acc} : temps pour accélérer un articulation du repos à vitesse maximale pour la pire configuration du robot (cette valeur est une constante pour chaque robot, mais difficile à calculer).

1 - temps alloué pour chaque segment: segment 'j'.

- distances:

$$\Delta q_{ij} = q_{ij+1} - q_{ij}$$

- temps pour chaque variable:

$$T_{ij} = \Delta q_{ij} / \dot{q}max_i$$

- temps alloué:

$$T_j = MAX \left[T_j \, , \, 2 \cdot T_{acc} \right]_{i=1}^{i=n}$$

2 - interpolation:

- transition: $(-T_{acc} \leq t \leq T_{acc})$.

$$q_{ij}(h) = [\Phi \cdot (2-h) \cdot h^2 - 2 \cdot \Delta^*] \cdot h + q^*$$

$$\dot{q}_{ij}(h) = [\Phi \cdot (1.5-h) \cdot 2 \cdot h^2 - \Delta^*] / T_{acc}$$

$$\ddot{q}_{ij}(h) = \Phi \cdot (1-h) \cdot 3 \cdot h \; / \; T_{acc}^2$$

où,

$$\Delta q_{ij} = q_{ij+1} - q_{ij}$$

$$q^* = q_{ij} - \frac{\Delta q_{ij-1}}{T_{j-1}} \cdot T_{acc}$$

$$\Delta^* = q^* - q_{ij}$$

$$\Phi = \Delta q_{ij} \cdot T_{acc} / T_j + \Delta^*$$

$$h = (t + T_{acc}) \; / \; (2 \cdot T_{acc})$$

· vitesse constante: $(T_{acc} \leq t \leq T_j - T_{acc})$.

$$q_{ij}(h) = \Delta q_{ij} \cdot h + q_{ij}$$

$$\dot{q}_{ij}(h) = \Delta q_{ij} \; / \; T_j$$

$$\ddot{q}_{ij}(h) = 0$$

où,

$$h = t \; / \; T_j$$

APPENDICES

APPENDICE A: Note sur les coefficients pour les transitions.

Pour comparer la modélisation des transitions obtenues pour le pour un polynôme du cinquième degré, l'appendice A.1 présente les résultats pour des polynômes d'ordres différents: 3e, 7e et 9e degré. Puis, à la section suivante, on évalue l'accélération moyenne constante pour la modélisation retenue (5e degré).

A.1 - Les polynômes de degrés divers:

La modélisation mathématique des transitions peut utiliser des polynômes d'ordres différents selon les contraintes de continuité imposées sur le mouvement. On a déjà présenté le résultat pour un polynôme du 5e degré.

A.1.1 - Le polynôme du 3e degré:

En imposant la continuité de la position et de la vitesse, il faut satisfaire les contraintes suivantes:

$$q(-T) = A \qquad q(T) = B$$
$$\dot{q}(-T) = C \qquad \dot{q}(T) = D$$

On obtient alors les coefficients suivants:

$$a_0 = (A+B)/2 + (C-D)\cdot T/4$$

$$a_1 = -[3\cdot(A-B) + (C+D)\cdot T] \;/\; (4\cdot T)$$

$$a_2 = -(C-D) \;/\; (4\cdot T)$$

$$a_3 = [(A-B) + (C+D)\cdot T] \;/\; (4\cdot T^3)$$

En développant les valeurs de "B et T" de la même façon que pour le polynôme du 5e degré, on obtient la solution suivante:

$$B = A + T\cdot(C+D)$$

$$T = (D-C) \;/\; (2\cdot\alpha_{max})$$

A.1.2 - Le polynôme du 7e degré:

En imposant les contraintes de continuité suivantes,

$q(-T) = A$ $\qquad$ $q(T) = B$

$\dot{q}(-T) = C$ $\qquad$ $\dot{q}(T) = D$

$\ddot{q}(-T) = 0$ $\qquad$ $\ddot{q}(T) = 0$

$q^3(-T) = 0$ $\qquad$ $q^3(T) = 0$

on obtient les coefficients suivants:

$$a_0 = [16(A+B) + 11\cdot(C-D)\cdot T] / 32$$

$$a_1 = -[35\cdot(A-B) + 19\cdot(C+D)\cdot T] / (32\cdot T)$$

$$a_2 = -15\cdot(C-D) / (32\cdot T)$$

$$a_3 = 35\cdot[(A-B) + (C+D)\cdot T] / (32\cdot T^3)$$

$$a_4 = 5\cdot(C-D) / (32\cdot T^3)$$

$$a_5 = -21\cdot[(A-B) + (C+D)\cdot T] / (32\cdot T^5)$$

$$a_6 = -(C-D) / (32\cdot T^5)$$

$$a_7 = 5\cdot[(A-B) + (C+D)\cdot T] / (32\cdot T^5)$$

En développant les valeurs de "B et T" de la même façon que pour le polynôme du 5^e degré, on obtient la solution suivante:

$$B = A + T\cdot(C+D)$$

$$T = 15\cdot(D-C) / (16\cdot\alpha_{max})$$

A.1.3 - Le polynôme du 9^e degré:

En imposant les contraintes de continuité suivantes,

$q(-T) = A$ $\qquad$ $q(T) = B$

$\dot{q}(-T) = C$ $\qquad$ $\dot{q}(T) = D$

$\ddot{q}(-T) = 0$ $\qquad$ $\ddot{q}(T) = 0$

$q^3(-T) = 0$ $\qquad$ $q^3(T) = 0$

$q^4(-T) = 0$ $\qquad$ $q^4(T) = 0$

on obtient les coefficients suivants:

$$a_0 = [128(A+B) + 23\cdot(C-D)\cdot T] / 256$$

$$a_1 = [-105\cdot(A-B) + 23\cdot(C+D)\cdot T] / (256\cdot T)$$

$$a_2 = -35\cdot(C-D) / (64\cdot T)$$

$$a_3 = 105\cdot[(A-B) + (C+D)\cdot T] / (64\cdot T^3)$$

$$a_4 = 35\cdot(C-D) / (128\cdot T^3)$$

$$a_5 = -189\cdot[(A-B) + (C+D)\cdot T] / (128\cdot T^5)$$

$$a_6 = -7\cdot(C-D) / (64\cdot T^5)$$

$$a_7 = 315\cdot[(A-B) + (C+D)\cdot T] / (448\cdot T^7)$$

$$a_8 = 5\cdot(C-D) / (256\cdot T^7)$$

$$a_9 = -35\cdot[(A-B) + (C+D)\cdot T] / (256\cdot T^9)$$

En développant les valeurs de "B et T" de la même façon que pour le polynôme du 5e degré, on obtient la solution suivante:

$$B = A + T\cdot(C+D)$$

$$T = 35\cdot(D-C) / (32\cdot\alpha_{max})$$

A.2 - L'accélération moyenne équivalente du polynôme du 5e degré:

Pour calculer l'accélération constante équivalente à celle d'un polynôme du 3e degré, il faut calculer la moyenne de l'accélération en prenant la formule bien connue de la moyenne:

$$q_c = \frac{\int_{-T}^{T} q(t)\cdot dt}{2\cdot T}$$

$$= \frac{2}{3}\cdot\alpha_{ijk}$$

Cette accélération constante de "2·α/3" produit le même accroissement de vitesse (D-C) que la transition utilisée dans la présente thèse.

APPENDICE B: La modélisation physique du robot ASEA.

B.1 - Les systèmes d'axes:

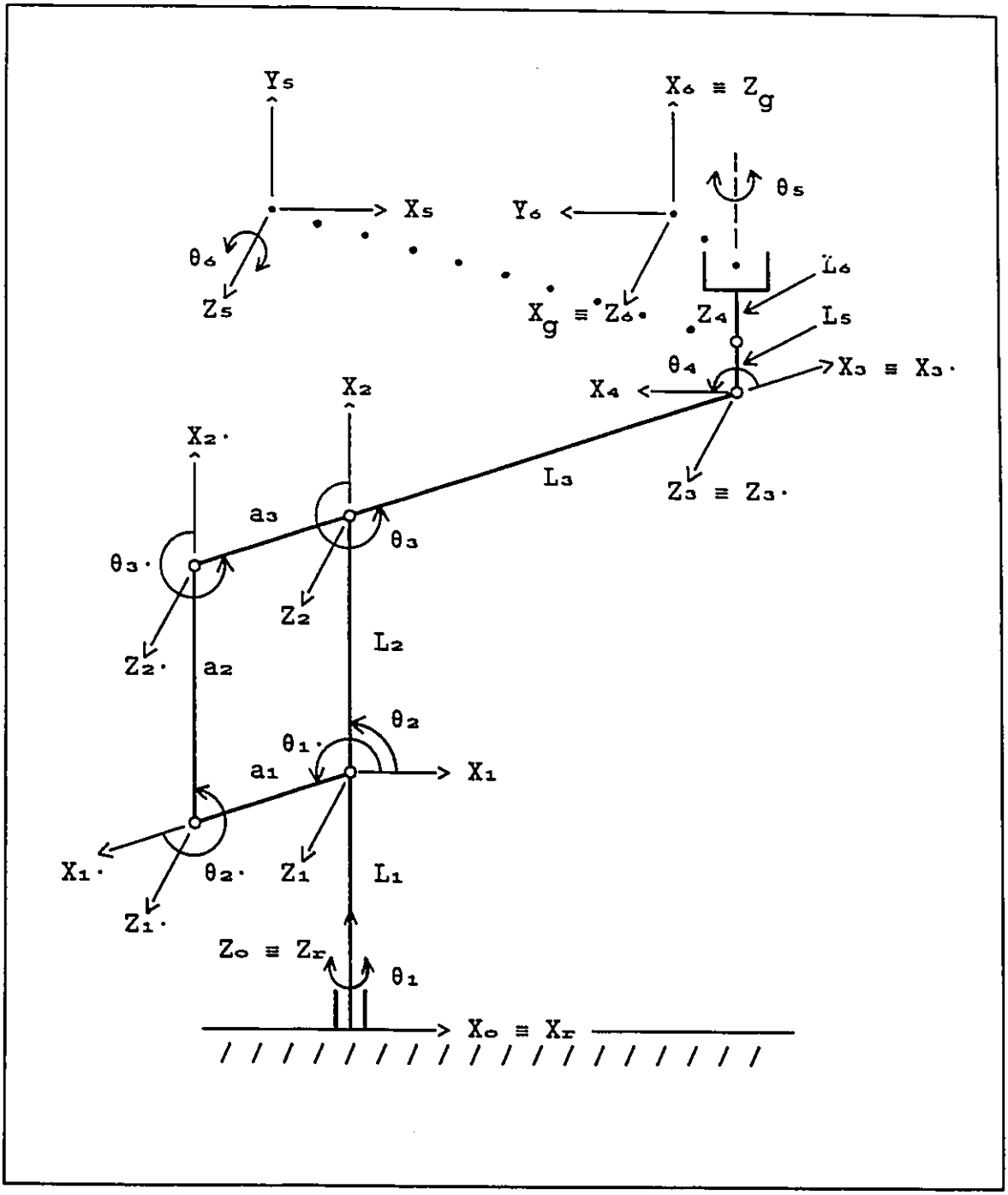


Fig.A.1 - Les systèmes d'axes sur le robot ASEA.

B.2 - Les caractéristiques physiques du robot ASEA:

Les caractériques physiques n'étant pas disponiples, elles ont été approximées ou mesurées. Les tableaux suivants contiennent les longueurs, les masses, les inerties , les limites articulaires et certaines valeurs pour l'algorithme de l'interpolation linéaire articulaire.

TABLEAU A.2

LES LONGUEURS CARACTERISTIQUES.

lien #	L_i (m.)	a_i (m.)	Lc_i (m)
1	0,69		0,345
2	0,69		0,345
3	0,67		0,265
4	0,00		0,000
5	0,10		0,025
6	0,20		0,100
---	---	---	---
1'		0,14	0,350
2'		0,69	0,345
3'		0,14	

TABLEAU A.3

LES MASSES ET LES INERTIES.

lien #	m_i (Kg)	Ixx_i (Kg·m²)	Iyy_i (Kg·m²)	Izz_i (Kg·m²)
1	60,0	2,2000	2,1000	0,7000
2	16,0	0,0600	0,6500	0,6400
3	12,0	0,0500	0,6300	0,6200
4	0,00	0,0000	0,0000	0,0000
5	2,75	0,0060	0,0015	0,0050
6	4,00	0,0020	0,0150	0,0150
--	----	------	------	------
1'	16,0	0,0500	0,0750	0,0950
2'	1,75	0,0005	0,0700	0,0700

TABLEAU A.4

LES LIMITES ARTICULAIRES.

positions (deg)	vitesses (rad./s)	moments (N-m)
$-170 \leq \theta_1 \leq 170$	$-1,75 \leq \dot{\theta}_1 \leq 1,75$	$-150 \leq \tau_1 \leq 150$
$50 \leq \theta_2 \leq 130$	$-1,75 \leq \dot{\theta}_2 \leq 1,75$	$-300 \leq \tau_2 \leq 300$
$230 \leq \theta_3 \leq 295$	$-1,75 \leq \dot{\theta}_3 \leq 1,75$	$-300 \leq \tau_3 \leq 300$
$0 \leq \theta_4 \leq 180$	$-1,75 \leq \dot{\theta}_4 \leq 1,75$	$-50 \leq \tau_4 \leq 50$
$-180 \leq \theta_5 \leq 180$	$-1,75 \leq \dot{\theta}_5 \leq 1,75$	$-25 \leq \tau_5 \leq 25$
$60 \leq \theta_6 \leq 120$	$-1,75 \leq \dot{\theta}_6 \leq 1,75$	$-25 \leq \tau_6 \leq 25$

TABLEAU A.5

VALEURS DIVERSES.

symbole	valeur	unité
M	2,0	Kg
T_{acc}	0,2	sec.

APPENCICE C: La cinématique du robot ASEA.

C.1 - La cinématique directe de position:

A l'annexe A.2, on a vu que l'équation de la cinématique directe de position se présente comme suit:

$$X = Z \cdot T \cdot G \qquad (A.2)$$

Le robot constitue une chaîne cinématique. Les différents systèmes d'axes sont reliés en utilisant des transformations homogènes. Pour construire la matrice "T", on fait la multiplication successive de matrices développées à partir de la représentation de Denavit-Hartenberg. Pour le robot ASEA IRB/6, les tables des paramètres et les matrices "A_i" se présentent comme suit:

- chaîne "1":

TABLEAU A.6

LA TABLE DES PARAMETRES: Chaîne "1".

lien #	systèmes d'axes	θ_i	d_i	a_i	α_i
1	1 -> 0	θ_1	L_1	0.	90°
2	2 -> 1	θ_2	0.	L_2	0°
3	3 -> 2	θ_3	0.	L_3	0°
4	4 -> 3	θ_4	0.	0.	90°
5	5 -> 4	θ_5	L_5	0.	90°
6	6 -> 5	θ_6	0.	L_6	0°

Ce qui donne les matrices homogènes suivantes:

$$A_0^1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_1^2 = \begin{bmatrix} C_2 & S_2 & 0 & L_2 \cdot C_2 \\ S_2 & C_2 & 0 & L_2 \cdot S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^3 = \begin{bmatrix} C_3 & -S_3 & 0 & L_3 \cdot C_3 \\ S_3 & C_3 & 0 & L_3 \cdot S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_3^4 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4^5 = \begin{bmatrix} C_5 & 0 & S_5 & 0 \\ S_5 & 0 & -C_5 & 0 \\ 0 & 1 & 0 & L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_5^6 = \begin{bmatrix} C_6 & -S_6 & 0 & L_6 \cdot C_6 \\ S_6 & C_6 & 0 & L_6 \cdot S_6 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- chaîne "2":

TABLEAU A.7

LA TABLE DES PARAMETRES: Chaîne "2".

lien #	systèmes d'axes	θ_i	d_i	a_i	α_i
1	1 -> 0	θ_1 ·	0.	a_1	0°
2	2 -> 1	θ_2 ·	0.	a_2	0°
3	3 -> 2	θ_3 ·	0.	a_3+L_3	0°

Ce qui donne les matrices homogènes suivantes:

$$A_0^{1'} = \begin{bmatrix} C_1 \cdot & -S_1 \cdot & 0 & a_1 \cdot C_1 \cdot \\ S_1 \cdot & C_1 \cdot & 0 & a_1 \cdot S_1 \cdot \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_1^{2'} = \begin{bmatrix} C_2 \cdot & -S_2 \cdot & 0 & a_2 \cdot C_2 \cdot \\ S_2 \cdot & C_2 \cdot & 0 & a_2 \cdot S_2 \cdot \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^{3'} = \begin{bmatrix} C_3' & -S_3' & 0 & (a_3+L_3)\cdot C_3' \\ S_3' & C_3' & 0 & (a_3+L_3)\cdot S_3' \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En utilisant l'équation (A.1), on développe la matrice "T" du manipulateur pour la chaîne "1":

$$T = \begin{bmatrix} \vec{n}_T & \vec{o}_T & \vec{a}_T & \vec{p}_T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

où:

$$\vec{n}_T = \begin{bmatrix} C_1\cdot(C_{234}\cdot C_5\cdot C_6 + S_{234}\cdot S_6) + S_1\cdot S_5\cdot C_6 \\ S_1\cdot(C_{234}\cdot C_5\cdot C_6 + S_{234}\cdot S_6) - C_1\cdot S_5\cdot C_6 \\ S_{234}\cdot C_5\cdot C_6 - C_{234}\cdot S_6 \\ 0 \end{bmatrix}$$

$$\vec{o}_T = \begin{bmatrix} C_1\cdot(-C_{234}\cdot C_5\cdot S_6 + S_{234}\cdot C_6) - S_1\cdot S_5\cdot S_6 \\ S_1\cdot(-C_{234}\cdot C_5\cdot S_6 + S_{234}\cdot C_6) + C_1\cdot S_5\cdot S_6 \\ -S_{234}\cdot C_5\cdot S_6 - C_{234}\cdot C_6 \\ 0 \end{bmatrix}$$

$$\vec{a}_T = \begin{bmatrix} C_1\cdot C_{234}\cdot S_5 - S_1\cdot C_5 \\ S_1\cdot C_{234}\cdot S_5 + C_1\cdot C_5 \\ S_{234}\cdot S_5 \\ 0 \end{bmatrix}$$

$$\vec{p}_T = \begin{bmatrix} C_1[L_6(C_{234}C_5C_6+S_{234}S_6)+L_5S_{234}+L_3C_{23}+L_2C_2]+S_1(L_6S_5C_6) \\ S_1[L_6(C_{234}C_5C_6+S_{234}S_6)+L_5S_{234}+L_3C_{23}+L_2C_2]-C_1(L_6S_5C_6) \\ L_6(S_{234}C_5C_6-C_{234}S_6) - L_5C_{234} + L_3S_{23} + L_2S_2 + L_1 \\ 1 \end{bmatrix}$$

Maintenant, il ne reste que "Z et G" à trouver. De par la figure A.1, il est évident que:

$$Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En utilisant l'équation (A.2), on trouve "X".

C.2 - <u>La cinématique inverse de position</u>:

Ici, on veut trouver la position articulaire du robot pour une position cartésienne "X" donnée:

$$X = \begin{bmatrix} \vec{n} & \vec{o} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- chaîne "1":

$$\theta_1 = \text{Atan2}\,(P_y - a_y \cdot L_6 \; ; \; P_x - a_x \cdot L_6)$$

ou

$$\theta_1 = \theta_1 + \pi \quad \text{(solution rejetée)}$$

$$\theta_6 = \text{Atan2}\,(o_x \cdot S_1 - o_y \cdot C_1 \; ; \; a_x \cdot S_1 - a_y \cdot C_1)$$

ou

$$\theta_6 = \theta_6 + \pi$$

$$\theta_5 = \text{Atan2}\,(n_z \; ; \; a_z \cdot C_6 + o_z \cdot S_6)$$

$$\theta_3 = \text{Atan2}\left(-\sqrt{1 - C_3^2}\,;\, C_3\right)$$

où;

$$C_3 = \frac{E^2 + F^2 - L_2^2 - L_3^2}{2 \cdot L_2 \cdot L_3}$$

$$E = C_1 \cdot [-L_5 \cdot (a_x \cdot S_6 - o_x \cdot C_6) - L_6 \cdot a_x + p_x] + S_1 \cdot [-L_5 \cdot (a_y \cdot S_6 - o_y \cdot C_6) - L_6 \cdot a_y + p_y]$$

$$F = -L_5 \cdot (a_z \cdot S_6 - o_z \cdot C_6) - L_6 \cdot a_z + p_z - L_1$$

$$\theta_2 = \beta - \text{Atan2}\left(E\,;\, \pm\sqrt{R^2 - E^2}\right)$$

où;

$$\beta = \text{Atan2}\left(L_3 \cdot C_3 + L_2\,;\, L_3 \cdot S_3\right)$$

$$R^2 = L_2^2 + L_3^2 + 2 \cdot L_2 \cdot L_3 \cdot C_3$$

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3$$

où;

$$\theta_{234} = \text{Atan2}\left(N_y\,;\, N_x\right)$$

$$N_x = C_1[C_5(a_x C_6 + o_x S_6) + n_x S_5] + S_1[C_5(a_y C_6 + o_y S_6) + n_y S_5]$$

$$N_y = C_5 \cdot (a_z \cdot C_6 + o_z \cdot S_6) + n_z \cdot S_5$$

- chaîne "2":

La boucle fermée du robot ASEA IRB/6 forme un parallélogramme; la relation entre les chaînes "1 et 2" est donc linéaire.

$$\theta_1' = \theta_2 + \theta_3 - \pi$$

$$\theta_2' = 3 \cdot \pi - \theta_3$$

$$\theta_3' = \theta_3$$

APPENDICE D: La matrice jacobienne et sa dérivée.

La matrice jacobienne "J" permet de relier l'espace vitesse cartésienne à l'espace vitesse articulaire. Les résultats ont été développés en utilisant la méthode proposée par ASADA [2]. De plus, on présentera la dérivée de la matrice jacobienne.

D.1 - La matrice jacobienne:

Pour simplifier la présentation de la matrice, on utilise des variables de remplacement "A et B" :

$$A_1 = L_6 \cdot (C_{234} \cdot C_5 \cdot C_6 + S_{234} \cdot S_6)$$

$$A_2 = L_5 \cdot S_{234}$$

$$A_3 = L_3 \cdot C_{23}$$

$$A_4 = L_2 \cdot C_2$$

$$A_5 = L_6 \cdot S_5 \cdot C_6$$

$$A_6 = L_6 \cdot (-S_{234} \cdot C_5 \cdot C_6 + C_{234} \cdot S_6)$$

$$A_7 = L_5 \cdot C_{234}$$

$$A_8 = -L_3 \cdot S_{23}$$

$$A_9 = -L_2 \cdot S_2$$

$$A_{10} = -L_6 \cdot C_{234} \cdot S_5 \cdot C_6$$

$$A_{11} = L_6 \cdot C_5 \cdot C_6$$

$$A_{12} = L_6 \cdot (-C_{234} \cdot C_5 \cdot S_6 + S_{234} \cdot C_6)$$

$$A_{13} = -L_6 \cdot S_5 \cdot S_6$$

$$A_{14} = -L_6 \cdot S_{234} \cdot S_5 \cdot C_6$$

$$A_{15} = L_6 \cdot (-S_{234} \cdot C_5 \cdot S_6 - C_{234} \cdot C_6)$$

$$A_{16} = -L_6 \cdot C_{234} \cdot C_5 \cdot C_6$$

$$A_{17} = L_6 \cdot C_5 \cdot S_6$$

$$A_{18} = L_6 \cdot C_{234} \cdot S_5 \cdot S_6$$

$$A_{19} = L_6 \cdot S_{234} \cdot C_5 \cdot C_6$$

$$A_{20} = L_6 \cdot S_{234} \cdot S_5 \cdot S_6$$
$$A_{21} = C_{234} \cdot S_5$$
$$A_{22} = S_{234} \cdot S_5$$
$$A_{23} = C_{234} \cdot C_5$$
$$A_{24} = S_{234} \cdot C_5$$
$$B_i = A_{i-1} + A_i$$

Soit "J_i" représentant la colonne "i" de la matrice jacobienne. La matrice "J" est présentée dans le tableau (A.8).

TABLEAU A.8

LA MATRICE JACOBIENNE.

J_1	J_2	J_3
$-S_1 \cdot B_4 + C_1 \cdot A_5$	$C_1 \cdot B_9$	$C_1 \cdot B_8$
$C_1 \cdot B_4 + S_1 \cdot A_5$	$S_1 \cdot B_9$	$S_1 \cdot B_8$
0.	B_4	B_3
0.	S_1	S_1
0.	$-C_1$	$-C_1$
1.	0.	0.

J_4	J_5	J_6
$C_1 \cdot B_7$	$S_1 \cdot A_{11} + C_1 \cdot A_{10}$	$S_1 \cdot A_{13} + C_1 \cdot A_{12}$
$S_1 \cdot B_7$	$-C_1 \cdot A_{11} + S_1 \cdot A_{10}$	$-C_1 \cdot A_{13} + S_1 \cdot A_{12}$
B_2	A_{14}	A_{15}
S_1	$C_1 \cdot S_{234}$	$C_1 \cdot A_{21} - S_1 \cdot C_5$
$-C_1$	$S_1 \cdot S_{234}$	$S_1 \cdot A_{21} + C_1 \cdot C_5$
0.	$-C_{234}$	A_{22}

D.2 - La dérivée de la matrice jacobienne:

$$K_1 = B_9 \cdot \dot{\theta}_2 + B_8 \cdot \dot{\theta}_3 + B_7 \cdot \dot{\theta}_4 + A_{10} \cdot \dot{\theta}_5 + A_{12} \cdot \dot{\theta}_6$$

$$K_2 = A_5 \cdot \dot{\theta}_1 + K_1$$

$$K_3 = -B_4 \cdot \dot{\theta}_1 + A_{11} \cdot \dot{\theta}_5 + A_{13} \cdot \dot{\theta}_6$$

$$K_4 = B_4 \cdot \dot{\theta}_2 + B_3 \cdot \dot{\theta}_3 + B_2 \cdot \dot{\theta}_4 + A_{14} \cdot \dot{\theta}_5 + A_{15} \cdot \dot{\theta}_6$$

$$K_5 = B_3 \cdot \dot{\theta}_{23} + B_2 \cdot \dot{\theta}_4 + A_{14} \cdot \dot{\theta}_5 + A_{15} \cdot \dot{\theta}_6$$

$$K_6 = A_8 \cdot \dot{\theta}_{23} + B_7 \cdot \dot{\theta}_4 + A_{10} \cdot \dot{\theta}_5 + A_{12} \cdot \dot{\theta}_6$$

$$K_7 = B_2 \cdot \dot{\theta}_{234} + A_{14} \cdot \dot{\theta}_5 + A_{15} \cdot \dot{\theta}_6$$

$$K_8 = B_7 \cdot \dot{\theta}_{234} + A_{10} \cdot \dot{\theta}_5 + A_{12} \cdot \dot{\theta}_6$$

$$K_9 = A_{10} \cdot \dot{\theta}_1 + A_5 \cdot \dot{\theta}_5 + A_{17} \cdot \dot{\theta}_6$$

$$K_{10} = A_{11} \cdot \dot{\theta}_1 - A_{14} \cdot \dot{\theta}_{234} + A_{16} \cdot \dot{\theta}_5 + A_{18} \cdot \dot{\theta}_6$$

$$K_{11} = A_{10} \cdot \dot{\theta}_{234} - A_{19} \cdot \dot{\theta}_5 + A_{20} \cdot \dot{\theta}_6$$

$$K_{12} = S_{234} \cdot \dot{\theta}_1$$

$$K_{13} = C_{234} \cdot \dot{\theta}_{234}$$

$$K_{14} = A_{12} \cdot \dot{\theta}_1 + A_{17} \cdot \dot{\theta}_5 + A_5 \cdot \dot{\theta}_6$$

$$K_{15} = A_{13} \cdot \dot{\theta}_1 - A_{15} \cdot \dot{\theta}_{234} + A_{18} \cdot \dot{\theta}_5 - A_1 \cdot \dot{\theta}_6$$

$$K_{16} = A_{12} \cdot \dot{\theta}_{234} + A_{20} \cdot \dot{\theta}_5 + A_6 \cdot \dot{\theta}_6$$

$$K_{17} = A_{21} \cdot \dot{\theta}_1 - S_5 \cdot \dot{\theta}_5$$

$$K_{18} = -C_5 \cdot \dot{\theta}_1 - A_{22} \cdot \dot{\theta}_{234} + A_{23} \cdot \dot{\theta}_5$$

$$K_{19} = A_{21} \cdot \dot{\theta}_{234} + A_{24} \cdot \dot{\theta}_5$$

TABLEAU A.9

L'INVERSE DE LA MATRICE JACOBIENNE.

J_1^{-1}	J_2^{-1}	J_3^{-1}
$C_1 \cdot K_3 - S_1 \cdot K_2$	$-S_1 \cdot B_9 \cdot \dot{\theta}_1 - C_1 \cdot K_4$	$-S_1 \cdot B_8 \cdot \dot{\theta}_1 - C_1 \cdot K_5$
$S_1 \cdot K_3 + C_1 \cdot K_2$	$C_1 \cdot B_9 \cdot \dot{\theta}_1 - S_1 \cdot K_4$	$C_1 \cdot B_8 \cdot \dot{\theta}_1 - S_1 \cdot K_5$
0.	K_1	K_6
0.	$C_1 \cdot \dot{\theta}_1$	$C_1 \cdot \dot{\theta}_1$
0.	$S_1 \cdot \dot{\theta}_1$	$S_1 \cdot \dot{\theta}_1$
0.	0.	0.

J_4^{-1}	J_5^{-1}	J_6^{-1}
$-S_1 \cdot B_7 \cdot \dot{\theta}_1 - C_1 \cdot K_7$	$C_1 \cdot K_{10} - S_1 \cdot K_9$	$C_1 \cdot K_{15} - S_1 \cdot K_{14}$
$C_1 \cdot B_7 \cdot \dot{\theta}_1 - S_1 \cdot K_7$	$S_1 \cdot K_{10} + C_1 \cdot K_9$	$S_1 \cdot K_{15} + C_1 \cdot K_{14}$
K_8	K_{11}	K_{16}
$C_1 \cdot \dot{\theta}_1$	$C_1 \cdot K_{13} - S_1 \cdot K_{12}$	$C_1 \cdot K_{18} - S_1 \cdot K_{17}$
$S_1 \cdot \dot{\theta}_1$	$S_1 \cdot K_{13} + C_1 \cdot K_{12}$	$S_1 \cdot K_{18} + C_1 \cdot K_{17}$
0.	$S_{234} \cdot \dot{\theta}_{234}$	K_{19}

APPENDICE E: Le programme informatique.

```
C*************************************************************************
C     PROGRAMME PRINCIPAL:  Simulation d'un robot ASEA.                  *
C                           Algorithme de linearisation.                 *
C     Resultats numeriques:  fichier FORT7 et FORT8.                     *
C     par:  Mario Tetreault, Mai 1988.                                   *
C*************************************************************************
C
      call INITIA

      ICHOIX =  0
10    IF (ICHOIX.NE.3) THEN
         call INITGR(1)
         call CLOSEG
C       Ecriture du menu principal.
         write (6,1000)
         read (5,*) ICHOIX
         if (ICHOIX.eq.1) then
C          A -  Entree des parametres de simulation:  donnees d'entree.
            call PARAME
         endif
         if (ICHOIX.eq.2) then
C          A -  Inverse cinematique:
            call INITGR(1)
            call CLOSEG
            call CINKI
C          B -  Planification de la trajectoire:
            call CTRPL
C          C -  La dynamique inverse:
            call CDYNA
C          D -  Les resultats sous forme graphique.
            call PLOTTR
         endif
         goto 10
      endif

1000  format(/'1',1X,77(' ')/' /',77X,'\'/33X,'MENU PRINCIPAL.'/' \',
     1       77('_'),'/'/7/6X,'1 -  changer certains PARAMETRES de ',
     2       'simulation.'//6X,'2 -  faire la SIMULATION.'//6X,
     3       '3 -  FIN:  retour au DOS.'/1X,78('_')///6X,
     4       '----> Entrez un chiffre (1 a 3): ')

      stop
      end
```

```
C**********************
      BLOCK data
C**********************
C     Valeurs par defaut:  les donnees sont entrees colonne par colonne.

C-------------------------------------------------------------------------
C  Variables:                                                             |
C     i           :  refere au joint 'i' (i =  1, N).                     |
C     j           :  refere au segment 'j' (j =  1, M).                   |
C                 :  refere aussi au point 'j' (j =  1, M+1).             |
C     k           :  partie 'k' d'un segment.  Un segment peut-etre       |
C                    divise en 6 sous-segments.                           |
C     l           :  refere a la chaine du robot (l =  1, 2).             |
C     M           :  nombre total de segments (M =  7 segments max.).     |
C     N           :  nombre de degre de liberte du robot ASEA (N =  6).   |
C     XL(i,l)     :  longueur des liens (metres) .                        |
C     ALPHA(8,2)  :  "twist angle" dans la representation d'Hartenberg-   |
C                    Denavitt.  Pour tout robot,  ALPHA(1,1) =  ALPHA(8,1)|
C                    =  0.  Les indices representes:  ALPHA(i+1,1)        |
C     TMAX(i,2)   :  angles limites pour les joints (degres).             |
C                    (i,1) ->  angle minimum.                             |
C                    (i,2) ->  angle maximum.                             |
C     XYT(6,j)    :  coordonnees des points par rapport au systeme de     |
C                    reference:  (1,j) coordonnee en 'X' (metres).        |
C                                (2,j) coordonnee en 'Y' (metres).        |
C                                (3,j) coordonnee en 'z' (metres).        |
C                                (4,j) angle autour de 'X' (degres).      |
C                                (5,j) angle autour de 'Y' (degres).      |
C                                (6,j) angle autour de 'Z' (degres).      |
C     VAL(IS,2)   :  'IS' represente la 'IS' ieme ligne dans le programme |
C                    ecrit en VAL.  VAL (IS,1) contient un numero corres- |
C                    pondant a une commande: (1->MOVEA, 2->MOVEL, 3->STOP)|
C                    VAL (IS,2) contient une valeur associee.             |
C     NS          :  nombre de lignes dans le programme ecrit en VAL.     |
C     NTP         :  nombre de VIA points.                                |
C     TT(i,j,1)   :  angles correspondant aux points de visite (radians). |
C     TDM(i)      :  approximation de la vitesse angulaire maximale (r/s).|
C     ADM(i)      :  approximation de l'acceleration angulaire maximale.  |
C     XJERK(i)    :  limite sur les impulsions angulaires (jerk).         |
C     TORMAX(i)   :  limite sur les moments de torsion pour les moteurs.  |
C     TINC        :  temps d'echantillonnage (sec).                       |
C     INC         :  numero d'echantillon plus un.                        |
C     NPS         :  nombre total d'increments moins un.                  |
C     XMASS(i,j)  :  masse des liens (Kg).  XMASS(7,1) contient la        |
C                    de la charge utile transportee.                      |
C     XINERT(3,i,1): matrice d'inertie (Ixx,Iyy,Izz en Kg*m*m).           |
C     XLC(i,1)    :  position du centre de masse du lien 'i' par rapport  |
C                    au joint 'i'.                                        |
C     ZX(6,3,200):   6   ->  X,Y,Z,TX,TY,TZ.                             |
C                    3   ->  position, vitesse et acceleration.           |
C                    200 ->  increments.                                  |
C     Z1(6,3,200):   6   ->  T1,T2,T3,T4,T5,T6.                          |
C                    3   ->  position, vitesse et acceleration.           |
C                    200 ->  increments.                                  |
```

```
C     Z2(3,3,200):  3   ->  T1',T2',T3'.
C                   3   ->  position, vitesse et acceleration.
C                   200 ->  increments.
C
C  - Planification de la trajectoire:
C     BTIME(NBT) :  temps au point de visite (sec).  NBT est le nombre
C                   de segments.
C     RT         :  Temps residuel d'un sous-segment precedent (sec).
C     TIME       :  Temps courant accumule depuis le debut (sec).
C     TLOCAL     :  Temps local pour un sous-segment (sec).
C                   - T(i,j,k)  <  TLOCAL  <  T(i,j,k).
C     IM         :  refere au type de commande dans le programme VAL.
C     E          :  facteur de forme pour la trajectoire cartesienne
C                   'pseudo-ideale'.  En fait, ce facteur represente un
C                   pourcentage (%) de la longueur cartesienne d'un
C                   segment (0 < E < 1).
C     A(i,j,k)   :  angles (radians).
C     C(i,j,k)   :  vitesses angulaires (rad/sec).
C     T(i,j,k)   :  mi-temps pour chaque segment.
C     VMAX(i,M,k):  vitesses angulaires maximales (rad/s).
C     AMAX(i,M,k):  accelerations angulaires (rad/s/s).
C     V(6,2)     :  pour le segment 'j', il contient les vecteurs vites-
C                   se orientation Vj et Vj+1 (vecteur normalise).
C     EW(6,M+1)  :  orientation angulaire aux points de visites pour le
C                   porteur et l'organe terminal (radians).
C     PLG(6,M+1) :  point de linearisation pour le porteur et l'organe
C                   terminal (position en metres).
C     THETA      :  angle reel de deviation (radians).
C     BETA       :  angle associe a la limite de deviation (radians).
C     X(i,2)     :  vitesses angulaires moyennes au segment 'j'.  Cette
C                   matrice contient:  Xij et Xij+1 (rad/s).
C     DELTA(M+1) :  facteur de correction pour 'KL'.
C     RK         :  norme des vitesses angulaires au point de visite
C                   (rad/s).
C     LPOINT(M)  :  variable logique qui indique l'ajout d'un point de
C                   linearisation:  1 --->  pour le porteur.
C                                   2 --->  pour le poignet.
C     AL(i,j)    :  angle correspondant au point de linearisation (rad).
C     TL(i)      :  angle avant la linearisation (rad.).
C     DLL(j)     :  distance minimale entre le point de linearisation et
C     DOO(j)        la droite reliant les points 'j et j+1' (metres).
C     NOMBRE     :  nombre de joints qui font que le decouplement de la
C                   trajectoire est appliquee (NOMBRE =  1, 3).
C     TS(NTS)    :  discretisation du temps en 200 points.
C     TN(i,2)    :  TN(i,1) -> torque minimum apres correction.
C                   TN(i,2) -> torque maximum apres correction.
C     C2(i)      :  'C' au carree dans l'algorithme de Hollerbach.
C     KT(i)      :  segment du lien 'i' relie au temps 'TS'.
C     ERROR      :  critere d'optimisation:  pourcentage d'erreur tolere
C                   pour l'optimisation de la trajectoire(0 < ERROR < 1).
C
C  - La dynamique pour un robot avec une chaine fermee.:  Luh and Zheng.
C     XLAMDA(2)  :  multiple de Lagrange (2 constraintes).
C     DC1(2,2)   :  matrice inverse utilisee pour trouver XLAMDA.
```

```
C     DC(6,2)     :  six premiere ligne dans la matrice des contraintes.
C     TFM(2)      :  moment de torsion sans moteur.
C     TF(6)       :  moment de torsion avec moteur.
C     TORQUE(6,200):moment de torsion pour chaque increment.
C
C  - Algorithme avec calculs recurrants:  Luh, Walker and Paul.
C     P(3,7,2)    :  vecteur 'P' (metres).
C                    3 ---> espace en trois dimensions.
C                    7 ---> six liens.  (2 a 7) correspond aux liens
C                           (1 a 6).
C                    2 ---> deux chaines.
C     S(3,7,2)    :  vecteur 'S' (metres).
C     RI(3,7,2)   :  matrice d'inertia pour chaque chaine.
C                    3 ---> (Ixx,Iyy,Izz)  kg/m/m.
C     RM(7,2)     :  masses pour chaque chaine.
C     TF1(7)      :  moment de torsion pour la chaine 1.
C     TF2(4)      :  moment de torsion pour la chaine 2.
C
C  - Routine graphique:
C     CVISIT      :  dessine la trajectoire (defaut:  Oui).
C     CPROJ       :  projection des points de visite sur le plan
C                    horizontal (defaut:  Non).
C     CSURF       :  projection de la trajectoire a l'aide de surface
C                    (defaut:  Oui).
C     CROB        :  representation du robot a l'aide de ligne (defaut:
C                    Non).
C     CARM        :  la trajectoire du porteur sera dessinee.  (defaut:
C                    Oui).
C     CEFFAC      :  apres chaque position dessinee, le robot est efface
C                    (defaut:  Non).
C     IPNTY       :  nombre de carre pour la projection de surface (10).
C     TWAIT       :  temps d'arret entre chaque dessin (1 sec.).
C     VISION(3)   :  axe de vision (-1.,-1.,-1.)
C     RHO         :  distance de l'observateur par rapport l'origine
C                    (8 metres).
C     FOCUS       :  distance focale (4 metres.).
C
C-----------------------------------------------------------------------
      character CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1            VAL(20,2),NS,NTP
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
      common/INERTI/XMASS(7,2),XINERT(3,6,2),XLC(6,2)
      common/INITI/CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT
      common/VISION/VISION(3),RHO,FOCUS
      common/CSTEP3/NOMBRE
      data XL/.69,.69,.67,0.,.1,.2, .14,.69,.14,3*0./
      data XYT/0.715, 0.  , 0.85,   0., 135. ,  0.,
     1         0.8  , 0.  , 1.74,   0.,  79.5,  0.,
     2         1.02 , 0.  , 1.22,   0., 135. ,  0.,
     3         0.   , 1.02, 1.22,   0., 135. , 90., 24*0./
```

```
      data VAL/2.,2.,2.,2.,3.,15*0., 1.,2.,3.,4.,16*0./
      data NS,NTP,NPS/5,4,200/
      data TDM,ADM,TORMAX,E/6*1.75, 6*6.5, 150.,300.,300.,50.,2*25., .1/
      data XJERK/6*1.E20/
      data TMAX/-170.,50.,230.,0.,0.,60., 170.,130.,295.,180.,360.,120./
      data ZX,Z1,Z2,ERROR/9000*0.,.05/
      data INC,TTOTAL,TINC/1,6.62,.0353/
      data XMASS/60.,16.,12.,0.,2.75,4.,2., 16.,1.75,5*0./
      data XINERT/2.2,2.1,.7, .06,.65,.64, .05,.63,.62, 3*0.,
     1            .006,.0015,.005, .002,.015,.015, .05,.075,.095,
     2            .0005,.07,.07, 12*0./
      data XLC/2*.345,.265,0.,.025,0.1, .35,.345,4*0./
      data CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC/'Oui',
     1     'Non','Oui','Non','Non','Non'/
      data IPNTY,TWAIT/10,1./
      data VISION,RHO,FOCUS/-1.,-1.,-1., 8., .5/
      data NOMBRE/7/

      end
C
C*****************************
      subroutine PARAME
C*****************************
C  Cette routine gere le changement des parametres (caracteristiques du robot,
C  programme en VAL).

      character*6 CCOM(3), CREP*1
      integer NUMBER(6)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/INERTI/XMASS(7,2),XINERT(3,6,2),XLC(6,2)
      common/CSTEP3/NOMBRE

      data CCOM/'MOVEA ','MOVEL ','STOP  '/
      data NUMBER/1,2,3,4,5,6/

      PI   =  ACOS(-1.)
      FACT =  180. / PI

      ICHOIX =  0

5     if (ICHOIX.NE.16) then
         call INITGR(1)
         call CLOSEG
         write (6,500)
         write (6,501)
         write (6,502)
         write (6,503)
         read (5,*) ICHOIX
         call INITGR(1)
         call CLOSEG
```

```
C       A.- . La longueur des liens:
        if (ICHOIX.EQ.1) then
           write (6,900)
           write (6,1000) NUMBER(1)
           write (6,1010) (I,XL(I,1),I=1,6)
           write (6,1000) NUMBER(2)
           write (6,1020) (I,XL(I,2),I=1,3)
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,1030)
              read (5,*) (XL(I,1),I=1,6)
              write (6,1040)
              read (5,*) (XL(I,2),I=1,3)
           endif
        endif

C       B -  La modification de la limite des angles.
        if (ICHOIX.EQ.2) then
           write (6,2000)
           do 10 I =  1, 6
              T1 =  TMAX(I,1) * FACT
              T2 =  TMAX(I,2) * FACT
              write (6,2010) T1,I,T2
10         continue
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              do 20 I =  1, 6
                 write (6,2020) I,I,I
                 read (5,*) TMAX(I,1),TMAX(I,2)
20            continue
              call CONV1
           endif
        endif

C       C -  Modification du programme en VAL.
        if (ICHOIX.EQ.3) then
C         Ecriture du programme en VAL:
          CREP =  'o'
32        if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
             write (6,3000)
             do 30 I = 1, NS
                IM  =  INT(VAL(I,1))
                if (IM.eq.1 .or.IM.eq.2) then
                   IS =  INT(VAL(I,2))
                   write (6,3020) CCOM(IM), IS
                endif
                if (IM.eq.3) then
                   write (6,3030) CCOM(IM)
                endif
30           continue
             write (6,9000)
             read (5,9010) CREP
```

```
                if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
C                 Demande du nouveau programme en VAL:
31                 write (6,3040)
                   IS  =  0
                   IM  =  0
                   NTP =  0
40                 if (IM.NE.3) then
                      IS =  IS + 1
                      write (6,3050)
                      read (5,*) VAL(IS,1)
                      IM =  INT(VAL(IS,1))
                      if (IM.gt.3 .or. IM.lt.1) IM =  3
                      if (IM.eq.1 .or. IM.eq.2) then
41                       write (6,3060) CCOM(IM)
                         read (5,*) VAL(IS,2)
                         NEW =  INT (VAL(IS,2))
                         if (NEW.gt.8 .or. NEW.lt.1) goto 41
                         NTP =  MAXO (NTP,NEW)
                      else
                         write (6,3065) CCOM(IM)
                      endif
                      GO TO 40
                   endif
                   if (IS.eq.1) goto 31
                   NS =  IS
                endif
                goto 32
             endif
          endif

          if (ICHOIX.EQ.3) then
C           Entree des points de visite:
             CREP =  'o'
42           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
                write (6,3066)
                do 45 I =  1, NTP
                   write(6,3067)I,(XYT(J,I),J=1,3),(XYT(J,I)*FACT,J=4,6)
45              continue
                write (6,9000)
                read (5,9010) CREP
                if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
                   write (6,3070)
                   write (6,3080)
                   do 50 I =  1, NTP
                      write (6,3090) I
                      read (5,*) (XYT(J,I),J=1,6)
50                 continue
                   call CONV2
                endif
                goto 42
             endif
          endif
```

```
C       D.- . Approximation des vitesses angulaires maximales:
        if (ICHOIX.EQ.4) then
           write (6,4000) (TDM(I),I=1,6)
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,4010)
              read (5,*) (TDM(I),I=1,6)
           endif
        endif

C       E -  Approximation des accelerations angulaires maximums:
        if (ICHOIX.EQ.5) then
           write (6,5000) (ADM(I),I=1,6)
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,5010)
              read (5,*) (ADM(I),I=1,6)
           endif
        endif

C       F -  Les impulsions maximales (jerk):
        if (ICHOIX.EQ.6) then
           write (6,5500) (XJERK(I),I=1,6)
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,5510)
              read (5,*) (XJERK(I),I=1,6)
           endif
        endif

C       G -  Les moments de torsions maximums:
        if (ICHOIX.EQ.7) then
           write (6,6000) (TORMAX(I),I=1,6)
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,6010)
              read (5,*) (TORMAX(I),I=1,6)
           endif
        endif

C       H -  Nombre de points d'increment.
        if (ICHOIX.EQ.8) then
           write (6,7000) NPS
           write (6,9000)
           read (5,9010) CREP
           if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
              write (6,7010)
              read (5,*) NPS
              if (NPS.GT.200)  NPS =  200
           endif
```

```
      endif

C     I -  Nombre du decouplement.
      if (ICHOIX.EQ.9) then
         write (6,7500) NOMBRE
         write (6,9000)
         read (5,9010) CREP
         if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
            write (6,7510)
            read (5,*) NOMBRE
            if (NOMBRE.gt.4)  NOMBRE =  4
            if (NOMBRE.lt.1)  NOMBRE =  1
         endif
      endif

C     J -  Pourcentage de la longueur d'un segment.
      if (ICHOIX.EQ.10) then
         E100 =  E * 100.
         write (6,8000) E100
         write (6,9000)
         read (5,9010) CREP
         if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
            write (6,8010)
            read (5,*) E100
            E =  E100 / 100.
            if (E.lt.0.)  E =  0.
            if (E.gt.1.)  E =  1.
         endif
      endif

C     K -  Critere d'optimisation:  erreur maximale allouee.
      if (ICHOIX.EQ.11) then
         E100 =  ERROR * 100.
         write (6,8050) E100
         write (6,9000)
         read (5,9010) CREP
         if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
            write (6,8010)
            read (5,*) E100
            ERROR =  E100 / 100.
            if (ERROR.lt.0.)  ERROR =  0.
            if (ERROR.gt.1.)  ERROR =  1.
         endif
      endif

C     L -  Les masses des liens:
      if (ICHOIX.EQ.12) then
         write (6,8100)
         write (6,8110)
         do 55 J =  1, 2
            write (6,8120) J
            if (J.EQ.1) then
               K =  6
            else
```

```
                K =  3
             endif
             do 60 I =  1, K
                write (6,8130) I, XMASS(I,J)
60           continue
             if (J.EQ.1) write (6,8140) XMASS(7,J)
             write (6,9000)
             read (5,9010) CREP
             if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
                write (6,8150)
                do 70 I =  1, K
                   write (6,8160) I
                   read (5,*) XMASS(I,J)
70              continue
                if (J.EQ.1) then
                   write (6,8170)
                   read (5,*) XMASS(7,J)
                endif
             endif
55        continue
          endif

C         M -  Les centres de masse:
          if (ICHOIX.EQ.13) then
             write (6,8100)
             write (6,8200)
             do 80 J =  1, 2
                write (6,8120) J
                if (J.EQ.1) then
                   K =  6
                else
                   K =  3
                endif
                do 90 I =  1, K
                   write (6,8130) I, XLC(I,J)
90              continue
                write (6,9000)
                read (5,9010) CREP
                if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
                   write (6,8150)
                   do 100 I =  1, K
                      write (6,8210) I
                      read (5,*) XLC(I,J)
100                continue
                endif
80           continue
          endif

C         N -  Les matrices d'inertie:
          if (ICHOIX.EQ.14) then
             write (6,8100)
             write (6,8500)
             do 110 J =  1, 2
                write (6,8120) J
```

```
            if (J.EQ.1) then
               K =  6
            else
               K =  3
            endif
            do 120 I =  1, K
               write (6,8510) I, (XINERT(L,I,J),L=1,3)
120         continue
            write (6,9000)
            read (5,9010) CREP
            if (CREP.EQ.'o' .OR. CREP.EQ.'O') then
               write (6,8150)
               do 130 I =  1, K
                  write (6,8520) I
                  read (5,*) (XINERT(L,I,J),L=1,3)
130            continue
            endif
110      continue
      endif
C
C     O -  Echo des donnees dans le fichier 8:
      if (ICHOIX.eq.15) then
         open (8,file='FORT8')
            write (6,900)
            write (8,900)
            write (6,1000) NUMBER(1)
            write (8,1000) NUMBER(1)
            write (6,1010) (I,XL(I,1),I=1,6)
            write (8,1010) (I,XL(I,1),I=1,6)
            write (6,1000) NUMBER(2)
            write (8,1000) NUMBER(2)
            write (6,1020) (I,XL(I,2),I=1,3)
            write (8,1020) (I,XL(I,2),I=1,3)
            write (6,9020)
            read (5,9010) CREP
            call INITGR(1)
            call CLOSEG
            write (6,2000)
            write (8,2000)
            do 140 I =  1, 6
               T1 =  TMAX(I,1) * FACT
               T2 =  TMAX(I,2) * FACT
               write (6,2010) T1,I,T2
               write (8,2010) T1,I,T2
140         continue
            write (6,9020)
            read (5,9010) CREP
            call INITGR(1)
            call CLOSEG
C           - le programme en VAL.
            write (6,3000)
            write (8,3000)
            do 150 I =  1, NS
               IM  =  INT(VAL(I,1))
```

```
                  if (IM.eq.1 .or.IM.eq.2) then
                     IS =  INT(VAL(I,2))
                     write (6,3020) CCOM(IM), IS
                     write (8,3020) CCOM(IM), IS
                  endif
                  if (IM.eq.3) then
                     write (6,3030) CCOM(IM)
                     write (8,3030) CCOM(IM)
                  endif
150               continue
                  write (6,3066)
                  write (8,3066)
                  do 155 I =  1, NTP
                     write (6,3067)I,(XYT(J,I),J=1,3),(XYT(J,I)*FACT,J=4,6)
                     write (8,3067)I,(XYT(J,I),J=1,3),(XYT(J,I)*FACT,J=4,6)
155               continue
                  write (6,9020)
                  read (5,9010) CREP
                  call INITGR(1)
                  call CLOSEG
                  write (6,4000) (TDM(I),I=1,6)
                  write (8,4000) (TDM(I),I=1,6)
                  write (6,5000) (ADM(I),I=1,6)
                  write (8,5000) (ADM(I),I=1,6)
                  write (6,5500) (XJERK(I),I=1,6)
                  write (8,5500) (XJERK(I),I=1,6)
                  write (6,6000) (TORMAX(I),I=1,6)
                  write (8,6000) (TORMAX(I),I=1,6)
                  write (6,7000) NPS
                  write (8,7000) NPS
                  write (6,7500) NOMBRE
                  write (8,7500) NOMBRE
                  E100 =  E * 100.
                  write (6,8000) E100
                  write (8,8000) E100
                  E100 =  ERROR * 100.
                  write (6,8050) E100
                  write (8,8050) E100
                  read (5,9010) CREP
C                 - Les masses:
                  call INITGR(1)
                  call CLOSEG
                  write (6,8100)
                  write (6,8110)
                  write (8,8110)
                  do 160 J =  1, 2
                     write (6,8120) J
                     write (8,8120) J
                     if (J.EQ.1) then
                        K =  6
                     else
                        K =  3
                     endif
                     do 170 I =  1, K
```

```
                write (6,8130) I, XMASS(I,J)
                write (8,8130) I, XMASS(I,J)
170             continue
                if (J.EQ.1) then
                   write (6,8140) XMASS(7,J)
                   write (8,8140) XMASS(7,J)
                endif
160             continue
                write (6,9020)
                read (5,9010) CREP
                call INITGR(1)
                call CLOSEG
C               - Les centres de masse:
                write (6,8100)
                write (6,8200)
                write (8,8100)
                write (8,8200)
                do 180 J =  1, 2
                   write (6,8120) J
                   write (8,8120) J
                   if (J.EQ.1) then
                      K =  6
                   else
                      K =  3
                   endif
                   do 190 I =  1, K
                      write (6,8130) I, XLC(I,J)
                      write (8,8130) I, XLC(I,J)
190                continue
180             continue
                write (6,9020)
                read (5,9010) CREP
C               - Les matrices d'inertie:
                call INITGR(1)
                call CLOSEG
                write (6,8100)
                write (6,8500)
                write (8,8100)
                write (8,8500)
                do 200 J =  1, 2
                   write (6,8120) J
                   write (8,8120) J
                   if (J.EQ.1) then
                      K =  6
                   else
                      K =  3
                   endif
                   do 210 I =  1, K
                      write (6,8510) I, (XINERT(L,I,J),L=1,3)
                      write (8,8510) I, (XINERT(L,I,J),L=1,3)
210                continue
200             continue
                write (6,9020)
                read (5,9010) CREP
```

```
            rewind (8)
           close (8)
         endif
       goto 5
       endif

  500  format(/'1 ',77('_')/' /',77X,'\'/21X,'SOUS-MENU:  changement ',
     1       'des parametres.'/' \',77(' '),'/'//6X,'1  -  la LONGUEUR',
     2       ' des liens.'/6X,'2  -  la limite des ANGLES.'/6X,
     3       '3  -  le programme en langage VAL.'/6X,'4  -  approximati',
     4       'on sur les VITESSES maximales angulaires.')
  501  format(6X,'5  -  approximation sur les ACCELERATIONS maximales',
     1       ' angulaires.'/6X,'6  -  les IMPULSIONS maximales.'/
     2        6X,'7  -  les MOMENTS DE TORSION maximums.'/
     3        6X,'8  -  le nombre de points d''INCREMENTATION.'/
     4        6X,'9  -  le nombre du DECOUPLEMENT.')
  502  format(6X,'10 -  le FACTEUR de FORME.'/6X,'11 -  le CRITERE ',
     5       'D''OPTIMISATION (erreur maximale allouee).')
  503  format(6X,'12 -  les MASSES des liens.'/6X,'13 -  les CENTRES ',
     1       'DE MASSE des liens.'/6X,'14 -  les matrices d''INERTIE ',
     2       'des liens.'/6X,'15 -  l''echo des DONNEES.'/6X,'16 -  ',
     3       'FIN:  retour au menu principal.'/1X,78('_')//6X,
     4        '----> Entrez un chiffre (1 a 16):  ')
  900  format(//' LA LONGUEUR DES LIENS:'/' ',21('-')/)
 1000  format(/6X,'- Longueur des liens de la chaine ',I1,':  (metres).')
 1010  format(11X,'L',I1,' =  ',F10.2)
 1020  format(11X,'L',I1,''' =  ',F10.2)
 1030  format(6X,'- CHAINE 1:  L1,L2,L3,L4,L5,L6 (longueur des liens en',
     1       ' metres):  '/11X)
 1040  format(6X,'- CHAINE 2:  L1'',L2'',L3'' (longueur des liens en',
     1       ' metres):  '/11X)
 2000  format(//' VALEUR LIMITE DES POSITIONS ANGULAIRES:'/ ' ',38('-')/)
 2010  format(6X,F10.2,' < θ',I1,' < ',F10.2)
 2020  format(6X,'Angles limites pour theta ',I1,' (θ',I1,' min, θ',I1,
     1       ' max) en degres:  ')
 3000  format(/' PROGRAMME EN VAL:'/' ',16('-')/)
 3010  format(6X,A6,1X,F7.2)
 3020  format(6X,A6,2X,'P',I1)
 3030  format(6X,A6)
 3040  format(//' COMMANDES DE VAL:'/' ',16('-')//
     1 6X,'1 -  MOVEA P1  -   va au point P1 en mouvement angulaire.'/
     2 6X,'2 -  MOVEL P2  -   va au point P2 en ligne plus droite.'/
     3 6X,'3 -  STOP      -   arret du mouvement (fin du programme).'/
     4 6X,'----> Entrez le programme:')
 3050  format(11X,'commande #:  ')
 3060  format('+',6X,A6,2X,'P')
 3065  format('+',6X,A6)
 3066  format(/' LES POINTS DE VISITE:'/' ',20('-')//
     1       16X,'X',9X,'Y',9X,'Z',8X,'TX',8X,'TY',8X,'TZ')
 3067  format(6X,'P',I1,':',6(F10.3))
 3070  format(6X,'- Les points de visite:'/11X,
     1       'Position de la main (coordonnees X,Y,Z):  en metres.')
 3080  format(11X,'Orientation RPY de la main (TX,TY,TZ):  en degres.'/)
 3090  format(11X,'X,Y,Z,TX,TY,TZ pour le point ',I1,':  ')
```

```
 4000  format(//' VITESSES MAXIMALES ANGULAIRES: T1D, T2D, T3D, T4D,',
     1        ' T5D, TD6 rad/s.'/' ',29('-')/6X,6F10.3)
 4010  format(' -  Entrez les vitesses angulaires maximales:  (T1D,',
     1        'T2D,T3D,T4D,T5D,T6D):  rad/s.'/6X)
 5000  format(//' ACCELERATIONS MAXIMALES ANGULAIRES:  A1, A2,',
     1        'A3, A4, A5, A6 rad/s/s.'/' ',34('-')/6X,6F10.3)
 5010  format(6X,'-  Entrez les accelerations angulaires maximales:'/11X,
     1        '(A1,A2,A3,A4,A5,A6):  rad/s/s.'/16X)
 5500  format(//' IMPULSIONS MAXIMALES:  j1, j2, j3, j4, j5, j6',
     1        ' r/sss.'/' ',20('-')/6X,6F10.3)
 5510  format(6X,'-  Entrez les impulsions maximales:'/11X,
     1        '(j1,j2,j3,j4,j5,j6):  r/s/s/s.'/16X)
 6000  format(//' MOMENTS DE TORSION MAXIMUMS:  t1, t2, t3, t4, t5, t6',
     1        ' N-m..'/' ',27('-')/6X,6F10.3)
 6010  format(6X,'-  Entrez les moments de torsion maximums:'/11X,
     1        '(t1,t2,t3,t4,t5,t6):  N-m.'/16X)
 7000  format(//' - LE NOMBRE D''INCREMENTS:  ',I3/'    ',22('-'))
 7010  format(6X,'- Entrez le nombre d''increments (10 < NPS < 200):  ')
 7500  format(//' - LE NOMBRE DU DECOUPLEMENT:  ',I1/'    ',25('-'))
 7510  format(6X,'- Entrez le nouveau nombre (0 < NOMBRE < 5)'/
     1       8X,'(''4'' indique qu''il y a aucun decouplement.)  :')
 8000  format(//' - LE FACTEUR DE FORME:  ',F8.2,' %.'/'    ',19('-'))
 8010  format(6X,'- Entrez le pourcentage d''erreur (0. < % < 100.):  ')
 8050  format(//' - LE CRITERE D''OPTIMISATION (erreur maximale ',
     1        'allouee):  ',F8.2,' %.'/'    ',51('-'))
 8100  format (//)
 8110  format (//' A -  MASSES:  les masses sont en Kg.'/6X,6('-'))
 8120  format (6X,'* Chaine ',I1,':')
 8130  format (11X,'- lien ',I1,'     :  ',F9.4)
 8140  format (11X,'- Masse de la charge utile:',F8.3//)
 8150  format (/6X,'ENTREZ:')
 8160  format (11X,'- Masse du lien ',I1,' (Kg):  ')
 8170  format (11X,'- Masse de la charge utile (Kg):  ')
 8200  format (' B -  CENTRES de MASSE:  all lenghts in meters.'/6X,
     1        16('-'))
 8210  format (11X,'- Position du centre de masse pour le lien ',
     1        I1,' (m):  ')
 8500  format (' C -  MATRICES D''INERTIE:  inerties en Kg/m².'/6X,
     1        18('-')//)
 8510  format (11X,'- lien ',I1,':'/16X,F9.4,2('   0.0000')/16X,
     1        '   0.0000',F9.4,'   0.0000'/16X,2('   0.0000'),F9.4)
 8520  format (11X,'- inertie du lien ',I1,' (Ixx,Iyy,Izz):  ')
 9000  format (/' ----> Desirez-vous faire des changements?  (O/N):  ')
 9010  format(A1)
 9020  format(/6X,'---->  Taper RETURN pour continuer:  ')
       return
       end

C*****************************
       subroutine CTRPL
C*****************************
C  Cette routine gere la planification de la trajectoire.

       real*4 BTIME(30)
```

```
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
      common/TIMES/BTIME,NBT
      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
C
      write (6,500)
C     commandes MOVEA or MOVEL:
      call MOVELA
      NBT =  M
      TTOTAL =  0.
      do 10 J =  1, M
         do 20 K =  1, 6
            TTOTAL =  TTOTAL + 2.*T(1,J,K)
20       continue
         BTIME(J) =  TTOTAL
10    continue
      TINC =  TTOTAL/real(NPS-1)
      INCMIN  =  201
      do 30 I =  1, 6
         INC  =  1
         RT   =  0.
         TIME =  0.
         do 40 J =  1, M
            do 50 K =  1, 6
               ENDTIME =  TIME + 2.*T(I,J,K)
               TIME    =  TIME + RT
               TLOCAL  = -T(I,J,K) + RT
60             if (TIME.le.ENDTIME) then
                  call COEFFI (I,J,K)
                  call SOLVE(Z1(I,1,INC),Z1(I,2,INC),Z1(I,3,INC),TLOCAL)
                  INC    =  INC + 1
                  TIME   =  real(INC-1)*TTOTAL / real(NPS-1)
                  TLOCAL =  TLOCAL + TINC
                  goto 60
               else
                  RT   =  TIME - ENDTIME
                  TIME =  ENDTIME
               endif
50          continue
40       continue
         INCMIN  =  MIN0 (INCMIN,INC)
30    continue
      NPS =  INCMIN - 1
      do 70 INC =  1, NPS
         call CHAIN2 (Z2,Z1,INC)
         call FORKIN (ZX(1,1,INC),Z1(1,1,INC))
         call FORVA
70    continue

C  C  Echo des resultats:
C  - Positions:
      write (7,1000) TTOTAL
      write (7,1050)
```

```
      write (7,1100)
      do 200 IS =  1, NPS
         write (7,1200) IS,(ZX(J,1,IS),J=1,6)
200   continue
      write (7,1050)
      write (7,1300)
      do 210 IS =  1, NPS
         write (7,1200) IS,(Z1(J,1,IS),J=1,6)
210   continue

C  - Vitesses:
      write (7,1400)
      write (7,1050)
      write (7,1060)
      write (7,1100)
      do 220 IS =  1, NPS
         write (7,1200) IS,(ZX(J,2,IS),J=1,6)
220   continue
      write (7,1050)
      write (7,1060)
      write (7,1300)
      do 230 IS =  1, NPS
         write (7,1200) IS,(Z1(J,2,IS),J=1,6)
230   continue

C  - Accelerations:
      write (7,1500)
      write (7,1050)
      write (7,1070)
      write (7,1100)
      do 240 IS =  1, NPS
         write (7,1200) IS,(ZX(J,3,IS),J=1,6)
240   continue
      write (7,1050)
      write (7,1070)
      write (7,1300)
      do 250 IS =  1, NPS
         write (7,1200) IS,(Z1(J,3,IS),J=1,6)
250   continue
C
 500  format(/6X,'- Debut de la planification de la trajectoire ',
     1       '.........')
1000  format(//' LA TRAJECTOIRE:'/' ',14('-')//6X,'Temps total =  ',
     1       F10.3//' LES POSITIONS:'/' ',13('-'))
1050  format(/)
1060  format(11X,'     .      .      .      .      .      .')
1070  format(11X,'     ..     ..     ..     ..     ..     ..')
1100  format(11X,'     X      Y      Z      TX     TY     TZ')
1200  format(6X,I3,2X,6(F7.2))
1300  format(11X,'     θ1     θ2     θ3     θ4     θ5     θ6')
1400  format(//' LES VITESSES:'/' ',12('-'))
1500  format(//' LES ACCELERATIONS:'/' ',17('-'))
      return
      end
```

```
C***************************************
      subroutine CHAIN2 (Z2,Z1,INC)
C***************************************
C  Cette routine calcule la cinetique angulaire de la chaine 2 connaissant
C  le mouvement angulaire de la chaine 1.

      real*4 Z1(6,3,200),Z2(3,3,200)

      PI =  ACOS(-1.)
      T2   =  Z1(2,1,INC)
      TD2  =  Z1(2,2,INC)
      TDD2 =  Z1(2,3,INC)
      T3   =  Z1(3,1,INC)
      TD3  =  Z1(3,2,INC)
      TDD3 =  Z1(3,3,INC)
      Z2(1,1,INC) =  T2   + T3 - PI
      Z2(1,2,INC) =  TD2  + TD3
      Z2(1,3,INC) =  TDD2 + TDD3
      Z2(2,1,INC) =  3.*PI - T3
      Z2(2,2,INC) =  - TD3
      Z2(2,3,INC) =  - TDD3
      Z2(3,1,INC) =  T3
      Z2(3,2,INC) =  TD3
      Z2(3,3,INC) =  TDD3

      return
      end


C*****************************
      subroutine MOVELA
C*****************************
C  Cette routine gere la planification de la trajectoire pour le nouvel
C  algorithme.

      logical FIN,LPOINT(7),LCHANG,ERREUR
      real*4  TL(6,7),TDL(6,7),X(6),V1(3),V2(3),V3(3)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),VAL(20,2),NS,
     1              NTP
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP1/EW(6,8),DELTA(8),RK
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)
      common/CSTEP4/LENT(7),TLENT(7)
      common/CSTEP6/AL(6,7),LPOINT,DLL(4,7)

C  Initialisation.
      do 10 I =  1, 6
         do 20 J =  1, 8
            do 30 K =  1, 6
               A(I,J,K) =  0.
               C(I,J,K) =  0.
               T(I,J,K) =  0.
```

```
30          continue
20        continue
10     continue

C  A -  STEP 1:  Calcul de la trajectoire cartesienne ideale (pseudo-point).
       call STEP1

C     Initialisation des vitesses et accelerations maximales.
       do 40 I =  1, 6
          do 50 J =  1, M
             do 60 K =  1, 6
                VMAX(I,J,K) =  TDM (I)
                AMAX(I,J,K) =  ADM (I)
60           continue
50        continue
40     continue

C     Cherchons le lien le plus lent:

       ITERA =  0
       FIN =  .FALSE.
70     if (.not.FIN) then
          ITERA =  ITERA + 1
          write (6,1000) ITERA
          FIN  = .TRUE.

          do 80 J =  1, M
C           B -  STEP 2:  Vitesse aux points de visite.
            call STEP2 (J)
            call STEP3 (J)

C     Le step 4 et 5:

            ERREUR =  .true.
90          if (ERREUR) then
               LENT(J)  =  0
               TLENT(J) =  0.
               do 100 I =  1, 6
                  C(I,J+1,1)   =  RK * EW(I,J+1)
100            continue
               do 110 I =  1, 6
                  call STEP4 (I,J)
                  T1 =  0.
                  do 120 K =  1, 3
                     T1 =  T1 + T(I,J,K)
120               continue
                  if (T1.gt.TLENT(J)) then
                     TLENT(J) =  T1
                     LENT(J)  =  I
                  endif
110            continue
               TLENT(J) =  2.*TLENT(J)
```

```
C     Le step 5:
                ERREUR =  .false.
                do 130 I =  1, 6
                   if (.not. ERREUR) then
                      call STEP5 (I,J,1,.FALSE.,LCHANG,ERREUR)
                   endif
130             continue
                if (ERREUR) then
                   RK =  0.
                endif
                goto 90
             endif
             call STEP7 (J,FIN)
80        continue
          goto 70
       endif

C     Ajoutons les points de linearisation:

C     Trouvons les vitesses aux points de linearisation:
       FIN  = .TRUE.
       do 140 J =  1, M
          if (LPOINT(J)) then
             LPOINT(J) =  .false.
C         Trouvons le point qui sera compare au point de linearisation.
             do 150 I =  1, 6
                K    =  0
                TIME =  0.
160             if (TIME.lt.TLENT(J)/2.) then
                   K =  K + 1
                   TIME =  TIME + 2.*T(I,J,K)
                   goto 160
                endif
                DTIME =  2.*T(I,J,K) - (TIME - TLENT(J)/2.)
C              Calcul de la position du point actuel:
                TIME  = -T(I,J,K) + DTIME
                call COEFFI (I,J,K)
                call SOLVE (TL(I,J),TD1,TDD1,TIME)
C              Calcul de la vitesse au point de linearisation:
                if (K.eq.6) then
                   C4  =  C(I,J+1,1)
                else
                   C4  =  C(I,J,K+1)
                endif
                TDL(I,J) =  (C4-C(I,J,K))*DTIME/(2.*T(I,J,K)) + C(I,J,K)
150          continue

C  Doit-on ajouter le point de linearisation?
             call FORKIN (X,TL(1,J))
             do 155 I =  1, 4, 3
                call PRODUI (V2,XYT(I,J+1),XYT(I,J),-1.)
                call SCALAR (V1,-1.,XYT(I,J),3,1)
                call MATADD (V3,X(I),V1,3,1)
                call CROSS  (V1,V2,V3)
```

```
                call NORME  (V1,3,DL1)
                if (DL1.gt.DLL(I,J)) then
                   LPOINT(J) =  .true.
                   FIN =  .FALSE.
                endif
155          continue
          endif
140    continue

       ITERA =  0
170    if (.not.FIN) then
          ITERA =  ITERA + 1
          write (6,1000) ITERA
          FIN  = .TRUE.
          do 180 J =  1, M
             if (LPOINT(J)) then
                call STEP4 (LENT(J),J)
                T1 =  0.
                do 190 K =  1, 3
                   T1 =  T1 + T(LENT(J),J,K)
190             continue
                TLENT(J) =  2.* T1
                call STEP6 (J,TL(1,J),TDL(1,J))
                call STEP7 (J,FIN)
             endif
180       continue
          goto 170
       endif

1000   format(11X,'iteration: ',I3)

       return
       end

C*****************************
       subroutine STEP1
C*****************************
C  Cette routine trouve la position du point de linearisation (step 1.1) et les
C  vecteur vitesse angulaire unitaire aux points de visite (step 1.2).

       logical ERREUR,LPOINT(7)
       real*4 V(6,2),JM(6,6),JMI(6,6),V1(3),V2(3),PLG(6)
       common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
      1              VAL(20,2),NS,NTP
       common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
       common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
       common/CSTEP1/EW(6,8),DELTA(8),RK
       common/CSTEP6/AL(6,7),LPOINT,DLL(4,7)
       common/CBETA /BETA

C  Initialisation
       M =  NS - 2
       if (M.le.0) then
          write (6,*) 'Le programme en VAL ne contient aucun segment'
```

```
      stop
    endif
    if (M.gt.7) then
      write (6,*) 'Le programme contient trop de segments (M = 7)'
      stop
    endif
    BETA =  ATAN2 (E,.5)
    MP1 =  M + 1
    do 5 J =  1, MP1
      IP =  INT (VAL(J,2))
      call EGAL (A(1,J,1),TT(1,IP,1),6,1)
5   continue
    do 10 J =  1, M
      LPOINT(J) =  .FALSE.
C     Step 1.1:  la trajectoire cartesienne ideale.
C     Orientation des vecteurs vitesses:  "Vj".
      if (J.eq.1) then
        call ZERO (V(1,1),6,1)
        DELTA(1) =  0.
      else
        call EGAL (V(1,1),V(1,2),6,1)
      endif
      if (J.eq.M) then
        call ZERO (V(1,2),6,1)
        DELTA(M+1) =  0.
      else
        call VECT (V(1,2),DELTA(J+1),XYT,J)
      endif
C     Point de linearisation:
      IM =  INT(VAL(J,1))
      if (IM.eq.2) then
        LPOINT(J) =  .TRUE.
        call LINEAR (PLG,XYT,J,V,E,DLL(4,J))
        call INVKIN (PLG,AL(1,J),V1)
      endif
C     Step 1.2:  Vecteur vitesse angulaire unitaire:
      if (J.eq.M) then
        call ZERO (EW(1,J+1),6,1)
      endif
      if (J.eq.1) then
        call ZERO (EW(1,J),6,1)
      else
        call JMMATR (JM,JMI,A(1,J,1),1)
        call MATINV (JMI,JM,6,ERREUR)
        if (ERREUR) then
          call ZERO (EW(1,J),6,1)
        else
          call MATMUL (EW(1,J),JMI,V(1,1),6,6,1)
          call UNITE  (EW(1,J),6)
        endif
      endif
10  continue
    return
    end
```

```
C*********************************
      subroutine STEP2 (J)
C*********************************
C  Cette routine trouve la vitesse aux points de visite 'j+1'.

      real*4 X(6,2)

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP1/EW(6,8),DELTA(8),RK
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)

C  Calcul de 'K' au point 'j+1':
      if (J.eq.1) then
         call VECTX (X(1,1),1)
      else
         call EGAL (X(1,1),X(1,2),6,1)
      endif
      if (J.eq.M) then
         RK =  0.
      else
         call VECTX (X(1,2),J+1)
         RK =  0.
         do 10 I =  1, 6
            RK =  RK + EW(I,J+1)*(X(I,2)+X(I,1))
10       continue
         RK =  (RK / 2.) * DELTA(J+1)
      endif

C    Reduction de 'K':
      do 20 I =  1, 6
         C1 =  ABS (RK*EW(I,J+1))
         if (C1.gt.VMAX(I,J+1,2)) RK =  RK*VMAX(I,J+1,2)/C1
20    continue

      return
      end


C*********************************
      subroutine STEP3 (J)
C*********************************
C  Cette routine fait le decouplement des segments de trajectoire.

      real*4 RK2(6)

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP1/EW(6,8),DELTA(8),RK
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)
      common/CSTEP3/NOMBRE

      if (J.ne.M) then
         IK =  0
         do 10 I =  1, 6
            RK2(I) =  RK
```

```
            DIST    =  A(I,J+2,1) - A(I,J+1,1)
            call CALCUL (DIST,AMAX(I,J+1,1),RK2(I),EW(I,J+1),IK)
10        continue
          if (IK.ge.NOMBRE) RK =  AMIN1 (RK2(1),RK2(2),RK2(3))
        endif

        return
        end


C*********************************
        subroutine STEP4 (I,J)
C*********************************
C  Cette routine calcule le lien le plus lent.

        common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
        common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)

        EPS =  0.0001

        A1  =  A(I,J,1)
        A4  =  A(I,J+1,1)
        C1  =  C(I,J,1)
        C4  =  C(I,J+1,1)
        AM1 =  AMAX(I,J,1)
        AM3 =  AMAX(I,J,3)
C     Le cas 'a':  une transition.
        AD   =  AMIN1 (AM1,AM3)
        T1   =  .75 * ABS(C4-C1) / AD
        B    =  A1 +  (C1+C4)*T1
        DIST =  A4 - B
        if (ABS(DIST).le.EPS) then
C         Le cas 'a' est la solution.
          T(I,J,1) =  T1
          T(I,J,2) =  0.
          T(I,J,3) =  0.
          C(I,J,2) =  C4
        else
          AD1   =  AM1
          T1    =  .75 * ABS(C4-C1) / AD1
          B     =  A1 +  (C1+C4)*T1
          DIST =  A4 - B
          call SIGNE (DIST,S1)
          AD3   =  AM3
          T3    =  .75 * ABS(C4-C1) / AD3
          B     =  A1 +  (C1+C4)*T3
          DIST =  A4 - B
          call SIGNE (DIST,S2)
          if (S1.ne.S2) then
C           Le cas 'b' est la solution.
            DIST =  C4 - C1
            call SIGNE (DIST,S)
            AD1 =  S * AD1
            AD3 =  S * AD3
```

```
            C2  =  4.*AD1*AD3*(A4-A1)/3. + AD3*C1**2 - AD1*C4**2
            C2P =  SQRT(C2/(AD3-AD1))
            T1P =  .75*(C2P-C1)/AD1
            A2P =  A1 + (C2P+C1)*T1P
            T3P =  .75*(C4-C2P)/AD3
            A4P =  A2P + (C2P+C4)*T3P
            EP  =  abs(A4-A4P)
            C2M = -C2P
            T1M =  .75*(C2M-C1)/AD1
            A2M =  A1 + (C2M+C1)*T1M
            T3M =  .75*(C4-C2M)/AD3
            A4M =  A2M + (C2M+C4)*T3M
            EM  =  abs(A4-A4M)
            if (T1P.ge.0. .and. T3P.ge.0) then
               if (T1M.ge.0. .and. T3M.ge.0) then
                  if (EP.le.EM) then
                     C(I,J,2) =  C2P
                     T(I,J,1) =  T1P
                     T(I,J,3) =  T3P
                  else
                     C(I,J,2) =  C2M
                     T(I,J,1) =  T1M
                     T(I,J,3) =  T3M
                  endif
               else
                  C(I,J,2) =  C2P
                  T(I,J,1) =  T1P
                  T(I,J,3) =  T3P
               endif
            else
               if (T1M.ge.0. .and. T3M.ge.0) then
                  C(I,J,2) =  C2M
                  T(I,J,1) =  T1M
                  T(I,J,3) =  T3M
               else
                  write (6,*) 'erreur 1:  voir STEP 4'
                  stop
               endif
            endif
            T(I,J,2) =   0.
         else
            S =  S1
            AD1 =  S * AD1
            AD3 = -S * AMAX(I,J,3)
            VM  =  S * VMAX(I,J,2)
            B   =  A1 + .75*(VM**2-C1**2)/AD1
            SA  =  A4 - .75*(C4**2-VM**2)/AD3
            if (S*SA .le. S*B) then
C             Le cas 'c':  deux transitions.
              C2  =  4.*AD1*AD3*(A4-A1)/3. +AD3*C1**2 -AD1*C4**2
              C2P =  SQRT(C2/(AD3-AD1))
              T1P =  .75*(C2P-C1)/AD1
              A2P =  A1 + (C2P+C1)*T1P
              T3P =  .75*(C4-C2P)/AD3
```

```
                A4P =  A2P + (C2P+C4)*T3P
                EP  =  abs(A4-A4P)
                C2M = -C2P
                T1M =  .75*(C2M-C1)/AD1
                A2M =  A1 + (C2M+C1)*T1M
                T3M =  .75*(C4-C2M)/AD3
                A4M =  A2M + (C2M+C4)*T3M
                EM  =  abs(A4-A4M)
                if (T1P.ge.0. .and. T3P.ge.0) then
                   if (T1M.ge.0. .and. T3M.ge.0) then
                      if (EP.le.EM) then
                         C(I,J,2) =  C2P
                         T(I,J,1) =  T1P
                         T(I,J,3) =  T3P
                      else
                         C(I,J,2) =  C2M
                         T(I,J,1) =  T1M
                         T(I,J,3) =  T3M
                      endif
                   else
                      C(I,J,2) =  C2P
                      T(I,J,1) =  T1P
                      T(I,J,3) =  T3P
                   endif
                else
                   if (T1M.ge.0. .and. T3M.ge.0) then
                      C(I,J,2) =  C2M
                      T(I,J,1) =  T1M
                      T(I,J,3) =  T3M
                   else
                      write (6,*) 'erreur 1:  voir STEP 4'
                      stop
                   endif
                endif
                T(I,J,2) =   0.
             else
C              Le cas 'd':  transition - vitesse constante - transition.
               T(I,J,1) =  .75 * (VM-C1)/AD1
               T(I,J,3) =  .75 * (C4-VM)/AD3
               T(I,J,2) =  .50 * (SA - B)/VM
               C(I,J,2) =   VM
             endif
          endif
       endif
       A(I,J,2) =  A1 + (C1+C(I,J,2))*T(I,J,1)
       A(I,J,3) =  A(I,J,2) + 2.*C(I,J,2)*T(I,J,2)
       C(I,J,3) =  C(I,J,2)
       do 10 K =  4, 6
          A(I,J,K) =  A4
          C(I,J,K) =  C4
          T(I,J,K) =  0.
10     continue
       return
       end
```

```
C*************************************************************
      subroutine STEP5 (I,J,K,LIGNE,LCHANG,ERREUR)
C*************************************************************
C  Cette routine ajuste les autres liens avec le lien le plus lent.

      logical LIGNE,LCHANG,FIN,ERREUR

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)
      common/CSTEP4/LENT(7),TLENT(7)

      EPS    =  0.0001
      LCHANG =  .FALSE.

      if (I.ne.LENT(J)) then
C       Initialisation:
        VM =  VMAX(I,J,K+1)
        A1 =  A(I,J,K)
        C1 =  C(I,J,K)
        if (.not.LIGNE) then
C         Initialisation pour l'etape '5'.
          A4 =  A(I,J+1,1)
          C4 =  C(I,J+1,1)
          TT =  TLENT(J)
        else
C         Initialisation pour l'etape '6'.
          if (K.eq.1) then
C           - segment 'j1':
            A4 =  A(I,J,4)
            C4 =  C(I,J,4)
          else
C           - segment 'j2':
            A4 =  A(I,J+1,1)
            C4 =  C(I,J+1,1)
          endif
          TT =  TLENT(J) / 2.
        endif
        AM1 =  AMAX(I,J,K)
        AM3 =  AMAX(I,J,K+2)
        B   =  A1 + (C1+C4)*TT/2.
        AD  =  1.5 * (C4 - C1) / TT
        DIST =  A4 - B
        if (ABS(DIST).le.EPS) then
C         Le cas 'a' est la solution:  une transition
          if (abs(AD) .le. AM1) then
            T(I,J,K)   =  TT/2.
            T(I,J,K+1) =  0.
            T(I,J,K+2) =  0.
          else
            T(I,J,K)   =  0.
            T(I,J,K+1) =  0.
            T(I,J,K+2) =  TT/2.
          endif
          C2 =  C4
```

```
else
   FIN =  .FALSE.
   call SIGNE (DIST,S)
   call SIGNE (C4-C1,SC)
   SM  =  S*SC
   if (SM .gt. 0.) then
      if (AM1 .le. abs(AD)) then
         if (LIGNE) then
            FIN    =  .TRUE.
            AD1    =  1.
            AD3    =  AD
            C2     =  C1
            LCHANG =  .TRUE.
         else
            ERREUR =  .true.
            goto 10
         endif
      endif
   else
      if (SM .lt. 0.) then
         if (AM3 .le. abs(AD)) then
            if (LIGNE) then
               FIN  =  .TRUE.
               AD1  =  AD
               AD3  =  1.
               C2   =  C4
               LCHANG =  .TRUE.
            else
               ERREUR =  .true.
               goto 10
            endif
         endif
      endif
   endif

   if (FIN) then
      T(I,J,K)   =  .75*(C2-C1) / AD1
      T(I,J,K+1) =   0.
      T(I,J,K+2) =  .75*(C4-C2) / AD3
   else
      BB  =  -2.*(A4-A1)/TT
      CC  =  (2.*(A4-A1)*(C1+C4)/TT - C1**2 - C4**2) / 2.
      C21 =  (-BB + SQRT(BB**2-4.*CC)) / 2.
      C22 =  (-BB - SQRT(BB**2-4.*CC)) / 2.
      if (S .gt. 0.) then
         C2 =  C21
      else
         C2 =  C22
      endif
      AD =  1.5*S* (2.*C2 - C1 - C4) / TT
      AD1 =  S * AD
      AD3 = -AD1
```

```
      if (abs(AD1).gt.AM1 .or. abs(AD3).gt.AM3) then
         if (abs(AD1).gt.AM1 .and. abs(AD3).lt.AM3) then
            AD1  =  S * AM1
            C2S  =  2.*TT*AD1/3. + C1
            B    =  A1 + (C1+C2S)*TT/2.
            call SIGNE (A4-B,S2)
            if (S.eq.S2 .or. S2.eq.0.) then
               if (LIGNE) then
                  LCHANG =  .TRUE.
               else
                  ERREUR =  .TRUE.
                  goto 10
               endif
            else
               TEMP =  (C1-C4)/AD1
               C2   =  4.*(A4-A1)/3. - 2.*TT*C4/3. + C1*TEMP
               C2   =  C2 / (2.*TT/3. + TEMP)
               AD3  =  (C4-C2)/(2.*TT/3. - (C2-C1)/AD1)
            endif
         else
            if (abs(AD1).lt.AM1 .and. abs(AD3).gt.AM3) then
               AD3  = -S * AM3
               C1S  =  C4 - 2.*TT*AD3/3.
               B    =  A1 + (C1S+C4)*TT/2.
               call SIGNE (A4-B,S2)
               if (S.eq.S2 .or. S2.eq.0.) then
                  if (LIGNE) then
                     LCHANG =  .TRUE.
                  else
                     ERREUR =  .TRUE.
                     goto 10
                  endif
               else
                  TEMP =  (C1-C4)/AD3
                  C2   =  4.*(A4-A1)/3.-2.*TT*C1/3.+C4*TEMP
                  C2   =  C2 / (2.*TT/3. + TEMP)
                  AD1  =  (C2-C1)/(2.*TT/3. - (C4-C2)/AD3)
               endif
            else
               if (LIGNE) then
                  LCHANG =  .TRUE.
               else
                  ERREUR =  .TRUE.
                  goto 10
               endif
            endif
         endif

         if (abs(AD1).gt.AM1 .or. abs(AD3).gt.AM3 .or. LCHANG)
     1                                                        then
            if (LIGNE) then
C             Le point de linearisation doit etre modifie pour:
              FIN  =  .TRUE.
              AD1  =  S * AM1
```

```
                AD3  = -S * AM3
                C2   =  2.*AD1*AD3*TT/3. + AD3*C1 - AD1*C4
                C2   =  C2 / (AD3-AD1)
                LCHANG =  .TRUE.
             else
                ERREUR =  .TRUE.
                goto 10
             endif
          endif
       endif

       if (ABS(C2) .le. VM) then
C        - Le cas 'c' est la solution:  deux transitions.
          T(I,J,K)   =  .75*(C2-C1) / AD1
          T(I,J,K+1) =   0.
          T(I,J,K+2) =  .75*(C4-C2) / AD3
       else
C        Le cas 'd' est la solution:
          C2  =  S*VM
          AD1 =  S*AM1
          AD3 = -S*AM3
          T1 =  .75*(C2-C1)/AD1
          T3 =  .75*(C4-C2)/AD3
          T2 =   TT/2. - T1 - T3
          A2 =  A1 + (C1+C2)*T1
          A3 =  A2 + 2.*C2*T2
          B  =  A3 + (C2+C4)*T3
          if (S*A1.le.S*B .and. S*B.le.S*A4) then
             if (LIGNE) then
                LCHANG =  .TRUE.
                T(I,J,K)   =  T1
                T(I,J,K+1) =  T2
                T(I,J,K+2) =  T3
             else
                ERREUR =  .TRUE.
                goto 10
             endif
          else
             AD  =  3.*((C2-C1)**2 + (C4-C2)**2) / (4.*S)
             AD  =  AD / (TT*C2 - (A4-A1))
             AD1 =  S * AD
             AD3 = -S * AD
             if (abs(AD1) .gt. AM1) then
                AD1 =  S * AM1
                AD3 = -3.*(C4-C2)**2 / 4.
                AD3 = AD3/(TT*C2-(A4-A1)-3.*(C2-C1)**2/(4.*AD1))
             endif
             if (abs(AD3) .gt. AM3) then
                AD3 = -S * AM3
                AD1 =  3.*(C2-C1)**2 / 4.
                AD1 = AD1/(TT*C2-(A4-A1)+3.*(C4-C2)**2/(4.*AD3))
             endif
             T(I,J,K)   =  .75*(C2-C1) / AD1
             T(I,J,K+2) =  .75*(C4-C2) / AD3
```

```
                  T(I,J,K+1) =   TT/2. - T(I,J,K) - T(I,J,K+2)
               endif
            endif
         endif
      endif

      if (K.eq.1) then
         A(I,J,K+1) =  A1 + (C1+C2)*T(I,J,K)
         A(I,J,K+2) =  A(I,J,K+1) + 2.*C2*T(I,J,K+1)
         if (LCHANG) then
            A(I,J,K+3) =  A(I,J,K+2) + (C2+C4)*T(I,J,K+2)
         endif
      else
         A(I,J,K+2) =  A4 - (C2+C4)*T(I,J,K+2)
         A(I,J,K+1) =  A(I,J,K+2) - 2.*C2*T(I,J,K+1)
         if (LCHANG) then
            A(I,J,K) =  A(I,J,K+1) - (C1+C2)*T(I,J,K)
         endif
      endif
      C(I,J,K+1) =  C2
      C(I,J,K+2) =  C2
   endif

10 continue

   return
   end


C**************************************
      subroutine STEP6 (J,TL,TDL)
C**************************************
C  Cette routine ajoute le point de linearisation.

      logical LPOINT(7),LCHANG,ERREUR
      real*4  TL(6),TDL(6)

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP4/LENT(7),TLENT(7)
      common/CSTEP6/AL(6,7),LPOINT,DLL(4,7)

      do 10 I =  1, 6
         if (LPOINT(J)) then
            if (I.ne.LENT(J)) then
               A(I,J,4) =  AL (I,J)
               C(I,J,4) =  TDL(I)
            endif
            call STEP5 (I,J,1,.TRUE.,LCHANG,.false.)
            call STEP5 (I,J,4,.TRUE.,LCHANG,.false.)
            if (LCHANG) call STEP5 (I,J,1,.TRUE.,LCHANG,.false.)
         else
            if (I.ne.LENT(J)) then
               call STEP5 (I,J,1,.FALSE.,LCHANG,ERREUR)
               if (ERREUR) then
```

```
                write(6,*) 'Erreur routine 6.'
                stop
              endif
            endif
          endif
10    continue

      return
      end


C************************************
      subroutine STEP7 (J,FIN)
C************************************
C Cette routine gere la dynamique (dynamic scaling  Hollerbach 1984).

      integer KT(6),KFLAG(6)
      logical FIN
      real*4 T1(8),TD1(7),TDD1(7),TF1(7),TA1(7),TG1(7),TS(150)
      real*4 T2(8),TD2(7),TDD2(7),TF2(7),TA2(7),TG2(7)
      real*4 TF(6),TA(6),TG(6),TN(6,2),C2(6),TEMPS(6)

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)
      common/CSTEP4/LENT(7),TLENT(7)

      PI  =  ACOS(-1.)
      EPS =  0.001

C  Discretisation de l'interval de temps 'Tj':
      call ZERO (TS,35,1)
      NTS =  0
      do 10 I =  1, 6
         TIME     =  0.
         do 20 K =  1, 6
            T11  =  T(I,J,K)
            if (T11.gt.0.) then
               NTS     =  NTS + 1
               TS(NTS) =  TIME + T11
               TIME =  TIME + 2.*T11
            endif
20       continue
10    continue
      NTS     =  NTS + 1
      TS(NTS) =  TIME

C  Ordonnons (par ordre croissant) les valeurs de 'TS':
      NT1  =  0
      NTM1 =  NTS - 1
30    if (NT1 .lt. NTM1) then
         NT1  =  NT1 + 1
         NT2 =  NT1
40       if (NT2 .lt. NTS) then
```

```
            NT2 =  NT2 + 1
 60         if (NT2.le.NTS) then
               if (TS(NT2).lt.TS(NT1)) then
                  TEMP    =  TS(NT1)
                  TS(NT1) =  TS(NT2)
                  TS(NT2) =  TEMP
               else
                  if (TS(NT2).eq.TS(NT1)) then
                     NT =  NT2
 50                  if (NT.lt.NTS) then
                        NT =  NT + 1
                        TS(NT-1) =  TS(NT)
                        goto 50
                     endif
                     NTS  =  NTS - 1
                     NTM1 =  NTM1 - 1
                     goto 60
                  endif
               endif
            endif
            goto 40
         endif
         goto 30
      endif
      DELTAT =  TLENT(J) * ERROR
      IT =  1
 61   if (IT.le.NTS) then
         if (IT .eq. 1) then
            DT =  TS(1)
         else
            DT =  TS(IT) - TS(IT-1)
         endif
         NDT =  int (DT/DELTAT)
         if (NDT.ge. 1) then
 C         On doit ajouter d'autres points.
           do 62 I =  IT, NTS
              II =  NTS + IT - I
              TS(II+NDT) =  TS(II)
 62        continue
           DT =  DT / (NDT+1)
           do 63 I =  1, NDT
              II     =  IT - 1 + I
              if (IT .eq. 1) then
                 TS(II) =  I*DT
              else
                 TS(II) =  TS(IT-1) + I*DT
              endif
 63        continue
           IT  =  IT  + NDT
           NTS =  NTS + NDT
         endif
         IT =  IT + 1
         goto 61
      endif
```

```
      NTS =  NTS - 1

C  Initialisation
      do 65 I =  1, 6
         KT(I)    =  0
         KFLAG(I) =  1
         TEMPS(I) =  0.
         C2(I)    =  3.4E38
65    continue

      do 70 IT =  1, NTS
C       Les positions, vitesse et acceleration angulaires au temps TS(IT):
        do 80 I =  1, 6
90         if (TEMPS(I).lt.TS(IT) .and. KT(I).lt.6) then
              KT(I)    =  KT(I) + 1
              TEMPS(I) =  TEMPS(I) + 2.*T(I,J,KT(I))
              goto 90
           endif
           K     =  KT(I)
           DTIME =  2.*T(I,J,K) - (TEMPS(I) - TS(IT))
           TIME  = -T(I,J,K) + DTIME
           call COEFFI (I,J,K)
           call SOLVE (T1(I+1),TD1(I+1),TDD1(I+1),TIME)
80      continue

C       Les moments de torsion actuels:
        call NEWTON (TF1,TA1,TG1,6,T1,TD1,TDD1,1)
        T2(2)   =  T1(2)
        T2(3)   =  T1(3) + T1(4) - PI
        T2(4)   =  3.*PI - T1(4)
        TD2(2)  =  TD1(2)
        TD2(3)  =  TD1(3) + TD1(4)
        TD2(4)  = -TD1(4)
        TDD2(2) =  TDD1(2)
        TDD2(3) =  TDD1(3) + TDD1(4)
        TDD2(4) = -TDD1(4)
        call NEWTON (TF2,TA2,TG2,3,T2,TD2,TDD2,2)
        call INIT2 (T1,T2)
        call CHAINE(TA,TA1,TA2)
        call CHAINE(TG,TG1,TG2)

        do 100 I =  1, 6
C         Les limites sur les moments de torsion:  Nmax =  Nmax - G(t).
          TN(I,1) = -TORMAX(I) - TG(I)
          TN(I,2) =  TORMAX(I) - TG(I)
C         la valeur maximale de C2:
          if (TN(I,2).gt.0. .and. TN(I,1).lt.0.) then
C           Cas 1:
            if (TA(I).gt.0.) then
               C22 =  TN(I,2)/TA(I)
            else
               if (TA(I).lt.0.) then
                  C22 =  TN(I,1)/TA(I)
               else
```

```
                C22 =  3.4E38
              endif
            endif
          else
            if (TN(I,2).gt.0. .and. TN(I,1).gt.0.) then
C             Cas 2:
              if (TA(I).gt.0.) then
                C22 =  TN(I,2)/TA(I)
              else
                write (6,*) 'erreur 1:  voir STEP7'
                stop
              endif
            else
              if (TN(I,2).lt.0. .and. TN(I,1).lt.0.) then
C               Cas 3:
                if (TA(I).ge.0.) then
                  write (6,*) 'erreur 2:  voir STEP7'
                  stop
                else
                  C22 =  TN(I,1)/TA(I)
                endif
              else
                write (6,*) 'erreur 3:  voir STEP7'
                stop
              endif
            endif
          endif
          if (KT(I).eq.KFLAG(I) .and. IT.ne.NTS) then
            C2(I) =  AMIN1(C2(I),C22)
          else
            if (KT(I).eq.KFLAG(I))  C2(I) =  AMIN1(C2(I),C22)
C           Modifions les vitesses et les accelerations maximales:
            K =  KFLAG(I)
            if (K.eq.2 .or. K.eq.5) then
              DIST =  A(I,J,K+1) - A(I,J,K)
              if (abs(DIST).gt.EPS) then
                TEMP        =  VMAX(I,J,K)
                if (C2(I).gt.3.4E38) C2(I) =  3.4E38
                TEMP2       =  SQRT(C2(I))
                VMAX(I,J,K) =  ABS((DIST)*TEMP2/(2.*T(I,J,K)))
                ERREUR      =  ABS(TEMP - VMAX(I,J,K)) / TEMP
                if (ERREUR.gt.ERROR) FIN =  .FALSE.
              endif
            else
              if (K.eq.6) then
                D =  C(I,J+1,1)
              else
                D =  C(I,J,K+1)
              endif
              DIST  =  D - C(I,J,K)
              if (abs(DIST).gt.EPS) then
                TEMP        =  AMAX(I,J,K)
                if (C2(I).gt.3.4E38) C2(I) =  3.4E38
                TEMP2       =  SQRT(C2(I))
```

```
                  AMAX(I,J,K) =  ABS(.75*(DIST)*TEMP2 / T(I,J,K))
                  AJERK       =  SQRT(3.*abs(DIST*XJERK(I))/8.)
                  AMAX(I,J,K) =  AMIN1(AMAX(I,J,K),AJERK)
                  AMAX(I,J,K) =  AMAX1(AMAX(I,J,K),.1)
                  ERREUR      =  ABS(TEMP - AMAX(I,J,K)) / TEMP
                  if (ERREUR.gt.ERROR) FIN =  .FALSE.
               endif
            endif
            if (IT .eq. NTS) then
110            if (K .lt. 6) then
                  K =  K + 1
                  if (K .eq. 3) then
                     AMAX(I,J,3) =  AMAX(I,J,1)
                  endif
                  if (K .eq. 4) then
                     AMAX(I,J,4) =  AMAX(I,J,3)
                  endif
                  if (K .eq. 5) then
                     VMAX(I,J,5) =  VMAX(I,J,2)
                  endif
                  if (K .eq. 6) then
                     AMAX(I,J,6) =  AMAX(I,J,4)
                  endif
                  goto 110
               endif
            else
               KFLAG(I) =  KT(I)
               C2(I)    =  C22
            endif
         endif
100   continue
70    continue

      return
      end

C*****************************************************
      subroutine VECT (V,DELTA,XYT,J)
C*****************************************************
C Cette routine calcule le vecteur orientation "V" a partir des positions.

      real*4 V(6),XYT(6,8),V1(3),V2(3)

      common/CBETA/BETA

      DELTA =  1.

      do 10 I =  1,4,3
         call PRODUI (V1,XYT(I,J),  XYT(I,J-1),-1.)
         call PRODUI (V2,XYT(I,J+1),XYT(I,J),  -1.)
         call PRODUI (V(I),V1,V2,1.)
         TEMP =  0.
         do 20 II =  1, 3
            TEMP =  TEMP +  V2(II)*V(I-1+II)
```

```
20       continue

C       Calcul de DELTA:
         THETA =  ACOS (TEMP) / 2.
         if (BETA.lt.THETA) then
            DELTA1 =  BETA / THETA
            DELTA  =  AMIN1(DELTA, DELTA1)
         endif
10    continue

      return
      end


C**************************************
      subroutine EGAL (A,B,M,N)
C**************************************
C Cette routine fait l'egalite suivante:  A =  B

      real*4 A(M,N), B(M,N)

      do 10 I =  1, M
         do 20 J =  1, N
            A(I,J) =  B(I,J)
20       continue
10    continue

      return
      end


C*******************************************
      subroutine PRODUI (V,V1,V2,S)
C*******************************************
C Cette routine calcule l'equation suivante:  V =  (V1 + sign*V2)
C                                                  --------------
C                                                  |V1 + sign*V2|

      real*4 V(3),V1(3),V2(3),V3(3)

      call SCALAR (V3,S,V2,3,1)
      call MATADD (V,V1,V3,3,1)
      call UNITE  (V,3)

      return
      end


C***********************************
      subroutine UNITE (V,N)
C***********************************
C Cette routine convertit 'V' en vecteur unitaire.

      real*4 V(N)
```

```
      EPS    =  0.00001
      call NORME (V,N,XNORME)
      if (XNORME.gt.EPS) then
         do 20 I =  1, N
            V(I) =  V(I)/XNORME
20       continue
      else
         call ZERO (V,N,1)
      endif

      return
      end


C*****************************************
      subroutine NORME (V,N,XNORME)
C*****************************************
C Cette routine trouve la norme d'un vecteur 'V'.

      real*4 V(N)

      XNORME =  0.
      do 10 I =  1, N
         XNORME =  XNORME + V(I)**2
10    continue
      XNORME =  SQRT (XNORME)

      return
      end


C*************************************************
      subroutine LINEAR (PLG,XYT,J,V,E,DL)
C*************************************************
C Cette routine calcule la position du point de linearisation et sa distance
C par rapport a la ligne droite reliant les deux points de visite.

      real*4 PLG(6),XYT(6,8),V(6,2),V1(3),V2(3),P1(3),P2(3),DL(4)

C  Calcul du point de linearisation:
      do 10 I =  1, 4, 3
         call SCALAR (V2,-1.,XYT(I,J),3,1)
         call MATADD (V1,XYT(I,J+1),V2,3,1)
         DLE  =  SQRT (V1(1)**2 + V1(2)**2 + V1(3)**2) * E
         call SCALAR (V1,DLE,V(I,1),3,1)
         call MATADD (P1,XYT(I,J),V1,3,1)
         DLE  = -DLE
         call SCALAR (V1,DLE,V(I,2),3,1)
         call MATADD (P2,XYT(I,J+1),V1,3,1)
         call MATADD (V1,P1,P2,3,1)
         call SCALAR (PLG(I),.5,V1,3,1)

C     Calcul de la distance minimum:
         call PRODUI (V2,XYT(I,J+1),XYT(I,J),-1.)
```

```
         call SCALAR (P1,-1.,XYT(I,J),3,1)
         call MATADD (P2,PLG(I),P1,3,1)
         call CROSS  (V1,V2,P2)
         call NORME  (V1,3,DL(I))
10    continue

      return
      end


C**********************************
      subroutine ZERO(A,M,N)
C**********************************
C  Cette routine initialise une matrice a zero.

      real*4 A(M,N)

      do 10 I =  1, M
         do 20 J =  1, N
            A(I,J) =  0.
20       continue
10    continue

      return
      end


C**********************************
      subroutine VECTX (X,J)
C**********************************
C  Cette routine trouve la vitesse 'X' du segment 'J'.

      real*4 X(6)

      common/DONNEE/A(6,8,6),C(6,8,6),T(6,8,6),M
      common/CSTEP2/VMAX(6,7,6),AMAX(6,7,6)

      TMAX =  0.
      do 10 I =  1, 6
         T1 =  ABS ((A(I,J+1,1)-A(I,J,1)) / VMAX(I,J,2))
         TMAX =  AMAX1(T1,TMAX)
10    continue
      do 20 I =  1, 6
         X(I) =  (A(I,J+1,1)-A(I,J,1)) / TMAX
20    continue

      return
      end
```

```
C*************************************************
      subroutine CALCUL (DIST,AMAX,RK,EW,IL)
C*************************************************
C  Cette routine fait certains calculs pour le STEP 3.

      call SIGNE (DIST,S)
      C1 =  S * SQRT(4.*AMAX*DIST*S/3.)
      V  =  RK * EW
      if (S*V.gt.S*C1) then
         RK =  RK * ABS(C1 / V)
         IL =  IL + 1
      endif

      return
      end


C**********************************
      subroutine SIGNE (A,S)
C**********************************
C Cette routine calcule le signe 'S' de la valeur 'A'.

      if (A.lt.0.) then
         S =  -1.
      else
         if (A.gt.0.) then
            S =  1.
         else
            S =  0.
         endif
      endif

      return
      end


C************************************
      subroutine COEFFI (I,J,K)
C************************************
C Cette routine calcule les coefficients du polynome.

      common/DONNEE/AA(6,8,6),CC(6,8,6),TT(6,8,6),M
      common/STEP66/A0,A1,A2,A4

      A =  AA(I,J,K)
      C =  CC(I,J,K)
      T =  TT(I,J,K)
      if (K.eq.6) then
         B =  AA(I,J+1,1)
         D =  CC(I,J+1,1)
      else
         B =  AA(I,J,K+1)
         D =  CC(I,J,K+1)
      endif
```

```
      A4  =     (C-D) / (16.*T**3)
      A2  = -3.*(C-D) / (8.*T)
      A1  = -(15.*(A-B) + 7.*T*(C+D)) / (16.*T)
      A0  =  ( 8.*(A+B) + 5.*T*(C-D)) / 16.

      return
      end


C*****************************************
      subroutine SOLVE (T,TD,TDD,TIME)
C*****************************************
C Cette routine calcule les positions, les vitesses et les accelerations
C angulaires.

      common/STEP66/A0,A1,A2,A4

      T   =      A4*TIME**4 +    A2*TIME**2 + A1*TIME + A0
      TD  =   4.*A4*TIME**3 + 2.*A2*TIME    + A1
      TDD =  12.*A4*TIME**2 + 2.*A2

      return
      end

C*****************************
      subroutine CONV1
C*****************************
C    Convertion des angles de degres a radians.

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      PI   =  ACOS(-1.)
      FACT =  PI / 180.

      do 10 I =  1, 6
         do 20 J =  1, 2
            TMAX(I,J) =  TMAX(I,J)*FACT
            call ANGPOS (TMAX(I,J))
            if (I.EQ.1) then
               if (TMAX(I,1).GT.PI) TMAX(I,1) =  TMAX(I,1) - 2.*PI
            endif
20       continue
10    continue

      return
      end


C*****************************
      subroutine CONV2
C*****************************
C    Convertion des angles de degres a radians.
```

```
      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      PI   =  ACOS(-1.)
      FACT =  PI / 180.

      do 10 I =  1, NTP
         do 20 J =  4, 6
            XYT(J,I) =  XYT(J,I)*FACT
            call ANGPOS (XYT(J,I))
20       continue
10    continue

      return
      end


C*****************************
      subroutine CINKI
C*****************************
C  Cette routine gere l'inverse cinematique pour chaque point de visite.

      character*1 CREP
      real*4 TD1(6),TD2(3)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      FACT =  ACOS(-1.)/180.

      call INITGR(1)
      call CLOSEG
      write (6,500)
      write (7,1000)
      do 10 I =  1, NTP
         write (7,2000) I,(XYT(J,I),J=1,6)
10    continue
      write (7,3000)
      do 20 I =  1, NTP
         call INVKIN(XYT(1,I),TT(1,I,1),TT(1,I,2))
         do 30 J =  1, 6
            TD1(J) =  TT(J,I,1)/FACT
30       continue
         write (7,4000) I,(TD1(J),J=1,6)
20    continue
      write (7,5000)
      do 35 I =  1, NTP
         do 40 J =  1, 3
            TD2(J) =  TT(J,I,2)/FACT
40       continue
         write (7,6000) I,(TD2(J),J=1,3)
35    continue
C
 500  format(/'1',78('_')//6X,'- Inverse cinematique pour chaque ',
```

```
     1       'point de visite ......')
1000  format(//' POSITION DES POINTS DE VISITE:'/' ',
     1       29('-')///' A -  Espace cartesien:'//6X,'point',10X,' X',
     2       9X,'Y',9X,'Z',8X,'TX',8X,'TY',8X,'TZ')
2000  format(8X,I1,'    ',2X,6F10.2)
3000  format(/' B -  Espace angulaire:'//4X,'chaine 1:'/6X,'point',5X,
     1       'θ1',8X,'θ2',8X,'θ3',8X,'θ4',8X,'θ5',8X,'θ6')
4000  format(8X,I1,6(F10.2))
5000  format(/4X,'chaine 2:'/6X,'point',4X,'θ1''',7X,'θ2''',7X,'θ3''')
6000  format(8X,I1,3(F10.2))
C
      return
      end


C************************************
      subroutine INVKIN(X,T1,T2)
C************************************
C  Cette sous-routine fait l'inverse cinematique.

      character*1 CREP
      real*4 X(6),T1(6),T2(3)
      logical LREP

      common/KINEMA/XL1,XL2,XL3,XL4,XL5,XL6,A1,A2,A3,R1(3),
     1              TMAX(6,2),RR(186)

      PI    =  ACOS(-1.)
      FACT  =  180. / PI
      EPS   =  .00001

C  Position dans l'espace cartesien:
      PX    =  X(1)
      PY    =  X(2)
      PZ    =  X(3)

C  Orientation (representation Roll-Pitch-Yaw):
      PSI   =  X(4)
      THETA =  X(5)
      PHI   =  X(6)
      SPSI  =  SIN(PSI)
      CPSI  =  COS(PSI)
      SPHI  =  SIN(PHI)
      CPHI  =  COS(PHI)
      STHETA=  SIN(THETA)
      CTHETA=  COS(THETA)
      XN    =  CPHI*CTHETA
      YN    =  SPHI*CTHETA
      ZN    =  -STHETA
      OX    =  CPHI*STHETA*SPSI - SPHI*CPSI
      OY    =  SPHI*STHETA*SPSI + CPHI*CPSI
      OZ    =  CTHETA*SPSI
      AX    =  CPHI*STHETA*CPSI + SPHI*SPSI
      AY    =  SPHI*STHETA*CPSI - CPHI*SPSI
```

```
      AZ    =  CTHETA*CPSI

C  A -  premiere chaine:

C    - Theta 1:
      T1(1) =  ATAN2(PY-AY*XL6,PX-AX*XL6)

C    - Theta 6:
      S1    =  SIN(T1(1))
      C1    =  COS(T1(1))
      C5    =  YN*C1 - XN*S1
      DY    =  OX*S1-OY*C1
      DX    =  AX*S1-AY*C1
      TEMP  =  SQRT(DX**2+DY**2)
      S5    =  SQRT(1.-C5**2)
      if (S5.LT.EPS .and. TEMP.lt.EPS) then
         T1(6) =  PI/2.
      else
         T1(6) =  ATAN2(DY,DX)
         call CHECK (T1(6),TMAX(6,1),TMAX(6,2),LREP)
         if (.NOT.LREP) then
            T1(6) =  T1(6) + PI
         endif
      endif
      call ANGPOS (T1(6))

C    - Theta 5:
      S6     =  SIN(T1(6))
      C6     =  COS(T1(6))
      DX1    =  AX*C6 + OX*S6
      DX2    =  AY*C6 + OY*S6
      DX3    =  AZ*C6 + OZ*S6
      if (ABS(DX3).GT.ABS(DX1) .AND. ABS(DX3).GT.ABS(DX2)) then
         S5     =  C5*ZN/DX3
      else
         if (ABS(DX1).GT.ABS(DX2)) then
            S5 =  (S1+C5*XN) / DX1
         else
            S5 =  (C5*YN-C1) / DX2
         endif
      endif
      T1(5) =  ATAN2(S5,C5)
      call ANGPOS (T1(5))

C    - Theta 3:
      E      =  C1*(-XL5*(AX*S6-OX*C6)-XL6*AX+PX)
      E      =  S1*(-XL5*(AY*S6-OY*C6)-XL6*AY+PY) + E
      F      =  -XL5*(AZ*S6-OZ*C6) - XL6*AZ + PZ - XL1
      C3     = (E**2 + F**2 - XL3**2 - XL2**2) / (2.*XL2*XL3)
      S3     = -SQRT(1. - C3**2)
      T1(3) =  ATAN2(S3,C3)
      call ANGPOS (T1(3))
```

```
C   - Theta 2:
      if (F.lt.0.) then
         SIGN =  -1.
      else
         SIGN =   1.
      endif
      BETA  =  ATAN2(XL3*C3+XL2,XL3*S3)
      R2    =  XL2**2 + XL3**2 + 2.*XL2*XL3*C3
      T1(2) =  BETA - ATAN2(E,SIGN*SQRT(R2-E**2))
      call ANGPOS (T1(2))

C   - Theta 4:
      S5    =  SIN(T1(5))
      C5    =  COS(T1(5))
      XNX   =  C1*(C5*(AX*C6+OX*S6)+XN*S5) + S1*(C5*(AY*C6+OY*S6)+YN*S5)
      YNY   =  C5*(AZ*C6+OZ*S6) + ZN*S5
      T1(4) =  ATAN2(YNY,XNX) - T1(2) - T1(3)
      call ANGPOS (T1(4))


C  B -  seconde chaine:

      T2(1) =  T1(2) + T1(3) - PI
      T2(2) =  3.*PI - T1(3)
      T2(3) =  T1(3)


C  C -  Verifier si les valeurs calculees pour la chaine 1 sont permissibles:

      do 10 I =  1, 6
         call CHECK (T1(I),TMAX(I,1),TMAX(I,2),LREP)
         if (.NOT.LREP) then
            T1D    =  T1(1) * FACT
            T2D    =  T1(2) * FACT
            T3D    =  T1(3) * FACT
            T4D    =  T1(4) * FACT
            T5D    =  T1(5) * FACT
            T6D    =  T1(6) * FACT
            THETAD =  THETA * FACT
            PHID   =  PHI * FACT
            PSID   =  PSI * FACT
            write (6,1000) (X(J),J=1,3),PSID,THETAD,PHID
            write (6,2000) T1D,T2D,T3D,T4D,T5D,T6D
            read (5,3000) CREP
            STOP
         endif
10    continue
C
1000  format(///79('-')/////' Le point suivant n''est pas accessible:',
     1       /1X,38('-')///' A -  Espace cartesien:'//13X,
     2       ' X',9X,'Y',9X,'Z',8X,'TX',8X,'TY',8X,'TZ'/7X,6F10.2)
2000  format(/' B -  Espace angulaire:'//4X,'chaine 1:'/13X,'01',8X,
     1       '02',9X,'03',9X,'04',8X,'05',8X,'06'/7X,6F10.2//6X,
     2       '----> Tapez RETURN pour continuer:  ')
```

```
3000  format(A1)

      return
      end


C*************************************************
      subroutine CHECK (THETA,THETAL,THETAH,IREP)
C*************************************************
C  Cette routine verifie la limite des angles de la chaine 1.

      logical IREP

      if (THETAL.LT.THETAH) then
         IREP =  .FALSE.
         if (THETAL.LE.THETA.AND.THETA.LE.THETAH) IREP = .TRUE.
      else
         IREP =  .TRUE.
         if (THETAH.LE.THETA.AND.THETA.LE.THETAL) IREP = .FALSE.
      endif

      return
      end


C************************************
      subroutine ANGPOS (THETA)
C************************************
C  Cette routine convertit THETA (0.<THETA<360 degres).

      PII =  2.* ACOS(-1.)
10    if (THETA.GT.PII) then
         THETA =  THETA - PII
         GOTO 10
      endif
20    if (THETA.LT.0.) then
         THETA =  THETA + PII
         GOTO 20
      endif

      return
      end


C************************************
      subroutine FORKIN (X,T)
C************************************
C  Cette routine calcule la cinematique directe.

      real*4 S(6),C(6),T(6),X(6)
      common/KINEMA/REF(3),XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      EPS =  0.00001
```

```
      PI = ACOS(-1.)

C     Positions angulaires de la premiere chaine.
      do 10 I = 1, 6
         S(I) = SIN(T(I))
         C(I) = COS(T(I))
10    continue
      T23   = T(2) + T(3)
      T234  = T23  + T(4)
      S23   = SIN(T23)
      C23   = COS(T23)
      S234  = SIN(T234)
      C234  = COS(T234)
      TEMP1 = C234*C(5)*C(6) + S234*S(6)
      XN    = C(1)*TEMP1 + S(1)*S(5)*C(6)
      YN    = S(1)*TEMP1 - C(1)*S(5)*C(6)
      TEMP2 = -C234*C(5)*S(6) + S234*C(6)
      OX    = C(1)*TEMP2 - S(1)*S(5)*S(6)
      OY    = S(1)*TEMP2 + C(1)*S(5)*S(6)
      AX    = C(1)*C234*S(5) - S(1)*C(5)
      AY    = S(1)*C234*S(5) + C(1)*C(5)
      AZ    = S234*S(5)
      TEMP1 = XL(6,1)*TEMP1 + XL(5,1)*S234 + XL(3,1)*C23 + XL(2,1)*C(2)
      X(1)  = C(1)*TEMP1 + S(1)*XL(6,1)*S(5)*C(6)
      X(2)  = S(1)*TEMP1 - C(1)*XL(6,1)*S(5)*C(6)
      TEMP1 = XL(6,1)*(S234*C(5)*C(6) - C234*S(6)) - XL(5,1)*C234
      X(3)  = TEMP1 + XL(3,1)*S23 + XL(2,1)*S(2) + XL(1,1)

C     Calcul de 'PHI':
      TEMP1 = SQRT (AX**2 + AY**2)
      if (TEMP1.LT.EPS) then
         if (AZ.GT.0.) then
            PHI = 0.
         else
            PHI = PI
         endif
      else
         PHI = ATAN2(AY,AX)
      endif
      X(6) = PHI

C     Calcul de 'THETA':
      SPHI = SIN(PHI)
      CPHI = COS(PHI)
      DY = -AZ
      DX = AX*CPHI + AY*SPHI
      TEMP1 = SQRT (DX**2 + DY**2)
      if (TEMP1.LT.EPS) then
         THETA = 0.
      else
         THETA = ATAN2(DY,DX)
      endif
      X(5) = THETA
```

```
C    Calcul de 'PSI':
      DY =  XN*SPHI - YN*CPHI
      DX =  OX*SPHI - OY*CPHI
      TEMP1 =  SQRT (DX**2 + DY**2)
      if (TEMP1.LT.EPS) then
         PSI =  0.
      else
         PSI =  ATAN2(DY,DX)
      endif
      X(4) =  PSI

      return
      end


C****************************
      subroutine FORVA
C****************************
C  Cette routine calcule la vitesse et l'acceleration directes.

      real*4 XJM(6,6),XJMD(6,6),V1(6),V2(6)
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC

      CALL JMMATR (XJM,XJMD,Z1(1,1,INC),0)

C  A -  Vitesse directe:
      CALL MATMUL (ZX(1,2,INC),XJM,Z1(1,2,INC),6,6,1)

C  B -  Acceleration directe:
      CALL MATMUL (V1,XJM,Z1(1,3,INC),6,6,1)
      CALL MATMUL (V2,XJMD,Z1(1,2,INC),6,6,1)
      CALL MATADD (ZX(1,3,INC),V1,V2,6,1)

      return
      end

C*********************************************
      subroutine JMMATR (XJM,XJMD,T,IFLAG)
C*********************************************
C  Cette routine calcule la matrice Jacobienne et sa derivee.

      real*4 XJM(6,6),XJMD(6,6),T(6)

      common/KINEMA/REF(3),XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC

C  A -  Initialization:
      XL1  =  XL(1,1)
      XL2  =  XL(2,1)
      XL3  =  XL(3,1)
      XL4  =  XL(4,1)
      XL5  =  XL(5,1)
      XL6  =  XL(6,1)
```

```
T1   =  T(1)
T2   =  T(2)
T3   =  T(3)
T4   =  T(4)
T5   =  T(5)
T6   =  T(6)
S1   =  SIN(T1)
C1   =  COS(T1)
S2   =  SIN(T2)
C2   =  COS(T2)
S23  =  SIN(T2+T3)
C23  =  COS(T2+T3)
S234 =  SIN(T2+T3+T4)
C234 =  COS(T2+T3+T4)
S5   =  SIN(T5)
C5   =  COS(T5)
S6   =  SIN(T6)
C6   =  COS(T6)
A1   =  XL6*(C234*C5*C6 + S234*S6)
A2   =  XL5*S234
A3   =  XL3*C23
A4   =  XL2*C2
A5   =  XL6*S5*C6
A6   =  XL6*(-S234*C5*C6 + C234*S6)
A7   =  XL5*C234
A8   = -XL3*S23
A9   = -XL2*S2
A10  = -XL6*C234*S5*C6
A11  =  XL6*C5*C6
A12  =  XL6*(-C234*C5*S6 + S234*C6)
A13  = -XL6*S5*S6
A14  = -XL6*S234*S5*C6
A15  =  XL6*(-S234*C5*S6 - C234*C6)
A16  = -XL6*C234*C5*C6
A17  =  XL6*C5*S6
A18  =  XL6*C234*S5*S6
A19  =  XL6*S234*C5*C6
A20  =  XL6*S234*S5*S6
A21  =  C234*S5
A22  =  S234*S5
A23  =  C234*C5
A24  =  S234*C5
B2   =  A1 + A2
B3   =  B2 + A3
B4   =  B3 + A4
B7   =  A6 + A7
B8   =  B7 + A8
B9   =  B8 + A9
TD1  =  Z1(1,2,INC)
TD2  =  Z1(2,2,INC)
TD3  =  Z1(3,2,INC)
TD4  =  Z1(4,2,INC)
TD5  =  Z1(5,2,INC)
TD6  =  Z1(6,2,INC)
```

```
C  B -  La matrice Jacobienne:
      if (IFLAG.EQ.0 .OR. IFLAG.EQ.1) then
         XJM(1,1) =  -S1*B4 + C1*A5
         XJM(2,1) =   C1*B4 + S1*A5
         XJM(3,1) =   0.
         XJM(4,1) =   0.
         XJM(5,1) =   0.
         XJM(6,1) =   1.
         XJM(1,2) =   C1*B9
         XJM(2,2) =   S1*B9
         XJM(3,2) =   B4
         XJM(4,2) =   S1
         XJM(5,2) =  -C1
         XJM(6,2) =   0.
         XJM(1,3) =   C1*B8
         XJM(2,3) =   S1*B8
         XJM(3,3) =   B3
         XJM(4,3) =   S1
         XJM(5,3) =  -C1
         XJM(6,3) =   0.
         XJM(1,4) =   C1*B7
         XJM(2,4) =   S1*B7
         XJM(3,4) =   B2
         XJM(4,4) =   S1
         XJM(5,4) =  -C1
         XJM(6,4) =   0.
         XJM(1,5) =   S1*A11 + C1*A10
         XJM(2,5) =  -C1*A11 + S1*A10
         XJM(3,5) =   A14
         XJM(4,5) =   C1*S234
         XJM(5,5) =   S1*S234
         XJM(6,5) =  -C234
         XJM(1,6) =   S1*A13 + C1*A12
         XJM(2,6) =  -C1*A13 + S1*A12
         XJM(3,6) =   A15
         XJM(4,6) =   C1*A21 - S1*C5
         XJM(5,6) =   S1*A21 + C1*C5
         XJM(6,6) =   A22
      endif

C  C -  La derivee de la matrice Jacobienne:
      if (IFLAG.EQ.0 .OR. IFLAG.EQ.2) then
         TEMP     =  B9*TD2 + B8*TD3 + B7*TD4 + A10*TD5 + A12*TD6
         TEMP1    =  A5*TD1 + TEMP
         TEMP2    = -B4*TD1 + A11*TD5 + A13*TD6
         XJMD(1,1) =  C1*TEMP2 - S1*TEMP1
         XJMD(2,1) =  S1*TEMP2 + C1*TEMP1
         XJMD(3,1) =  0.
         XJMD(4,1) =  0.
         XJMD(5,1) =  0.
         XJMD(6,1) =  0.
         TEMP1    =  B4*TD2 + B3*TD3 + B2*TD4 + A14*TD5 + A15*TD6
         XJMD(1,2) = -S1*B9*TD1 - C1*TEMP1
         XJMD(2,2) =  C1*B9*TD1 - S1*TEMP1
```

```
      XJMD(3,2) =  TEMP
      XJMD(4,2) =  C1*TD1
      XJMD(5,2) =  S1*TD1
      XJMD(6,2) =  0.
      TEMP      =  B3*(TD2 + TD3) + B2*TD4 + A14*TD5 + A15*TD6
      XJMD(1,3) = -S1*B8*TD1 - C1*TEMP
      XJMD(2,3) =  C1*B8*TD1 - S1*TEMP
      XJMD(3,3) =  B8*(TD2 + TD3) + B7*TD4 + A10*TD5 + A12*TD6
      XJMD(4,3) =  C1*TD1
      XJMD(5,3) =  S1*TD1
      XJMD(6,3) =  0.
      TEMP      =  B2*(TD2 + TD3 + TD4) + A14*TD5 + A15*TD6
      XJMD(1,4) = -S1*B7*TD1 - C1*TEMP
      XJMD(2,4) =  C1*B7*TD1 - S1*TEMP
      XJMD(3,4) =  B7*(TD2 + TD3 + TD4) + A10*TD5 + A12*TD6
      XJMD(4,4) =  C1*TD1
      XJMD(5,4) =  S1*TD1
      XJMD(6,4) =  0.
      TEMP1     =  A10*TD1 + A5*TD5 + A17*TD6
      TEMP2     =  A11*TD1 - A14*(TD2 + TD3 +TD4) + A16*TD5 + A18*TD6
      XJMD(1,5) =  C1*TEMP2 - S1*TEMP1
      XJMD(2,5) =  S1*TEMP2 + C1*TEMP1
      XJMD(3,5) =  A10*(TD2 + TD3 + TD4) - A19*TD5 + A20*TD6
      TEMP1     =  S234*TD1
      TEMP2     =  C234*(TD2 + TD3 + TD4)
      XJMD(4,5) =  C1*TEMP2 - S1*TEMP1
      XJMD(5,5) =  S1*TEMP2 + C1*TEMP1
      XJMD(6,5) =  S234*(TD2 + TD3 + TD4)
      TEMP1     =  A12*TD1 + A17*TD5 + A5*TD6
      TEMP2     =  A13*TD1 - A15*(TD2 + TD3 +TD4) + A18*TD5 - A1*TD6
      XJMD(1,6) =  C1*TEMP2 - S1*TEMP1
      XJMD(2,6) =  S1*TEMP2 + C1*TEMP1
      XJMD(3,6) =  A12*(TD2 + TD3 + TD4) + A20*TD5 + A6*TD6
      TEMP1     =  A21*TD1 - S5*TD5
      TEMP2     =  -C5*TD1 - A22*(TD2 + TD3 +TD4) + A23*TD5
      XJMD(4,6) =  C1*TEMP2 - S1*TEMP1
      XJMD(5,6) =  S1*TEMP2 + C1*TEMP1
      XJMD(6,6) =  A21*(TD2 + TD3 + TD4) + A24*TD5
   endif

   return
   end


C*****************************************
      subroutine MATMUL (A,B,C,L,M,N)
C*****************************************
C  Cette routine fait la multiplication de matrice:  A =  B(L,M) * C(M,N).

      DIMENSION A(L,N), B(L,M), C(M,N)

      do 10 I =  1, L
         do 20 J = 1, N
            A(I,J) =  0.
```

```
            do 30 K =  1, M
               A(I,J) =  A(I,J) + B(I,K)*C(K,J)
30          continue
20       continue
10    continue

      return
      end


C*****************************************************
      subroutine MATINV (AI,ASTART,N,ERREUR)
C*****************************************************
C  Cette routine calcule l'inverse d'une matrice carree.

      LOGICAL ERREUR
      DIMENSION AI(N,N), A(6,6), ASTART(N,N)

      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC

      EPS    = 0.0001
      ERREUR =  .FALSE.

C  Initialisation de AI a la matrice identite:
      do 10 I =  1, N
         do 20 J =  1, N
            A (I,J) =  ASTART(I,J)
            AI(I,J) =  0.
20       continue
         AI(I,I) =  1.
10    continue

      do 30 I =  1, N
         KK =  I
         do 40 M =  I, N
            if (ABS(A(M,I)).GT.ABS(A(KK,I))) KK =  M
40       continue
         if (ABS(A(KK,I)).lt.EPS) then
C          La matrice est singuliere.
            ERREUR =  .TRUE.
         else
            if (KK.NE.I) then
C             Pivotage partiel:
              do 50 J =  1, N
                 TEMP     =  A (KK,J)
                 A(KK,J)  =  A (I,J)
                 A(I,J)   =  TEMP
                 TEMP     =  AI(KK,J)
                 AI(KK,J) =  AI(I,J)
                 AI(I,J)  =  TEMP
50            continue
            endif
```

```
C          Normalisation des elements de la diagonale a 1.:
           DIA =  A(I,I)
           do 60 J =  1, N
              A (I,J) =  A (I,J) / DIA
              AI(I,J) =  AI(I,J) / DIA
60         continue

C          Reduction des elements (exepte le pivot) a zero:
           do 70 I1 =  1, N
              if (I1.NE.I) then
                 DIA =  A(I1,I)
                 do 80 J =  1, N
                    A (I1,J) =  A (I1,J) - A (I,J)*DIA
                    AI(I1,J) =  AI(I1,J) - AI(I,J)*DIA
80               continue
              endif
70         continue
        endif
30    continue

      return
      end


C****************************************
      subroutine MATADD (A,B,C,M,N)
C****************************************
C  Cette routine additionne deux matrices:  A(M,N) =  B(M,N) + C(M,N)

      DIMENSION A(M,N), B(M,N), C(M,N)

      do 10 I =  1, M
         do 20 J =  1, N
            A(I,J) =  B(I,J) + C(I,J)
20       continue
10    continue

      return
      end

C*****************************************
      subroutine SCALAR (A,CTE,B,M,N)
C*****************************************
C  Cette routine calcule le produit de deux matrices: A(M,N) =  CTE * B(M,N).

      DIMENSION A(M,N), B(M,N)

      do 10 I =  1, M
         do 20 J =  1, N
            A (I,J) =  B(I,J)*CTE
20       continue
10    continue
      return
      end
```

```
C*****************************
      subroutine INITIA
C*****************************
C  Cette routine initialise la cinematique et la dynamique.

      character IBM*12
      real*4 XO(3),YO(3),ZO(3)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1              VAL(20,2),NS,NTP
      common/INERTI/XMASS(7,2),XINERT(3,6,2),XLC(6,2)
      common/INIT  /ALPHA(8,2),RM(8,2),RI(3,7,2),P(3,7,2),S(3,7,2)

      data XO,YO,ZO/1.,0.,0., 0.,1.,0., 0.,0.,1./
      data IBM/'HALOIBMG.DEV'/


      call SETDEV(IBM)
      call SETIEE (1)

C     A -  Convertissons des angles de degree a radian:
      call CONV1
      call CONV2

C     B -  Initialisation pour la dynamique:
      PI =  ACOS(-1.)
      ALPHA(1,1) =  0.
      ALPHA(2,1) =  PI/2.
      ALPHA(3,1) =  0.
      ALPHA(4,1) =  0.
      ALPHA(5,1) =  PI/2.
      ALPHA(6,1) =  PI/2.
      ALPHA(7,1) =  0.
      ALPHA(8,1) =  0.
      ALPHA(1,2) =  0.
      ALPHA(2,2) =  PI/2.
      ALPHA(3,2) =  0.
      ALPHA(4,2) =  0.
      ALPHA(5,2) =  0.
      ALPHA(6,2) =  0.
      ALPHA(7,2) =  0.
      ALPHA(8,2) =  0.

C     - Chaine 1:
      do 10 J =  1, 6
         I =  J + 1
         RM (I,1)  =  XMASS(J,1)
         RI(1,I,1) =  XINERT(1,J,1)
         RI(2,I,1) =  XINERT(2,J,1)
         RI(3,I,1) =  XINERT(3,J,1)
10    continue
      RM (8,1)   =  XMASS (7,1)
      call SCALAR (P(1,2,1),XL(1,1),YO,3,1)
      call SCALAR (P(1,3,1),XL(2,1),XO,3,1)
```

```
      call SCALAR (P(1,4,1),XL(3,1),XO,3,1)
      call SCALAR (P(1,5,1),XL(4,1),ZO,3,1)
      call SCALAR (P(1,6,1),XL(5,1),YO,3,1)
      call SCALAR (P(1,7,1),XL(6,1),XO,3,1)
      TEMP =  XLC(1,1) - XL(1,1)
      call SCALAR (S(1,2,1),TEMP,YO,3,1)
      TEMP =  XLC(2,1) - XL(2,1)
      call SCALAR (S(1,3,1),TEMP,XO,3,1)
      TEMP =  XLC(3,1) - XL(3,1)
      call SCALAR (S(1,4,1),TEMP,XO,3,1)
      TEMP =  XLC(4,1) - XL(4,1)
      call SCALAR (S(1,5,1),TEMP,ZO,3,1)
      TEMP =  XLC(5,1) - XL(5,1)
      call SCALAR (S(1,6,1),TEMP,YO,3,1)
      TEMP =  XLC(6,1) - XL(6,1)
      call SCALAR (S(1,7,1),TEMP,XO,3,1)

c     - Chaine 2:
      J       =  1
      I       =  J + 1
      RM (I,2)  =  0.
      RI(1,I,2) =  0.
      RI(2,I,2) =  0.
      RI(3,I,2) =  0.
      do 20 J =  1, 2
         I =  J + 2
         RM (I,2)  =  XMASS(J,2)
         RI(1,I,2) =  XINERT(1,J,2)
         RI(2,I,2) =  XINERT(2,J,2)
         RI(3,I,2) =  XINERT(3,J,2)
20    continue
      call SCALAR (P(1,2,2),XL(1,1),YO,3,1)
      call SCALAR (P(1,3,2),XL(1,2),XO,3,1)
      call SCALAR (P(1,4,2),XL(2,2),XO,3,1)
      TEMP =  XLC(1,1) - XL(1,1)
      call SCALAR (S(1,2,2),TEMP,YO,3,1)
      TEMP =  XLC(1,2) - XL(1,2)
      call SCALAR (S(1,3,2),TEMP,XO,3,1)
      TEMP =  XLC(2,2) - XL(2,2)
      call SCALAR (S(1,4,2),TEMP,XO,3,1)

      return
      end


C*****************************
      subroutine CDYNA
C*****************************
C  Cette routine gere la dynamique.

      real*4 TF1(7),TF2(7),TA(7),TG(7),TORQUE(6,200),T1(8),T2(8),TD(7),
     1       TDD(7)
```

```
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
      common/TORQUE/TORQUE

      write (6,1000)

C  1 -  Calculs 'Off-line':
C    Etape 1:  ouvrir la chaine au joint '2'.
C    Etape 2:  determine " M = 2 " contraintes holonomique independante.
C    Etape 3:  renumerote les joints.


C  2 -  Calculs 'On-line':

      do 10 INC =  1, NPS
C       Etape 4:  l'algorithme recurent de Luh-Walker-Paul.
C       -  chaine 1:
         N =  6
         do 20 J =  1, N
            I =  J + 1
            T1 (I)  =  Z1(J,1,INC)
            TD (I)  =  Z1(J,2,INC)
            TDD(I)  =  Z1(J,3,INC)
20       continue
         call NEWTON (TF1,TA,TG,N,T1,TD,TDD,1)

C        -  chaine 2:
         N =  3
         J       =  1
         I       =  J + 1
         T2 (I)  =  Z1(1,1,INC)
         TD (I)  =  Z1(1,2,INC)
         TDD(I)  =  Z1(1,3,INC)
         do 30 J =  1, 2
            I =  J + 2
            T2 (I)  =  Z2(J,1,INC)
            TD (I)  =  Z2(J,2,INC)
            TDD(I)  =  Z2(J,3,INC)
30       continue
         call NEWTON (TF2,TA,TG,N,T2,TD,TDD,2)

         call INIT2 (T1,T2)

         call CHAINE (TORQUE(1,INC),TF1,TF2)
10    continue

C     Ecriture des resultats dans le fichier FORT7:
C      OPEN (7,FILE='FORT7')
         write (7,2000)
         do 40  INC =  1, NPS
            write (7,2100) INC,(TORQUE(I,INC),I=1,6)
40       continue
         REWIND (7)
      CLOSE (7)
```

```
1000  format(/6X,'- Debut de la dynamique ',32('.'))
2000  format(//' D -  MOMENTS DE TORSION:'/6X,18('-')/)
2100  format(6X,I3,2X,6(F7.2,3X))

      return
      end


C*************************************************************
      subroutine NEWTON (TF,TA,TG,N,T,TD,TDD,ICHAIN)
C*************************************************************
C  Cette routine calcule les moments de torsion pour un robot avec une chaine
C  fermee.

      real*4 R(3,3),RT(3,3),V(3,7),VD(3,7),W(3,7),WD(3,7),T(8),TD(7)
      real*4 TDD(7),XI(3,3),F(3,7),RN(3,7),SF(3,8),REPVT(1,3),SN(3,8)
      real*4 TF(7),TA(7),TG(7),WPI(3),VDC(3),REPV1(3),REPV2(3),REPV3(3)
      real*4 ZO(3),VDG(3,7),FG(3,7),SFG(3,8),SNG(3,8),REPV4(3),REPV5(3)
      real*4 REPV6(3)

      common/INIT  /ALPHA(8,2),RM(8,2),RI(3,7,2),P(3,7,2),S(3,7,2)

      data V,VD,W,WD,XI/93*0./
      data VDG/2*0.,9.80621,18*0./
      data ZO/0.,0.,1./

C  'R'    :  matrice de rotation reliant le systeme d'axe 'i' par rapport au
C            systeme d'axe 'i-1'.
C  'V (i)':  vitesse lineaire du systeme d'axe 'i-1'.
C  'VD(i)':  acceleration lineaire du systeme d'axe 'i-1'.
C  'T(i)  :  position angulaire partielle du systeme d'axe 'i-1'.
C  'TD(i) :  vitesse angulaire partielle du systeme d'axe 'i-1'.
C  'TDD(i):  acceleration angulaire partielle du systeme d'axe 'i-1'.
C  'W (i)':  vitesse angulaire total du systeme d'axe 'i-1'.
C  'WD(i)':  acceleration angulaire total du systeme d'axe 'i-1'.
C  'TF(i) :  moment de torsion total pour la chaine ICHAIN (TF =  TA + TG)
C  'TA(i) :  moment de torsion du a l'acceleration des corps.
C  'TG(i) :  moment de torsion du a la gravite.
C  'ZO'   :  vecteur unite suivant 'Z'.

C  Initialisation:
      T(N+2) =  0.
      do 30 I =  1, 3
         SN (I,N+2) =  0.
         SNG(I,N+2) =  0.
         SF (I,N+2) =  0.
         SFG(I,N+2) =  0.
30    continue


C  A -  Etape directe:

      I =  1
40    continue
```

```
C       - Calculons 'Wi+1':
        call SCALAR (WPI,TD(I+1),ZO,3,1)
        call MATADD (REPV1,W(1,I),WPI,3,1)
        call RHD (R,RT,T(I+1),ALPHA(I+1,ICHAIN))
        call MATMUL (W(1,I+1),RT,REPV1,3,3,1)

C       - Calculons 'WDi+1':
        call CROSS (REPV1,W(1,I),WPI)
        call SCALAR(REPV2,TDD(I+1),ZO,3,1)
        call MATADD(REPV3,REPV1,REPV2,3,1)
        call MATADD(REPV1,REPV3,WD(1,I),3,1)
        call MATMUL(WD(1,I+1),RT,REPV1,3,3,1)

C       - Calculons 'VDi+1':
        call MATMUL(REPV1,RT,VD(1,I),3,3,1)
        call CROSS (REPV2,WD(1,I+1),P(1,I+1,ICHAIN))
        call MATADD(REPV3,REPV1,REPV2,3,1)
        call CROSS (REPV1,W(1,I+1),P(1,I+1,ICHAIN))
        call CROSS (REPV2,W(1,I+1),REPV1)
        call MATADD(VD(1,I+1),REPV3,REPV2,3,1)
        call MATMUL(VDG(1,I+1),RT,VDG(1,I),3,3,1)

C       - Calculons 'VDC':
        call CROSS (REPV1,WD(1,I+1),S(1,I+1,ICHAIN))
        call MATADD(REPV2,VD(1,I+1),REPV1,3,1)
        call CROSS (REPV3,W(1,I+1),S(1,I+1,ICHAIN))
        call CROSS (REPV1,W(1,I+1),REPV3)
        call MATADD(VDC,REPV1,REPV2,3,1)

C       - Calculons 'Fi+1':
        call SCALAR(F(1,I+1),RM(I+1,ICHAIN),VDC,3,1)
        call SCALAR(FG(1,I+1),RM(I+1,ICHAIN),VDG(1,I+1),3,1)

C       - Calculons 'Ni+1':
        XI(1,1) =  RI(1,I+1,ICHAIN)
        XI(2,2) =  RI(2,I+1,ICHAIN)
        XI(3,3) =  RI(3,I+1,ICHAIN)
        call MATMUL(REPV1,XI,WD(1,I+1),3,3,1)
        call MATMUL(REPV2,XI,W (1,I+1),3,3,1)
        call CROSS (REPV3,W(1,I+1),REPV2)
        call MATADD(RN(1,I+1),REPV1,REPV3,3,1)
        if (I.EQ.N) then
           continue
        else
           I =  I + 1
           goto 40
        endif
      continue


C  B -  Etape inverse:

      I =  N + 1
```

```
      if (ICHAIN.EQ.1) then
         call SCALAR(SF(1,N+2),RM(8,1),VD(1,I),3,1)
         call SCALAR(SFG(1,N+2),RM(8,1),VDG(1,I),3,1)
      endif
50    continue

C       - Calculons 'fi':
        call RHD (R,RT,T(I+1),ALPHA(I+1,ICHAIN))
        call MATMUL(REPV1,R,SF(1,I+1),3,3,1)
        call MATADD(SF(1,I),REPV1,F(1,I),3,1)
        call MATMUL(REPV1,R,SFG(1,I+1),3,3,1)
        call MATADD(SFG(1,I),REPV1,FG(1,I),3,1)

C       - Calculons 'ni':
        call MATMUL(REPV1,RT,P(1,I,ICHAIN),3,3,1)
        call CROSS (REPV2,REPV1,SF(1,I+1))
        call CROSS (REPV4,REPV1,SFG(1,I+1))
        call MATADD(REPV3,REPV2,SN(1,I+1),3,1)
        call MATADD(REPV5,REPV4,SNG(1,I+1),3,1)
        call MATMUL(REPV1,R,REPV3,3,3,1)
        call MATMUL(REPV6,R,REPV5,3,3,1)
        call MATADD(REPV2,P(1,I,ICHAIN),S(1,I,ICHAIN),3,1)
        call CROSS (REPV3,REPV2,F(1,I))
        call CROSS (REPV5,REPV2,FG(1,I))
        call MATADD(REPV2,REPV1,REPV3,3,1)
        call MATADD(SNG(1,I),REPV6,REPV5,3,1)
        call MATADD(SN(1,I),REPV2,RN(1,I),3,1)

C       - Calculons 't':
        call RHD (R,RT,T(I),ALPHA(I,ICHAIN))
        call MATMUL(REPV1,R,SN(1,I),3,3,1)
        call MATTRA(REPVT,REPV1,3,1)
        call MATMUL(TA(I),REPVT,ZO,1,3,1)
        call MATMUL(REPV1,R,SNG(1,I),3,3,1)
        call MATTRA(REPVT,REPV1,3,1)
        call MATMUL(TG(I),REPVT,ZO,1,3,1)
        TF(I) =  TA(I) + TG(I)
        if (I.EQ.2) then
           continue
        else
           I =  I - 1
           goto 50
        endif
      continue

      return
      end


C*******************************************************
      subroutine CHAINE (TORQUE,TF1,TF2)
C*******************************************************
C  Cette routine applique la contrainte pour un robot avec une boucle fermee.
```

```
      real*4 TFM(2),TF(6),TF1(7),TF2(7),TORQUE(6),V1(6),V2(6),XLAMDA(2)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP
      common/INITDC/DC(6,2),DC1(2,2)

C     Etape 5:  les multiples de Lagrange 'XLAMDA'.
      TFM(1)   =  TF2(4)
      TFM(2)   =  TF1(4)
      call MATMUL(XLAMDA,DC1,TFM,2,2,1)

C     Etape 6:  moment de torsion pour la chaine fermee.
      TF(1)   =  TF1(2) + TF2(2)
      TF(2)   =  TF1(3)
      TF(3)   =  TF2(3)
      TF(4)   =  TF1(5)
      TF(5)   =  TF1(6)
      TF(6)   =  TF1(7)
      call MATMUL(V1,DC,XLAMDA,6,2,1)
      call SCALAR(V2,-1.,V1,6,1)
      call MATADD(TORQUE,TF,V2,6,1)

      return
      end


C************************************
      subroutine INIT2 (T1,T20)
C************************************
C  Cette routine initialise DC et DC1 pour la routine CHAINE.

      real*4 T1(8),T20(8),DC(6,2),DC1(2,2)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP
      common/INITDC/DC,DC1

C     La matrice DC1:
      T2  =  T1(3)
      T3  =  T1(4)
      T11 =  T20(3)
      T22 =  T20(4)
      T23 =  T2 + T3
      T12 =  T11 + T22
      BETA =  T23 - T12
      TEMP =  1./(XL(1,2)*XL(2,2)*SIN(BETA))
      DC1(1,1) =  XL(1,2)*COS(T23) * TEMP
      DC1(2,1) =  XL(1,2)*SIN(T23) * TEMP
      DC1(1,2) = -XL(2,2)*COS(T12) * TEMP
      DC1(2,2) = -XL(2,2)*SIN(T12) * TEMP

C     La matrice DC:
      DC(1,1) =  0.
      DC(1,2) =  0.
```

```
      DC(2,1) = -XL(1,2)*SIN(T23) + XL(2,1)*SIN(T2)
      DC(2,2) =  XL(1,2)*COS(T23) - XL(2,1)*COS(T2)
      DC(3,1) = -XL(2,2)*SIN(T12) - XL(1,2)*SIN(T11)
      DC(3,2) =  XL(2,2)*COS(T12) + XL(1,2)*COS(T11)
      DC(4,1) =  0.
      DC(4,2) =  0.
      DC(5,1) =  0.
      DC(5,2) =  0.
      DC(6,1) =  0.
      DC(6,2) =  0.

      return
      end


C**************************************
      subroutine MATTRA (RT,R,M,N)
C**************************************
C  Cette routine transpose une matrice 'R (M,N)'.

      DIMENSION RT(N,M),R(M,N)

      do 10 I =  1, M
         do 20 J = 1, N
            RT(J,I) =  R(I,J)
20       continue
10    continue

      return
      end


C************************************
      subroutine CROSS (C,A,B)
C************************************
C  Cette routine calcule le produit vectoriel:  C =  A x B.

      real*4 A(3), B(3), C(3)

      AX =  A(1)
      AY =  A(2)
      AZ =  A(3)
      BX =  B(1)
      BY =  B(2)
      BZ =  B(3)
      C(1) =  AY*BZ - AZ*BY
      C(2) =  AZ*BX - AX*BZ
      C(3) =  AX*BY - AY*BX

      return
      end
```

```
C*********************************************
      subroutine RHD (R,RT,THETA,ALPHA)
C*********************************************
C  Cette routine calcule les matrices de rotation en utilisant la notation
C  de Denavitt-Hartenberg.  'R' est une matrice qui fait la rotation d'un
C  vecteur exprime dans le systeme d'axe 'i' dans un nouveau systeme d'axe
C  parallele au systeme 'i-1'.

      real*4 R(3,3),RT(3,3)

      STHETA =  SIN (THETA)
      CTHETA =  COS (THETA)
      SALPHA =  SIN (ALPHA)
      CALPHA =  COS (ALPHA)

C     - Calculons 'R':
      R(1,1) =  CTHETA
      R(1,2) = -CALPHA*STHETA
      R(1,3) =  SALPHA*STHETA
      R(2,1) =  STHETA
      R(2,2) =  CALPHA*CTHETA
      R(2,3) = -SALPHA*CTHETA
      R(3,1) =  0.
      R(3,2) =  SALPHA
      R(3,3) =  CALPHA

C     - Calculons 'R' transpose:
      call MATTRA(RT,R,3,3)

      return
      end

C*****************************
      subroutine PLOTTR
C*****************************
C  Cette routine gere le menu principal pour le graphique.

      character CREP*1

      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC


C  - Montrer le menu:

      JM =  1
10    if (JM.NE.3) then
         call INITGR(1)
         call CLOSEG
         write(6,1000)
         read(5,*) JM
         if (JM.NE.3) then
            call INITGR(1)
            call CLOSEG
```

```
C          3-D simulation:
           if (JM.EQ.1) then
              call SIM3D
           endif

C          2-D simulation:
           if (JM.EQ.2) then
              call SIM2D
           endif
         endif
         goto 10
       endif

1000   format (//'1 ',77('_')/' /',77X,'\'/33X,'MENU:  graphiques.'/
      1        ' \',77('_'),'/'///6X,'1 -  mouvement cartesian.'//6X,
      2        '2 -  mouvement des joints.'//6X,'3 -  FIN:  retour au ',
      3        'menu principal.'///6X,
      4        '----> Entrez un chiffre (1 a 3):  ')

       return
       end


C*****************************
       subroutine SIM2D
C*****************************
C  Cette routine fait les graphiques en deux dimensions (mouvement des joints
C  et les torques obtenus).

       character CJ2*19(4),CX*11,CY1*8(4),CY2*8(4),CY3*8(4)
       character CN*1(6),CXMIN1*9,CYMIN1*9,CXMAX1*9,CYMAX1*9,CREP*1
       real*4 BTIME(30),TORQUE(6,200),X(200),Y(200)

       common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
       common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
       common/TORQUE/TORQUE
       common/TIMES/ BTIME,NBT

       data CJ2/ 'positions.          ','vitesses.          ',
      1          'accelerations.     ','moments de torsion.'/
       data CX/'Temps (sec)'/
       data CY1/'        ','.       ','..      ','        '/
       data CY2/'    T   ','    T   ','    T   ','    t   '/
       data CY3/'  (rad) ','  (r/s) ',' (r/s²)','  (N•m) '/
       data CN /'1','2','3','4','5','6'/

       NPS1 =  NPS - 1
       EPS  =  0.0001
       J2 =  1
20     if (J2.NE.5) then
          call INITGR(1)
          call CLOSEG
          write(6,2000)
          read(5,*) J2
```

```
      if (J2.NE.5) then
         J21 =  1
25       if (J21.NE.7) then
            call INITGR(1)
            call CLOSEG
            write(6,2100) CJ2(J2)
            write(6,2110) (I,I,I=1,6)
            write(6,2120)
            read(5,*) J21
            if (J21.NE.7) then

C             - Trouvons les valeurs minimales et maximales en X et Y:

C               - Initialisons les valeurs en 'X':
                XMIN1 =  0.
                XMAX1 =  TTOTAL
                if (J2.NE.4) then

C               - Initialisons les valeurs en 'Y':
                   YMIN1 =  Z1(J21,J2,1)
                   YMAX1 =  YMIN1
                   do 30 I =  2, NPS
                      Y1 =  Z1(J21,J2,I)
                      YMIN1 =  AMIN1 (YMIN1,Y1)
                      YMAX1 =  AMAX1 (YMAX1,Y1)
30                 continue
                else
                   YMIN1 =  TORQUE(J21,1)
                   YMAX1 =  YMIN1
                   do 40 I =  2, NPS
                      Y1 =  TORQUE(J21,I)
                      YMIN1 =  AMIN1 (YMIN1,Y1)
                      YMAX1 =  AMAX1 (YMAX1,Y1)
40                 continue
                endif
                DX1   =  XMAX1 - XMIN1
                DY1   =  YMAX1 - YMIN1
                if (DX1.GT.EPS .AND. DY1.GT.EPS) then
                   XMAX2 =  XMAX1 + 28.*DX1/524.
                   YMAX2 =  YMAX1 + 12.*DY1/152.
                   XMIN2 =  XMIN1 - 88.*DX1/524.
                   YMIN2 =  YMIN1 - 36.*DY1/152.
                   DX2   =  XMAX2 - XMIN2
                   DY2   =  YMAX2 - YMIN2
                   call INITGR(1)
                   call SETWOR(XMIN2,YMIN2,XMAX2,YMAX2)
                   XMAX3 =  XMAX2 - 2.*DX2/640.
                   YMAX3 =  YMAX2 - 1.*DY2/200.

C                  Dessinons les boites pour la presentation:
                   call BOX (XMIN2,YMIN2,XMAX3,YMAX3)
                   call BOX (XMIN1,YMIN1,XMAX1,YMAX1)
                   call PTABS (XMIN1,YMAX1)
                   X1 =  XMIN1 - 4.*DX2/640.
```

```
      call LNABS (X1,YMAX1)
      call PTABS (XMIN1,YMIN1)
      call LNABS (X1,YMIN1)
      call PTABS (XMIN1,YMIN1)
      Y1 =  YMIN1 - 2.*DY2/200.
      call LNABS (XMIN1,Y1)
      call PTABS (XMAX1,YMIN1)
      call LNABS (XMAX1,Y1)

C     Identification de l'axe des 'X'.
      X1 =  XMAX2 - 96.*DX2/640.
      Y1 =  YMIN2 +  8.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CX)

C     Identification de l'axe des 'Y'.
      X1 =  XMIN2 +  8.*DX2/640.
      Y1 =  (YMAX1 + YMIN1)/2. + 24.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CY1(J2))
      Y1 =  Y1 - 8.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CY2(J2))
      Y1 =  Y1 - 16.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CY3(J2))
      X1 =  XMIN2 +  48.*DX2/640.
      Y1 =  Y1 + 12.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CN(J21))

C     Ecrivons les valeurs maximales:
      OPEN (2,FILE='BIDON')
         write (2,2200) XMIN1,YMIN1,XMAX1,YMAX1
         REWIND (2)
         read(2,2300)CXMIN1,CYMIN1,CXMAX1,CYMAX1
         REWIND (2)
      CLOSE (2)

C     - Ecrivons XMIN:
      X1 =  XMIN1 - 48.*DX2/640.
      Y1 =  YMIN2 + 20.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CXMIN1)

C     - Ecrivons XMAX:
      X1 =  XMAX1 - 60.*DX2/640.
      call MOVTCA (X1,Y1)
      call TEXT (CXMAX1)

C     - Ecrivons YMIN:
      X1 =  XMIN2 + 8.*DX2/640.
      Y1 =  YMIN1 - 4.*DY2/200.
      call MOVTCA (X1,Y1)
```

```
                    call TEXT (CYMIN1)

C                  - Ecrivons YMAX:
                    Y1 =  YMAX1 - 4.*DY2/200.
                    call MOVTCA (X1,Y1)
                    call TEXT (CYMAX1)
                    X1 =  2.*XMAX2
                    Y1 =  2.*YMAX2
                    call MOVTCA (X1,Y1)

C                   Dessinons la courbe obtenue:
                    if (J2.NE.4) then
                       do 50 I =  1, NPS
                          Y(I) =  Z1(J21,J2,I)
                          X(I) =  TTOTAL*real(I-1)/real(NPS1)
50                     continue
                    else
                       do 60 I =  1, NPS
                          Y(I) =  TORQUE(J21,I)
                          X(I) =  TTOTAL*real(I-1)/real(NPS1)
60                     continue
                    endif
                    NBTM1 =  NBT - 1
                    if (NBTM1.ge.1) then
                       call SETLNS (2)
                       do 70 I =  1, NBTM1
                          call PTABS(BTIME(I),YMAX1)
                          call LNABS(BTIME(I),YMIN1)
70                     continue
                    endif
                    call SETLNS (1)
                    call PTABS(X(1),Y(1))
                    call POLYLA (X(1),Y(1),NPS)
                    read (5,3000) CREP
                 else
                    call CLOSEG
                    write (6,4000) XMIN1,XMAX1,YMIN1,YMAX1
                    read (5,3000) CREP
                 endif
              endif
              goto 25
           endif
        endif
        goto 20
     endif

2000  format (/'1 ',77('_')/' /',77X,'\'/33X,'MENU:  mouvement des ',
     1        'joints.'/' \',77(' '),'/'///6X,'1 -  Positions.'//6X,
     2        '2 -  Vitesses.'//6X,'3 -  Accelerations.'//6X,'4 -  ',
     3        'Moment de torsion.'//6X,'5 -  FIN:  retour au menu ',
     4        'precedent.'///6X,'----> Entrez un chiffre (1 a 5):  ')
2100  format (//'1 ',77('_')/' /',77X,'\'/30X,'MENU:  ',A19/
     1        ' \',77('_'),'/'/)
2110  format(6X,I1,' -  Joint ',I1,'.'/)
```

```
2120  format(6X,'7 -  FIN.'///6X,'----->  Entrez un chiffre (1 a 7):  ')
2200  format(4(F9.4))
2300  format(4(A9))
3000  format(A1)
4000  format(//' PAS DE GRAPHIQUE:'//6X,'XMIN =  ',F9.4//6X,'XMAX =  ',
     1       F9.4//6X,'YMIN =  ',F9.4//6X,'YMAX =  ',F9.4///
     2       6X,'----> Tapez la touche RETURN pour continuer:  ')

      return
      end


C*****************************
      subroutine SIM3D
C*****************************
C Cette routine permet la simulation en 3D.

      character CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3
      real VISION (3)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),VAL(20,2),NS,
     1              NTP
      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
      common/INERTI/XMASS(7,2),XINERT(3,6,2),XLC(6,2)
      common/INITI/CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT
      common/VISION/VISION,RHO,FOCUS
      common/MAX/XMIN,XMAX,YMIN,YMAX


C 1 - Donnee pour le centrage du dessin:

C     - limitation reelle du robot a dessiner:
      XMIN  =  - XL(2,1) - XL(3,1) - XL(5,1) - XL(6,1)
      XMAX  =  - XMIN
      YMIN  =  XL(1,1) + XMIN*5./4.
      YMAX  =  0.865*(XMAX-XMIN) + YMIN


C 2 -  affichons le menu principal:
      ICHOIX = 0
C     while "I" not equal "3", alors ce n'est pas la fin.
3     if (ICHOIX.ne.3) then
         call CLOSEG
         write (*,1000)
         write (*,1001)
         write (*,1002)
         write (*,1003)
         read (*,*) ICHOIX
         if (ICHOIX.EQ.1) call PART1
         if (ICHOIX.EQ.2) call PART2
         goto 3
      endif
```

```
1000  format (2X,77(' ')/' /',77X,'\'/37X,'MENU',
     4        / ' \',77(' '),'/'/)
1001  format (6X,'1 -  FAIRE la simulation. '/)
1002  format (6X,'2 -  Changer certains parametres pour la simulation.')
1003  format (/6X,'3 -  Retour au menu PRECEDENT.'//' ',79('-')//
     5        ' ----> Entrer un chiffre:  ')

      return
      end


C*****************************
      subroutine PART1
C*****************************
*  Cette routine permet de faire la simulation.

      integer IV(3)
      character CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3
      character*10 CVISIO,CFOCUS,CRHO
      real VISION (3),POS3D(3,10),POS2D(2,10),T(6),PB3(3),PB2(2)
      logical LROB,LARM,LSURF,LPROJ,LVISIT,LEFFAC

      common/TRAJEC/TDM(6),ADM(6),XJERK(6),TORMAX(6),NPS,E,ERROR
      common/CUR/ZX(6,3,200),Z1(6,3,200),Z2(3,3,200),INC,TTOTAL,TINC
      common/INITI/ CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT
      common/VISION/VISION,RHO,FOCUS
      common/MAX/XMIN1,XMAX1,YMIN1,YMAX1


c  A -  Preparons l'ecran pour la presentation.

      DX1   =  XMAX1 - XMIN1
      DY1   =  YMAX1 - YMIN1
      XMAX2 =  XMAX1
      YMAX2 =  YMAX1
      XMIN2 =  XMIN1 - 112.*DX1/536.
      YMIN2 =  YMIN1 - 12.*DY1/188.
      DX2   =  XMAX2 - XMIN2
      DY2   =  YMAX2 - YMIN2
      call INITGR(1)
      call SETWOR(XMIN2,YMIN2,XMAX2,YMAX2)

C     Dessinons des boites:
      YMAX3 =  YMAX1 - 1.*DY2/200.
      call BOX (XMIN2,YMIN1,XMIN1,YMAX3)
      YMIN3 =  YMIN1 + DY1/3.
      YMAX3 =  YMAX1 - DY1/3.
      call BOX (XMIN2,YMIN3,XMIN1,YMAX3)

C     Axe de vision:
      X1 =  XMIN2 + 12.*DX2/640.
      X11=  XMIN2 + 16.*DX2/640.
      Y1 =  YMAX2 - 16.*DY2/200.
```

```
      call MOVTCA (X11,Y1)
      call TEXT ('   axe')
      Y1 =  YMAX2 - 24.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT ('    de')
      Y1 =  YMAX2 - 32.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT ('  VISION')

C     La distance focale:
      Y1 =  YMAX2 - 80.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT ('    la')
      Y1 =  YMAX2 - 88.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (' distance')
      Y1 =  YMAX2 - 96.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT ('  FOCALE')

C     La distance de l'observateur:
      Y1 =  YMAX2 - 144.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (' distance')
      Y1 =  YMAX2 - 152.*DY2/200.
      call MOVTCA (X11,Y1)
      call TEXT ('  de l''')
      X111=  XMIN2 + 8.*DX2/640.
      Y1  =  YMAX2 - 160.*DY2/200.
      call MOVTCA (X111,Y1)
      call TEXT ('OBSERVATEUR')

C     Ecriture des valeurs a l'ecran:
      IV (1) =  INT (VISION(1))
      IV (2) =  INT (VISION(2))
      IV (3) =  INT (VISION(3))
      OPEN (2,FILE='BIDON')
         write (2,1000) (IV(I),I=1,3),FOCUS,RHO
         REWIND (2)
         read(2,1100) CVISIO,CFOCUS,CRHO
         REWIND (2)
      CLOSE (2)
      Y1 =  YMAX2 - 48.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CVISIO)
      Y1 =  YMAX2 - 112.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CFOCUS)
      Y1 =  YMAX2 - 176.*DY2/200.
      call MOVTCA (X1,Y1)
      call TEXT (CRHO)
      X1 =  113./640.
      Y1 =  1./199.
      X2 =  638./639.
```

```
      Y2 =  186./199.
      call SETWOR(XMIN1,YMIN1,XMAX1,YMAX1)
      call SETVIE(X1,Y1,X2,Y2,1,-1)
      call AXE3D

      RADIUS =  6.*DX2/640.
      VALUE  =  DX1/20.
      LEFFAC =  CEFFAC.EQ.'Oui'
      LROB   =  CROB  .EQ.'Oui'
      LARM   =  CARM  .EQ.'Oui'
      LVISIT =  CVISIT.EQ.'Oui'
      LPROJ  =  CPROJ .EQ.'Oui'
      LSURF  =  CSURF .EQ.'Oui'


C  B -  Dessinons la trajectoire:

      INC =  1
      T(1) =  Z1(1,1,INC)
      T(2) =  Z1(2,1,INC)
      T(3) =  Z1(3,1,INC)
      T(4) =  Z1(4,1,INC)
      T(5) =  Z1(5,1,INC)
      T(6) =  Z1(6,1,INC)
      call POSI (POS2D,POS3D,T)
      if (LARM) then
         N =  3
      else
         N =  10
      endif
      do 10 I =  1, 2
         PB3 (I)   =  POS3D (I,N)
         PB2 (I)   =  POS2D (I,N)
10    continue
      PB3 (3)   =  POS3D (3,N)
      if (LSURF) then
         IPROJ =  1
         call SURF(PB3,POS3D(1,N),PB2,POS2D(1,N),IPROJ)
         IPROJ =  0
      endif
      DIST  =  0.
      do 20 INC =  2, NPS
         T(1) =  Z1(1,1,INC)
         T(2) =  Z1(2,1,INC)
         T(3) =  Z1(3,1,INC)
         T(4) =  Z1(4,1,INC)
         T(5) =  Z1(5,1,INC)
         T(6) =  Z1(6,1,INC)
         call POSI (POS2D,POS3D,T)
         call PTABS (PB2(1),PB2(2))
         call LNABS (POS2D(1,N),POS2D(2,N))
         if (LSURF) then
            D    =  SQRT ((POS3D(1,N)-PB3(1))**2+(POS3D(2,N)-PB3(2))**2
     1                  + (POS3D(3,N)-PB3(3))**2)
```

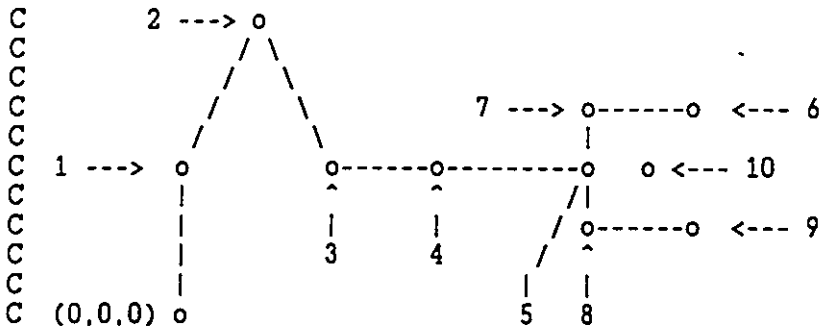
```
            DIST =  DIST + D
            if (DIST.GE.VALUE .OR. INC.EQ.NPS) then
               IPROJ =  1
               DIST =  0.
            else
               IPROJ =  0
            endif
            call SURF(PB3,POS3D(1,N),PB2,POS2D(1,N),IPROJ)
         endif
         do 30 I =  1, 2
            PB2(I) =  POS2D (I,N)
            PB3(I) =  POS3D (I,N)
30       continue
         PB3(3) =  POS3D(3,N)
20    continue

      if (LVISIT .OR. LPROJ .OR. LROB) then
         call PV (RADIUS,N)
      endif

C     Regenerons l'ecran:
      call AXE3D
      X1 =  XMIN2 + 4.*DX2/640.
      Y1 =  YMIN2 + 1.*DY2/200.
      call SETVIE(0.,0.,1.,1.,-1,-1)
      call MOVTCA (X1,Y1)
      call TEXT ('----> Tapez la touche RETURN pour continuer:  ')
      read  (*,2000) C
      call SETCOL (1)

1000  format ('(',I2,',',I2,',',I2,')',1X,F5.1,' m. ',1X,F5.1,' m. ')
1100  format (3A10)
2000  format (A1)

      return
      end


C*****************************
      subroutine PART2
C*****************************
C Cette routine permet de changer les parametres pour la simulation graphique.
C Elle se divise en deux parties:  la premiere option permet de choisir le
C type de simulation et la seconde modifie les parametres de dessin.

      character C*1,CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3
      real VISION (3)

      common/INITI/CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT
      common /VISION/  VISION,RHO,FOCUS

      ICHANG =  0
      ICHOIX =  0
```

```
3     if (ICHOIX.ne.3) then
         call CLOSEG
         write (6,500)
         write (6,501)
         write (6,502)
         write (6,503)
         read (5,*) ICHOIX
         if (ICHOIX.EQ.1) then
            IC1 =  0
18          if (IC1.NE.7) then
               call CLOSEG
               write (*,700)
               write (*,701) CROB
               write (*,702) CEFFAC
               write (*,703) CVISIT
               write (*,704) CARM
               write (*,705) CPROJ
               write (*,706) CSURF
               write (*,707)
               read (*,*) IC1
               if (IC1.eq.1) then
                  if (CROB.eq.'Oui') then
                     CROB =  'Non'
                  else
                     CROB =  'Oui'
                  endif
               endif

               if (CROB.EQ.'Non') CEFFAC =  'Non'

               if (IC1.eq.2) then
                  call CLOSEG
                  write (6,8000)
                  write (6,8001) CEFFAC,TWAIT
                  write (6,6002)
                  read  (5,7000) C
                  if (C.eq.'1') then
                     if (CEFFAC.eq.'Oui') then
                        CEFFAC =  'Non'
                     else
                        CEFFAC =  'Oui'
                     endif
                  else
                     call CLOSEG
                     write (6,3002)
                     read  (5,*) TWAIT
                  endif
               endif

               if (IC1.eq.3) then
                  if (CVISIT.eq.'Oui') then
                     CVISIT =  'Non'
                  else
                     CVISIT =  'Oui'
```

```
               endif
            endif

            if (IC1.eq.4) then
               if (CARM.eq.'Oui') then
                  CARM =  'Non'
               else
                  CARM =  'Oui'
               endif
            endif

            if (IC1.eq.5) then
               if (CPROJ.eq.'Oui') then
                  CPROJ =  'Non'
               else
                  CPROJ =  'Oui'
               endif
            endif
            if (IC1.eq.6) then
               call CLOSEG
               write (6,6000)
               write (6,6001) CSURF,IPNTY
               write (6,6002)
               read  (5,7000) C
               if (C.eq.'1') then
                  if (CSURF.eq.'Oui') then
                     CSURF =  'Non'
                  else
                     CSURF =  'Oui'
                  endif
               else
                  call CLOSEG
                  CSURF =  'Oui'
                  write (6,3002)
                  read  (5,*) IPNTY
               endif
            endif
            goto 18
         endif
      endif

      if (ICHOIX.EQ.2) then
         IC1 =  0
28       if (IC1.NE.4) then
            call CLOSEG
            write (6,1000)
            write (6,1001)  (VISION (I), I =  1, 3)
            write (6,1002)  RHO
            write (6,1003)  FOCUS
            write (6,1004)
            read (5,*) IC1
            if (IC1.NE.4) ICHANG =  ICHANG + 1
            if (IC1.eq.1) then
               call CLOSEG
```

```
                write (6,3000)
                write (6,3001) (VISION (I), I =  1, 3)
                write (6,3002)
                read  (5,*) (VISION (I), I =  1, 3)
             endif

             if (IC1.eq.2) then
                call CLOSEG
                write (6,4000)
                write (6,4001) RHO
                write (6,3002)
                read  (5,*) RHO
             endif

             if (IC1.eq.3) then
                call CLOSEG
                write (6,5000)
                write (6,4001) FOCUS
                write (6,3002)
                read  (5,*) FOCUS
             endif
             goto 28
          endif
       endif
       goto 3
    endif

    call CLOSEG

500  format (2X,77('_')/' /',77X,'\'/28X,'SOUS-MENU:  parametres.',
   1        /  ' \',77('_'),'/'/)
501  format (6X,'1 -  Changer le TYPE de simulation.'/)
502  format (6X,'2 -  Changer les PARAMETRES de simulation.'/)
503  format (6X,'3 -  Retour au MENU PRECEDENT.'//' ',79('-')//
   2        ' ----> Entrez un chiffre:  ')
700  format (2X,77('_')/' /',77X,'\'/28X,'SOUS-MENU:  type de robot',
   3        /  ' \',77('_'),'/'/)
701  format (6X,'1 -  Robot avec membre lineaire:'/
   4        11X,'[valeur par defaut:  ',A3,'].')
702  format (6X,'2 -  Membre precedent du robot sera efface:'/
   6        11X,'[valeur par defaut:  ',A3,'].')
703  format (6X,'3 -  Laisser une marque pour la TRAJECTOIRE:'/
   7        11X,'[valeur par defaut:  ',A3,'].')
704  format (6X,'4 -  Trajectoire du porteur:'/
   5        11X,'[valeur par defaut:  ',A3,'].')
705  format (6X,'5 -  PROJECTION de la trajectoire sur le plan X-Y:'/
   8        11X,'[valeur par defaut:  ',A3,'].')
706  format (6X,'6 -  PROJECTION d''une surface sur le plan X-Y:'/
   9        11X,'[valeur par defaut:  ',A3,'].')
707  format (6X,'7 -  Retour au MENU PRECEDENT.'/' ',79('-')//
   a        ' ----> Entrez un chiffre:  ')

1000 format (2X,77('_')/' /',77X,'\'/29X,'SOUS-MENU:  parametres.',
   b        /  ' \',77('_'),'/'/)
```

```
1001  format (6X,'1 -  Changer l''AXE de vision:'/11X,'[valeur par ',
     c        'defaut:  (',F5.1,',',F5.1,',',F5.1,')].'/)
1002  format (6X,'2 -  Changer la DISTANCE entre l''observateur et le ',
     d       'robot:'/11X,'[valeur par defaut:  ',F7.2,' metre(s)].'/)
1003  format (6X,'3 -  Changer la position du FOCUS:'/11X,'[valeur ',
     e        'par defaut:  ',F7.2,' metre(s)].'/)
1004  format (6X,'4 -  Retour au MENU PRECEDENT.'//' ',79('-')//
     f        ' ----> Entrez un chiffre:  ')
2000  format ('+',A1,'*  Vous devez entrer un nombre entre 1 et ',
     g        I1,'. *')
3000  format (2X,77('_')/' /',77X,'\'/27X,'Changer l''AXE de vision.'
     h        /' \',77('_'),'/'//)
3001  format (6X,'Valeurs par defaut (',F7.3,',',F7.3,',',F7.3,').')
3002  format (/' ----> Entrez la(les) nouvelle(s) valeur(s):  ')
4000  format (2X,77('_')/' /',77X,'\'/23X,'Changer la DISTANCE de ',
     i        'l''observateur.'/' \',77(' '),'/'//)
4001  format (6X,'Valeur par defaut:  ',F9.3,' metre(s).')
5000  format (2X,77('_')/' /',77X,'\'/25X,'Changer la position du ',
     j        'FOCUS.'/' \',77(' '),'/'//)
6000  format (2X,77('_')/' /',77X,'\'/15X,'Changer la valeur de defaut',
     k        ' ou le nombre de division.'/' \',77('_'),'/'//)
6001  format (6X,'1 -  Courbe projetee:  [',A3,'].'//6X,'2 -  Nombre',
     l        ' de divisions selon l''axe ''Z'':  [',I5,'].'/)
6002  format (1X,79('_')//' ---->  Choisir un chiffre:  ')
7000  format (A1)
8000  format (2X,77('_')/' /',77X,'\'/17X,'Changer la valeur de defaut',
     k        ' ou le temps d''attente.'/' \',77('_'),'/'//)
8001  format (6X,'1 -  Effacer la position precente:  [',A3,'].'//6X,
     l '2 -  Le temps d''attente avant d''effacer:  [',F5.1,' sec.].'/)

      return
      end


C************************************
      subroutine WAIT (TWAIT)
C************************************

      IWAIT =  5000 * INT (TWAIT + 0.45)

      do 10 I =  1, IWAIT
         X =  2.**2.
10    continue

      return
      end

C*******************************************
      subroutine POSI (POS2D,POS3D,T)
C*******************************************
C Cette routine calcule les 10 points necessaires pour dessiner le robot.  Les
C points sont calcules en coordonnees reelles et gardees dans POS.
```




```
      real POS3D(3,10),POS2D(2,10),T(6)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      T1   =  T(1)
      T2   =  T(2)
      T3   =  T(3)
      T4   =  T(4)
      T5   =  T(5)
      T6   =  T(6)
      S1   =  SIN(T1)
      S2   =  SIN(T2)
      S5   =  SIN(T5)
      S6   =  SIN(T6)
      S23  =  SIN(T2 + T3)
      S234 =  SIN(T2 + T3 + T4)
      C1   =  COS(T1)
      C2   =  COS(T2)
      C5   =  COS(T5)
      C6   =  COS(T6)
      C23  =  COS(T2 + T3)
      C234 =  COS(T2 + T3 + T4)

      POS3D(1,1) =  0.
      POS3D(2,1) =  0.
      POS3D(3,1) =  XL(1,1)
      POS3D(1,2) =  XL(2,1)*C1*C2
      POS3D(2,2) =  XL(2,1)*S1*C2
      POS3D(3,2) =  XL(2,1)*S2 + XL(1,1)
      TEMP       =  XL(3,1)*C23 + XL(2,1)*C2
      POS3D(1,3) =  C1*TEMP
      POS3D(2,3) =  S1*TEMP
      POS3D(3,3) =  XL(3,1)*S23 + POS3D(3,2)
      TEMP       =  XL(5,1)*S234 + TEMP
      POS3D(1,4) =  C1*TEMP
      POS3D(2,4) =  S1*TEMP
      POS3D(3,4) = -XL(5,1)*C234 + POS3D(3,3)
      TEMP1      =  XL(6,1)*(C234*C5*C6 + S234*S6)/2. + TEMP
      TEMP2      =  XL(6,1)*S5*C6/2.
      POS3D(1,5) =  C1*TEMP1 + S1*TEMP2
```

```
      POS3D(2,5) =  S1*TEMP1 - C1*TEMP2
      POS3D(3,5) =  XL(6,1)*(S234*C5*C6 - C234*S6)/2. + POS3D(3,4)
      TEMP1      =  XL(6,1)*(C234*C5*C6 + S234*S6) + TEMP
      TEMP2      =  TEMP2 * 2.
      POS3D(1,10)=  C1*TEMP1 + S1*TEMP2
      POS3D(2,10)=  S1*TEMP1 - C1*TEMP2
      POS3D(3,10)=  XL(6,1)*(S234*C5*C6 - C234*S6)/2. + POS3D(3,4)

C     Le vecteur 'o' donnant l'orientation de la pince est:
      TEMP1      = -C234*C5*S6 + S234*C6
      TEMP2      =  S5*S6
      OX         =  C1*TEMP1 - S1*TEMP2
      OY         =  S1*TEMP1 + C1*TEMP2
      OZ         = -S234*C5*S6 - C234*C6

C     Le vecteur 'n' donnant l'approche de la pince est:
      TEMP1      =  C234*C5*C6 + S234*S6
      TEMP2      =  S5*C6
      AX         =  C1*TEMP1 + S1*TEMP2
      AY         =  S1*TEMP1 - C1*TEMP2
      AZ         =  S234*C5*C6 - C234*S6

      POS3D(1,7) =  POS3D(1,5) + XL(6,1)*OX
      POS3D(2,7) =  POS3D(2,5) + XL(6,1)*OY
      POS3D(3,7) =  POS3D(3,5) + XL(6,1)*OZ
      POS3D(1,8) =  POS3D(1,5) - XL(6,1)*OX
      POS3D(2,8) =  POS3D(2,5) - XL(6,1)*OY
      POS3D(3,8) =  POS3D(3,5) - XL(6,1)*OZ
      POS3D(1,6) =  POS3D(1,7) + XL(6,1)*AX
      POS3D(2,6) =  POS3D(2,7) + XL(6,1)*AY
      POS3D(3,6) =  POS3D(3,7) + XL(6,1)*AZ
      POS3D(1,9) =  POS3D(1,8) + XL(6,1)*AX
      POS3D(2,9) =  POS3D(2,8) + XL(6,1)*AY
      POS3D(3,9) =  POS3D(3,8) + XL(6,1)*AZ

C     Projetons les points en 3D en 2D:
      do 10 I =  1, 10
         call SCREEN (POS3D(1,I),POS3D(2,I),POS3D(3,I),POS2D(1,I),
     1               POS2D(2,I))
10    continue

      return
      end


C*****************************************
      subroutine LINK (POS2D,K,RADIUS)
C*****************************************

      real POS2D(2,10)

      call SETCOL (K)
```

```
C     Dessinons les membres:
      call SCREEN(0.,0.,0.,X0,Y0)
      call PTABS (X0,Y0)
      call LNABS (POS2D(1,1),POS2D(2,1))
      call LNABS (POS2D(1,2),POS2D(2,2))
      call LNABS (POS2D(1,3),POS2D(2,3))
      call LNABS (POS2D(1,4),POS2D(2,4))
      call LNABS (POS2D(1,5),POS2D(2,5))
      call PTABS (POS2D(1,6),POS2D(2,6))
      call LNABS (POS2D(1,7),POS2D(2,7))
      call LNABS (POS2D(1,8),POS2D(2,8))
      call LNABS (POS2D(1,9),POS2D(2,9))

C     Dessinons les joints:
      call PTABS (POS2D(1,1),POS2D(2,1))
      call CIR (RADIUS)
      call PTABS (POS2D(1,2),POS2D(2,2))
      call CIR (RADIUS)
      call PTABS (POS2D(1,3),POS2D(2,3))
      call CIR (RADIUS)
      call PTABS (POS2D(1,4),POS2D(2,4))
      call CIR (RADIUS)

      return
      end


C************************************
      subroutine  PV (RADIUS,N)
C************************************

      character CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3
      real POS3D(3,10), POS2D(2,10)

      common/KINEMA/XL(6,2),TMAX(6,2),XYT(6,8),TT(6,8,2),
     1           VAL(20,2),NS,NTP

      common/INITI/CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT

      do 10 I =  1, NTP
         call POSI (POS2D,POS3D,TT(1,I,1))
         call SCREEN (POS3D(1,N),POS3D(2,N),0.,X1,Y1)
         if (CROB.EQ.'Oui') then
            call LINK (POS2D,1,RADIUS)
         endif
         if (CVISIT.eq.'Oui') then
            call PTABS (POS2D(1,N),POS2D(2,N))
            call CIR (RADIUS)
         endif
         if (CPROJ.eq.'Oui') then
            call SETLNS (2)
            call PTABS (POS2D(1,N),POS2D(2,N))
            call LNABS (X1,Y1)
```

```
            call SETLNS (1)
         endif
         if (CEFFAC.EQ.'Oui') then
            call WAIT (TWAIT)
            call LINK (POS2D,0,RADIUS)
         endif
10    continue

      return
      end


C*****************************************************
      subroutine SURF (PB3,PE3,PB2,PE2,IPROJ)
C*****************************************************

      character CVISIT*3,CPROJ*3,CSURF*3,CROB*3,CARM*3,CEFFAC*3
      real PB3(3),PE3(3),PB2(2),PE2(2)

      common/INITI/CVISIT,CPROJ,CSURF,CROB,CARM,CEFFAC
      common/INITI2/IPNTY,TWAIT

C     Initialisation:
        XPB =  PB2(1)
        YPB =  PB2(2)
        XPE =  PE2(1)
        YPE =  PE2(2)
        call SCREEN (PB3(1),PB3(2),0.,XMB,YMB)
        call SCREEN (PE3(1),PE3(2),0.,XME,YME)

C     Dessinons la projection sur le plan X-Y:
        if (IPROJ.EQ.1) then
           call PTABS (XME,YME)
           call LNABS (XPE,YPE)
        endif

C     Dessinons les division le long de la trajectoire:
        call PTABS (XMB,YMB)
        call LNABS (XME,YME)
        DXB =  XPB - XMB
        DYB =  YPB - YMB
        DXE =  XPE - XME
        DYE =  YPE - YME
        do 10 I =  1, IPNTY
           XB =  XMB + (real(I)*DXB) / real(IPNTY)
           YB =  YMB + (real(I)*DYB) / real(IPNTY)
           XE =  XME + (real(I)*DXE) / real(IPNTY)
           YE =  YME + (real(I)*DYE) / real(IPNTY)
           call PTABS (XB,YB)
           call LNABS (XE,YE)
10    continue

      return
      end
```

```
C*********************************************
      subroutine SCREEN (X,Y,Z,XS,YS)
C*********************************************
C  Cette routine permet de faire la convertion des coordonnees reelles en
C  trois dimensions a des coordonnees reelles en deux dimensions.

      real VISION(3)

      common /VISION/ VISION,RHO,FOCUS

      EPS =  0.0001

C  A - Les coordonnees dans le systeme d'axe de l'oeil:

C    Position the l'oeil par rapport au systeme de reference:
      R =  SQRT (VISION(1)**2 + VISION(2)**2 + VISION(3)**2)
      RH =  - (VISION (1) * RHO) / R
      RK =  - (VISION (2) * RHO) / R
      RL =  - (VISION (3) * RHO) / R

C    Position de l'oeil en coordonnees spheriques:
      DY    =  SQRT (RH**2 + RK**2)
      IF (DY.LT.EPS) THEN
         THETA =  ATAN2 (1.,0.)
      ELSE
         THETA =  ATAN2 (RK,RH)
      ENDIF
      PHI   =  ATAN2 (DY,RL)

C    Les nouvelles coordonnees:
      S1 =  SIN (THETA)
      S2 =  SIN (PHI)
      C1 =  COS (THETA)
      C2 =  COS (PHI)
      XE =  -X*S1 + Y*C1
      YE =  -X*C1*C2 - Y*S1*C2 + Z*S2
      ZE =  -X*C1*S2 - Y*S1*S2 - Z*C2 + RHO


C  B - Les coordonnes dans les coordonnees de l'ecran:

      XS =  (RHO - FOCUS) * (XE/ZE)
      YS =  (RHO - FOCUS) * (YE/ZE)

      return
      end


C*****************************
      subroutine AXE3D
C*****************************

      character*1 CH
```

```
      common/MAX/XMIN,XMAX,YMIN,YMAX

      RO =  XMAX / 2.5

      call SETCOL (1)


C  A -  L'axe 'X':
      CH =  'x'
      call AXIS (RO,0.,0.,CH)

C  B -  L'axe 'Y':
      CH =  'y'
      call AXIS (0.,RO,0.,CH)

C  C -  L'axe 'Z':
      CH =  'z'
      call AXIS (0.,0.,RO,CH)

C D -  Le plan de reference 'X-Y':
      call SCREEN (XMAX,XMAX,0.,X1,Y1)
      call SCREEN (XMAX,-XMAX,0.,X2,Y2)
      call PTABS (X1,Y1)
      call LNABS (X2,Y2)

      call SCREEN (-XMAX,-XMAX,0.,X1,Y1)
      call LNABS  (X1,Y1)

      call SCREEN (-XMAX,XMAX,0 ,X1,Y1)
      call LNABS  (X1,Y1)

      call SCREEN (XMAX,XMAX,0.,X1,Y1)
      call LNABS  (X1,Y1)

      return
      end


C*******************************************************
      subroutine AXIS (X,Y,Z,CH)
C*******************************************************

      character*1 CH
      real VISION (3)

      common/MAX/XMIN,XMAX,YMIN,YMAX
      common/VISION/VISION,RHO,FOCUS

      EPS =  0.0001


C  A -  dessinons l'axe:

      call SCREEN (X,Y,Z,X1,Y1)
```

```
      call SCREEN (0.,0.,0.,X0,Y0)
      call PTABS  (X0,Y0)
      call LNABS  (X1,Y1)


C  B -  dessinons la tete de la fleche:

      R1 =  8./10.
      R3 =  XMAX / 30.
      R  =  3.*(YMAX - YMIN)/199.
      A225  =  225.*ACOS(-1.)/180.

C     a) normalisons le vecteur vision:
      D =  SQRT (VISION(1)**2 + VISION(2)**2 + VISION(3)**2)
      VX =  VISION (1) / D
      VY =  VISION (2) / D
      VZ =  VISION (3) / D

C     b) trouvons un vecteur perpendiculaire a l'axe et a la ligne de vision:
      V1 =  Y*VZ - Z*VY
      V2 =  Z*VX - X*VZ
      V3 =  X*VY - Y*VX
      D =  SQRT (V1**2 + V2**2 + V3**2)

C     c) dessinons la pointe de la fleche:
      if (D.GT.EPS) then
         VX =  X*R1 + V1*R3
         VY =  Y*R1 + V2*R3
         VZ =  Z*R1 + V3*R3
         call SCREEN (VX,VY,VZ,X2,Y2)
         call LNABS  (X2,Y2)

         VX =  X*R1 - V1*R3
         VY =  Y*R1 - V2*R3
         VZ =  Z*R1 - V3*R3
         call SCREEN (VX,VY,VZ,X2,Y2)
         call PTABS  (X1,Y1)
         call LNABS  (X2,Y2)

C     d) positionnons la lettre de l'axe:
         VX =  1.4 * X
         VY =  1.4 * Y
         VZ =  1.4 * Z
         call SCREEN (VX,VY,VZ,X2,Y2)
         X1 =  X2 + R*COS(A225)
         Y1 =  Y2 + R*SIN(A225)
         call MOVTCA (X1,Y1)
         call TEXT (CH)
      endif

      return
      end
```

BIBLIOGRAPHIE

[1] ANGELES, Jorge. "Iterative Kinematic Inversion of General Five-Axis Robot Manipulators", The international Journal of Robotics Research, vol. 4, no. 4, Hiver 1986, p. 59-70.

[2] ASADA, H. et J.-J. E. SLOTINE. Robot analysis and control, Etat-Unis, John Wiley and Sons, 1986, 266 p.

[3] BENHABIB, B., et al. "Optimal Continuous Path Planning for Seven-Degrees-of-Freedom Robots", Journal of Engineering for Industry, vol. 108, août 1986, p. 213-218.

[4] BOBROW, J.E., et al. "On The Optimal Control of Robotic Manipulators with Actuator Constraints", Proceeding of American Control Conference, juin 1983, p. 782-787.

[5] BRADY, M., et al. Basics of Robot Motion Planning and Control, Cambridge, Massachusetts, Etat-Unis, MIT press.

[6] CHAND, Sujeet, et Keith L. DOTY. "On-line Polynomial Trajectories for Robot Manipulators", The International Journal of Robotics Research, vol. 4, no. 2, été 1985, p. 38-48.

[7] CLARKE, F.H. CALCULUS OF VARIATIONS (Lecture notes), Montréal, s.éd., 1986.

[8] CRAIG, J.J. Introduction to Robotics: Mechanics & Control, Massachusetts, Etat-Unis, Addison-Wesley publishing co., 1986, 303 p.

[9] CUGY, A. LES ROBOTS: spécifications techniques, France, Hermes, 1983, 255 p.

[10] DUBOWSKY, S., et D.T. DESFORGES. "The application of Model-Referenced Adaptive Control to Robotic Manipulators", Journal of Dynamic Systems, Measurements, and Control (ASME), vol. 101, septembre 1979, p. 193-200.

[11] DUBOWSKY, S., et Z. SHILLER. "Optimal Dynamic Trajectories for robotic manipulators", Theory and Practice of Robots and Manipulators", édition A. Morecki et G. Bianchi, Proceeding RomanSy 1984, Cambridge, Etat-Unis, MIT press, 1985, p. 133-144.

[12] ELGELBERGER, J.F. ROBOTICS IN PRACTICE: Management and applications of industrial robots, Grande-Bretagne, Anchor Press, 1980, 291 p.

[13] ERDMAN, Arthur G., et George N. SANDOR. MECHANISM DESIGN: Analysis and Synthesis, Etat-Unis, Prentice-Hall, 1984, 2 vol., 530 p.

[14] FAHIM, A., et al. "Robot Trajectory Optimisation with Dynamic Contraints", The International Journal of Advanced Manufacturing Technology, vol. 3, 1988, p. 71-76.

[15] FEATHERSTONE, R. "Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles", The International Journal of Robotics Research, vol. 2, no. 2, été 1983, p. 35-45.

[16] GEERING, Hans P., et al. "Time-Optimal Motions of Robots in Assembly Tasks", IEEE Transactions on Automatic Control, vol. AC-31, no. 6, juin 1986, p. 512-518.

[17] GILBERT, Elmer G., et Daniel W. JOHNSON. "Distance Functions and Their Application to Robot Path Planning in the Presence of Obstacles", IEEE Journal of Robotics and Automation, vol. RA-1, no.1, mars 1985, p. 21-30.

[18] GROOVER, P.M., et al. Industrial Robotics: technology, programming, and applications, Etat-Unis, McGraw-Hill, 1986, 546 p.

[19] GROOVER, P.M. Automation, Production Systems, and Computer-Integrated Manufacturing, Etat-Unis, Prentice-Hall, 1987, 808 p.

[20] HANAFI, A., et al. "Optimal Trajectory Control of Robotic Manipulators", Mechanism and Machine Theory, vol. 19, no. 2, 1984, p.267-273.

[21] HOLLERBACH, John M. "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative study of Dynamics Formulation Complexity", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-10, no. 11, novembre 1980, p. 730-736.

[22] HOLLERBACH, John M. "Dynamic Scaling of Manipulator Trajectories", Journal of Dynamic Systems, Measurement, and Control (ASME), vol. 106, mars 1984, p. 102-106.

[23] KAHN, M.E., et B. ROTH. "The Near-Minimum-Time Control Of Open-Loop Articulated Kinematic Chains", Journal of Dynamic Systems, Measurement, and Control (ASME), septembre 1971, p. 164-172.

[24] KANE, Thomas R., et FAESSLER. "Dynamics of Robots and Manipulators Involving Closed Loops", Manipulators Romansy, juin 1984.

[25] KANE, Thomas R., et David A. Levinson. "The Use of Kane's Dynamical Equations in Robotics", The International Journal of Robotics Research, vol. 2, no. 3, automne 1983, p. 3-21.

[26] KANT, Kamal, et Steven W. ZUCKER. "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition", The International Journal of Robotics Research, vol. 5, no. 3, automne 1986, p. 72-89.

[27] KIM, Byung Kook, et Kang G. SHIN. "Minimum-Time Path Planning for Robot Arms and Their Dynamics", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 2, mars/avril 1985, p. 213-223.

[28] KIM, Byung Kook, et Kang G. SHIN. "Suboptimal Control of Industrial manipulators with a Weighted Minimum Time-Fuel Criterion", IEEE Transactions on Automatic Control, vol. AC-30, no. 1, janvier 1985, p. 1-10.

[29] KOHLI, D., et J. SPANOS. "Workspace Analysis of Mechanical Manipulators Using Polynomial Discriminants", Journal of Mechanisms, Transmissions, and Automation in Design, vol. 107, juin 1985, p. 209-215.

[30] KRUT'KO, P.D., et YE.P. POPOV. "Kinematic Algortihms for Manipulating Robot Movement Control", Engineering Cybernetic, vol. 17, no. 4, 1981, p. 65-75.

[31] KRUT'KO, P.D., et YE.P. POPOV. "Motion Control of the Manipulating Robots Based on Second-Order Kinematic Algorithms", Engineering Cybernetic, vol. 19, no. 6, 1982, p. 89-97.

[32] LEE, C.S. George. "Robot Arm Kinematics, Dynamics, and Control", Computer, vol. 15, no. 12, décembre 1982, p. 62-80.

[33] LEE, C.S. George, et al. Tutorial on ROBOTICS, s.l., Computer Society press (IEEE), s.d., 275 p.

[34] LEE, C.S. George, et M.J. CHUNG, "An Adaptive Control Strategy for Mechanical Manipulators", IEEE Transactions on Automatic Control, p 243-249.

[35] LIN, Chun-Shin, et al. "Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots", IEEE Transactions on Automatic Control, vol. AC-28, no.12, décembre 1983, p. 1066-1073.

[36] LIN, Chun-Shin, et Po-Rong CHANG. "Joint Trajectories of Mechanical Manipulators for Cartesian Path Approximation", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 6, novembre/décembre 1983, p. 1094-1102.

[37] LIN, Chun-Shin, et Po-Rong CHANG. "Approximate Optimum Paths of Robot Manipulators Under Realistic Physical Constraints", Proceeding IEEE International Conference Robotics and Automation, mars 1985, p. 737-742.

[38] LOPEZ, P., et J.-N. FOULC. Introduction à la robotique, Paris, EdiTests, 1984, 2 vol., 318 p.

[39] LOZANO-PEREZ, Tomas. "Spatial Planning: A Configuration Space Approach", IEEE Transactions on Computers, vol. C-32, no. 2, février 1983, p. 108-120.

[40] LUH, John Y.S. "Resolved-Acceleration Control of Mechanical Manipulators", IEEE Transactions on Automatic Control, vol. AC-25, no. 3, juin 1980, p. 468-474.

[41] LUH, John Y.S., et al. "On-line Computational Scheme for Mechanical Manipulators", Journal of Dynamic Systems, Measurement, and Control (ASME), vol. 102, 1980, p. 69-76.

[42] LUH, John Y.S., et Yuan-Fang ZHENG. "Computation of Input Generalized Forces for Robots with Closed Kinematic Chain Mechanisms", IEEE Journal of Robotics and Automation, vol. RA-1, no. 2, juin 1985, p. 95-103.

[43] LUH, John Y.S., et C.S. LIN. "Approximate Joint Trajectories for Control of Industrial Robots Along Cartesian Paths", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-14, no. 3, mai/juin 1984, p. 444-450.

[44] LUH, John Y.S., et C.S. LIN. "Optimum Path Planning for Mechanical Manipulators", Journal of Dynamic Systems, Measurement, and Control (ASME), vol. 102, juin 1981, p. 142-151.

[45] MEGAHED, S., et M. RENAUD. "Dynamic Modelling of Robot Manipulators Containing Closed Kinematic Chains", Advanced Software in Robotics, 1984, p. 147-158.

[46] MYERS, R.E. Microcomputer GRAPHICS for the Apple Computer, s.l., s. éd., 1982.

[47] PAUL, B. "Analytical Dynamics of Mechanisms - A computer Oriented Overview", Mechanism and Machine Theory, vol. 10, 1975, p. 481-507.

[48] PAUL, Richard P. et B.E. SHIMANO. "COMPLIANCE AND CONTROL", Proceedings of the Joint Automatic Control Conference, 1976, p. 694-699.

[49] PAUL, Richard P. "Manipulator Cartesian Path Control", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-9, no. 11, novembre 1979, p. 702-711.

[50] PAUL, Richard P. Robot Manipulators: mathematics, programming, and control, Etat-Unis, MIT press, 1981, 279 p.

[51] PAUL, Richard P., et al. "Kinematic Control Equations for Simple Manipulators", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-11, no. 6, juin 1981, p. 449-455.

[52] PAUL, Richard P., et al. "Differential Kinematic Control Equations for Simple Manipulators", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-11, no. 6, juin 1981, p. 456-460.

[53] PAUL, Richard P., et Charles N. Stevenson. "Kinematics of Robot Wrists", The International Journal of Robotics Research, vol. 2, no. 1, printemps 1983, p. 31-38.

[54] RAIBERT, M.H., et J.J. CRAIG. "Hybrid Position/Force Control of Manipulators", Journal of Dynamic Systems, Measurement and Control (ASME), vol. 102, juin 1981, p. 126-133.

[55] RAJAN, V.T. "Planning of Minimum-Time Trajectories for Robot Arms", Proceeding IEEE International Conference Robotics and Automation, mars 1985, p. 759-764.

[56] SAHAR, Gideon, et John M. HOLLERBACH. "Planning of Minimum-Time Trajectories for Robot Arms", Proceeding IEEE International Conference Robotics and Automation, mars 1985, p. 751-758.

[57] SAHAR, Gideon, et John M. HOLLERBACH. "Planning of Minimum-Time Trajectories for Robot Arms", The International Journal of Robotics Research, vol. 5, no. 3, automne 1986.

[58] SALISBURY, J. Kenneth. "ACTIVE STIFFNESS CONTROL OF A MANIPULATOR IN CARTESIAN COORDINATES", Proceedings of the 19th IEEE Conference on Decision and Control, 1980, p. 95-100.

[59] SCHARF, E.M., et al. "A Self-Organizing Algorithm for the Control of a Robot Arm", International Journal of Robotics and Automation, vol. 1, no. 1, 1986, p. 33-41.

[60] SCHEINMAN, V., et B. ROTH. "On the Optimal Selection and Placement of Manipulators", Theory and Practice of Robots and Manipulators", édition A. Morecki et G. Bianchi, Proceeding RomanSy 1984, Cambridge, Etat-Unis, MIT press, 1985, p. 39-46.

[61] SCHMITT, D., et al. "Optimal Motion Programming of Robot Manipulators", Journal of Mechanisms, Transmissions, and Automation in Design, vol. 107, juin 1985, p. 239-244.

[62] SHAHINPOOR, M. A robot engineering textbook, New York, Harper & Row edition, 1987, 480 p.

[63] SHILLER, Z., et S. DUBOWSKY. "On the Optimal Control of Robotic Manipulators with Actuator and End-Effector Constraints", Proceeding IEEE Conference Robotics and Automation, 1985, p. 614-620.

[64] SHIMANO, B., et B. ROTH. "ON FORCE SENSING INFORMATION AND ITS USE IN CONTROLLING MANIPULATORS", Proceeding of the Eight Industrial Symposium on Industrial Robots, Washington, p. 119-126.

[65] SHIN, Kang G., et Neil D. McKAY. "A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators", IEEE Transactions on Automatic Control, vol. AC-31, no. 6, juin 1986, p. 491-500.

[66] SHIN, Kang G., et Neil D. McKAY. "Minimum-Time Control of Robototic Manipulators wint Geometric Path Constraints", IEEE Transactions on Automatic Control, vol. AC-30, no. 6, juin 1985, p. 531-541.

[67] SHIN, Kang G., et Neil D. McKAY. "Selection of Near-Minimum Time Geometric Paths for Robotic Manipulators", IEEE Transactions on Automatic Control, vol. AC-31, no. 6, juin 1986, p. 501-511.

[68] SILVER, William M. "On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators", The International Journal of Robotics Research, vol. 1, no. 2, été 1982, p. 60-70.

[69] TAKASE, Kunikatsu, et al. "A structured Approach to Robot Programming and Teaching", IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-11, no. 4, avril 1981, p. 274-289.

[70] TAYLOR, Russell H. "Planning and Execution of Straight Line Manipulator Trajectories", IBM Journal of Research and Development, vol.23, juillet 1979, p. 424-436.

[71] VUKOBRATOVIC, M., et M. KIRCANSKI. "A Method for Optimal Systhesis of Manipulation Robot Trajectories", Journal of Dynamic Systems, Measurement and Control (ASME), vol. 104, juin 1982, p. 188-193.

[72] WANG, J.T., et R.L. HUSTON. "Kane's Equations With Undetermined Multipliers - Application to Constrained Multibody Systems", Journal of Applied Mechanics, vol. 54, juin 1987, p. 424-429.

[73] WHITNEY, Daniel E. "FORCE FEEDBACK CONTROL OF MANIPULATOR FINE MOTIONS", Journal of Dynamic Systems, Measurement and Control (ASME), juin 1977, p. 91-97.

[74] WHITNEY, Daniel E. "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators", Journal of Dynamic Systems, Measurement, and Control, décembre 1972, p. 303-309.

[75] WU, Chi-Haur, et Richard P. PAUL. "Resolved Motion Force Control of Robot Manipulator", IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-12, no. 3, juin 1982, p. 266-275.

[76] ZELDMAN, M.I. What every engineer should know about robots, édition Marcel Dekker, New-York, 1984, 197 p.