Robots autonomes : du concept au robot. Architectures, représentations et algorithmes

Rachid Alami

Année 1996

Notice sur les titres et travaux

présentée au

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

en vue de l'obtention de

l'Habilitation à diriger des recherches de l'Université Paul Sabatier de Toulouse
Spécialité : Robotique

par

Rachid ALAMI

Robots autonomes: du concept au robot. Architectures, représentations et algorithmes

---

Soutenue le 15 Fevrier 1996 devant le jury:

Président	**Georges**	**GIRALT**
Rapporteurs	**J.Michael**	**BRADY**
	Bernard	**ESPIAU**
	Richard P.	**PAUL**
Examinateurs	**Marc**	**COURVOISIER**
	Bernard	**DUBUISSON**
	Malik	**GHALLAB**
	Jean-Louis	**LACOMBE**

Cette habilitation a été préparée au LAAS-CNRS
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4

Rapport LAAS N° 96258

Table des matières

I AVANT-PROPOS

Mon ambition est de doter le robot d'un haut niveau de flexibilité et d'adaptation à la tâche en présence d'imprécisions et d'incertitudes liées à celle-ci et à son état interne.

Ceci se traduit par le développement de concepts et d'outils visant à permettre au robot de planifier sa tâche et de mettre en œuvre pour son exécution des *processus décisionnels bouclés sur la tâche et sur l'environnement.*

Profondément convaincu de la nécessité d'avancer à la fois:

- sur les aspects conceptuels de l'autonomie des robots;
- sur l'intégration de leurs différentes composantes décisionnelles et fonctionnelles;
- et sur leur concrétisation algorithmique, logicielle puis expérimentale dans des robots systèmes complets;

j'ai contribué à ces différents niveaux.

Ceci s'est traduit par une forte implication dans une approche de la robotique avancée, fruit d'une volonté, d'une réflexion et d'un travail d'équipe au sein du groupe RIA [1] du LAAS.

Animé par la volonté de traduire mes travaux dans une réalité tangible, celle de la robotique "avec des robots", je me suis fortement impliqué dans des projets internes au groupe RIA mais aussi dans des programmes ou projets coopératifs au niveau régional, national, européen ou international.

Je présente, dans ce qui suit, une synthèse de mes contributions et ma prospective de recherche.

Les annexes détaillent mes activités d'encadrement de jeunes chercheurs, de gestion et de participation à des contrats de recherche et à des conventions de collaboration, de recherche et développement en relation avec le milieu industriel et institutionnel, d'enseignement et d'administration de la recherche.

1. Robotique et Intelligence Artificielle

II CONTRIBUTIONS SCIENTIFIQUES

II.1 Ma problématique

Ma contribution scientifique concerne trois volets:

1. l'élaboration d'*architectures* permettant d'intégrer les composantes décisionnelle et fonctionnelle et de mettre en œuvre des processus bouclés sur l'environnement à différents niveaux d'abstraction;
2. le développement de *représentations et d'algorithmiques pour la planification et l'interprétation de plans*: planification logique et temporelle (au niveau de la mission) mais aussi planification géométrique (plus proche de la tâche);
3. la réalisation effective de *systèmes robotiques complets* démontrant les capacités développées et servant de support de validation et d'aiguillons exigeants à l'extension de ces mêmes capacités.

Je suis intimement persuadé de la complémentarité de ces trois volets et de l'intérêt de les conjuguer dans un processus d'élaboration et de maturation progressives. Ils correspondent en effet à une démarche globale en trois étapes concentriques, du plus général au plus particulier, du plus conceptuel au plus concret.

Considérons un robot mobile dont l'objectif est d'atteindre un lieu distant dans un environnement d'intérieur dont il possède une description (topologique et géométrique). Ajoutons qu'il doit être capable d'effectuer sa mission malgré une description imprécise de son environnement (forme et position des obstacles), en détectant et évitant des obstacles imprévus (fixes ou mobiles) et en coopérant avec d'autres robots mobiles du même type (croisement, dépassement, suivi).

Un tel robot est aujourd'hui une réalité tangible: une démonstration mettant en œuvre trois robots mobiles de ce type est aujourd'hui opérationnelle au LAAS.

Il est le résultat d'un long cheminement, de contributions nombreuses et d'un travail d'équipe de longue haleine. Il concrétise plusieurs de mes contributions relevant des trois volets cités plus haut.

Je présente, dans ce qui suit, une synthèse de mes contributions sous la forme de 9 "fiches" qui résument les approches et les résultats:

1. Une architecture de contrôle générique
2. Programmation et Conduite d'une Cellule Flexible d'Assemblage
3. Téléprogrammation niveau tâche et autonomie en robotique d'intervention
4. Planification de mission avec prise en compte de contraintes temporelles et du non-déterminisme

5. La planification automatique de tâches de type Prendre-et-Pose
6. Une formulation générale de la planification des tâches de manipulation
7. Planification de stratégies de déplacement pour un robot mobile en présence d'incertitudes
8. Un robot mobile autonome
9. Une approche nouvelle de la coopération multi-robots

II.2 Une architecture de contrôle générique

L'ambition ici est de définir une architecture pour le contrôle d'un robot autonome et plus généralement de concevoir des systèmes décisionnels temps réels permettant à la fois:

- l'élaboration d'un plan d'actions (processus généralement coûteux en temps calcul et non borné dans le temps);
- et la disponibilité permanente dans un environnement évolutif (réactivité), les réactions devant être cohérentes avec l'état du robot et de son environnement et avec les objectifs du plan.

Cette problématique fait l'objet depuis plusieurs années d'un débat passionnant (dans la communauté robotique et au-delà) et constitue indubitablement un élément décisif de la robotique à venir.

J'ai élaboré avec R. Chatila et G. Giralt, une *Architecture de contrôle générique* [43, 64] qui organise le robot-système en trois niveaux composés de *systèmes décisionnels temps réel* à des niveaux d'abstraction et avec des contraintes temporelles différents (figure 2). Chaque niveau est construit selon un *Paradigme pour l'intégration de la Planification et de l'Exécution* qui distingue deux types de décisions traités par des processus différents (figure 1) et mettant en œuvre des algorithmes de nature et de complexité différentes: la planification d'une part, l'interprétation de plans et la réaction aux situations d'autre part.

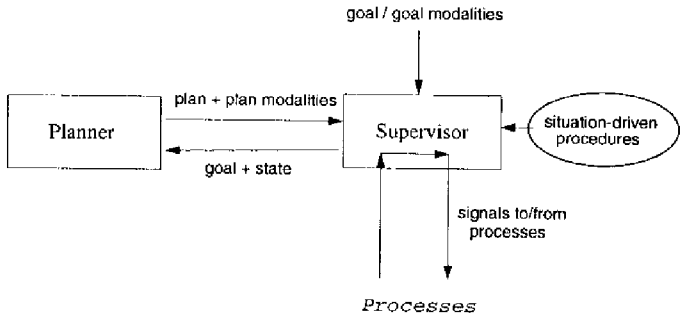


FIG. 1 – *Paradigme pour l'intégration de la Planification et de l'Exécution*

Les trois niveaux mettent en œuvre des processus décisionnels à des niveaux d'abstraction et avec des contraintes temporelles différents.

- Le niveau de planification le plus élevé permet de produire des plans d'actions correspondant à la réalisation d'un objectif donné. Le superviseur associé lui transmet l'objectif à réaliser, puis contrôle l'exécution du plan compte tenu des événements produits par l'exécution ou par une évolution de l'environnement.
- Le niveau suivant est un niveau d'affinement. L'existence de ce niveau est essentielle et est l'une des originalités de notre approche. En effet, au niveau précédent, les représentations de l'environnement et des tâches sont nécessairement incomplètes car trop abstraites. Elles ne peuvent en particulier pas exprimer toutes les interactions du robot avec son environnement.

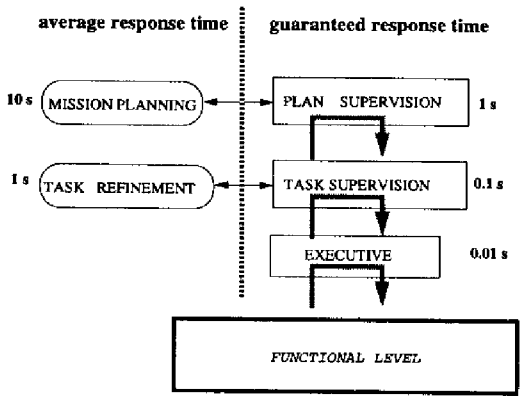


FIG. 2 – *Une architecture de contrôle générique*

- Le troisième niveau (exécution) ne comprend pas de planificateur. Il est composé d'une part d'un ensemble de modules encapsulant les fonctions de perception, de modélisation, de génération de mouvement, de commandes asservies etc., et d'autre part d'un exécutif qui contrôle l'exécution des actions. Ceci peut être notamment réalisé au moyen d'un ensemble de règles de production compilées.

Points marquants:
Cette architecture a évolué et s'est affinée au cours de divers travaux (depuis la première architecture pour une Cellule d'Assemblage jusqu'à la mise en œuvre effective d'un ensemble de plusieurs robots mobiles autonomes coopèrants). Elle a donné lieu à de nombreux débats et présentations [19, 20, 61, 43, 67] dans des sessions spéciales de conférences ou de workshops thématiques (IEEE-RA, ICAR, ISRR, ISRAM...). Elle s'est avérée opératoire en permettant:

- de concevoir et réaliser des robots dotés d'une autonomie réelle dans différents domaines d'application particulièrement contraignants.
- de mettre clairement en évidence la nécessité de prendre en compte au niveau de la planification les incertitudes liées aux actions du robot, la dynamique de l'environnement et de la tâche, ainsi que les actions faisant intervenir des boucles perception-action. C'est à ces problèmes, que je considère aujourd'hui comme centraux, que je consacre une partie de mes travaux (Cf. §II.4, §II.5, §II.8).

Cette architecture a fait l'objet de plusieurs instanciations notamment dans le cadre des projets VAP-RISP [2] (Robotique mobile d'exploration planétaire) [83], AMR-EUREKA [3] (robot d'intervention pour la protection civile) [29], MARTHA [4] (robots de transitique lourde) [89, 94]. Elle a donné lieu également à une implantation complète sur les robots HILARE 1.5 [109] et Hilare 2 et 2-bis[51, 50] ainsi que dans le cadre du projet MARTHA.

II.3 Programmation et Conduite d'une Cellule Flexible d'Assemblage

Le projet *NNS* a été développé initialement dans le cadre du programme ARA pour l'aide à la programmation et le contrôle d'exécution de tâches d'assemblage conduites sur une *Cellule Flexible d'Assemblage (CFA)* multi-robots, multi-capteurs.

La problématique de la CFA est celle de la robotique dite "à poste fixe". Elle s'intéresse à l'exécution, par un système robotisé, d'une tâche préétablie dans un environnement bien structuré et connu a priori. Les aspects essentiels sont, dans ce cas, la programmation ou plus généralement la "préparation des travaux" ainsi que l'ensemble des moyens permettant une exécution efficace et "robuste" de la tâche.

Dans cette perspective, j'ai mené et dirigé des travaux de recherche portant sur les aspects suivants:

- la définition d'une modélisation originale des tâches d'assemblage, de la cellule et de ses composants;
- la définition d'une architecture générale fondée sur cette modélisation et destinée à fournir une structure d'accueil pour l'ensemble des travaux;
- la construction de modules décisionnels spécialisés destinés à compléter progressivement l'environnement NNS: évitement d'obstacles, génération de mouvements fins, analyse des incertitudes, contrôle d'exécution, module de diagnostic et d'aide à la reprise...

Le système NNS: Le système NNS comporte un *Environnement de Programmation* (environnement hors-ligne) et un *Système de Conduite* (système en-ligne) [16, 4, 58, 5].

- *L'Environnement de programmation* inclut essentiellement des *Modules Décisionnels Spécialisés* destinés à aider le programmeur à spécifier et à raffiner progressivement une tâche d'assemblage: la structuration de l'espace de la cellule, les trajectoires d'approche, les stratégies d'insertion... L'ensemble de ces informations est introduit de manière granulaire sur la base de la modélisation définie plus haut. Des modules de vérification ou de simulation graphique permettent de valider les choix du programmeur. A partir de ces informations, un module appelé *Compilateur de Tâche d'Assemblage* produit le *Modèle d'Exécution* (thèse de H. Chochon, soutenue en novembre 1986).
- *Le Système de Conduite* est un système basé sur la connaissance (Knowledge-Based System). Il utilise le *Modèle d'Exécution* pour générer, au fur et à mesure, les consignes nécessaires à la réalisation de la tâche, à partir des informations provenant de l'environnement (arrivée d'une pièce, fin d'une action, détection d'une anomalie...). Il est constitué d'un ensemble de boucles imbriquées à des niveaux d'abstraction différents: niveau tâche, niveau action, niveau fonction.

  Le niveau *tâche* comprend notamment un *Générateur de Plans d'Exécution* dont le rôle consiste à déterminer les actions à réaliser à partir d'un état quelconque de la tâche. Il est invoqué chaque fois qu'un événement se produit dans la cellule comme le résultat d'une action non déterministe (identification, inspection), l'arrivée d'un convoyeur,un échec ou un incident (thèse de E. Lopez Mellado soutenue en décembre 1986).

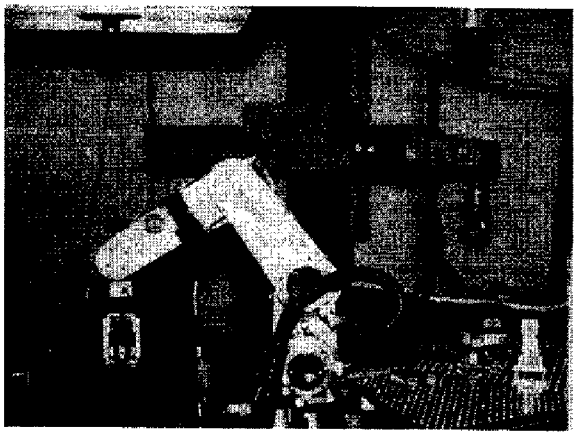


FIG. 3 – *La Cellule Flexible d'Assemblage du programme ARA*

Points marquants:
Cette approche constituait, à l'époque, une contribution originale et sur le front de l'état de l'art [4, 5].
Ce système a été entièrement implanté et a été démontré notamment dans le cadre d'expérimentations canoniques réalistes (ARA) mettant en œuvre une tâche d'assemblage réalisée par une cellule composée de deux bras manipulateurs et d'un ensemble de capteurs et de dispositifs péri-robotiques (1985, 1986).
Par ailleurs, ces travaux ont également donné lieu à:

- l'analyse formelle des séquences d'assemblage issue de la modélisation que nous avons proposée en collaboration avec Paul Freedman (Université de McGill à Montréal) lors de son séjour post-doctoral au LAAS [1, 23, 24];
- le développement d'un environnement pour la programmation automatique de tâches de prise et pose (cf. §II.6);
- l'application de la modélisation que nous avons proposée et de l'architecture de NNS à la spécification d'une tâche d'assemblage en langage ESTELLE (projet ESPRIT SEDOS en collaboration avec le groupe OLC[5] du LAAS) [59].

II.4 Téléprogrammation niveau tâche et autonomie en robotique d'intervention

Cette architecture complète l'architecture générique (présentée en §II.2) en fournissant un cadre général pour l'utilisation par l'homme d'un robot dans les cas - nombreux dés qu'on quitte des environnements spécialement aménagés - où l'autonomie totale est hors de portée (ou non souhaitée) et où la téléopération est impossible ou trop lente (limitations physiques dues à l'éloignement du robot) ou encore trop "technique".

Ainsi, j'ai élaboré, en collaboration avec R. Chatila et G. Giralt, une architecture fonctionnelle générale pour un système robotique d'intervention sur site distant [19, 20, 22, 26, 29, 28, 34].

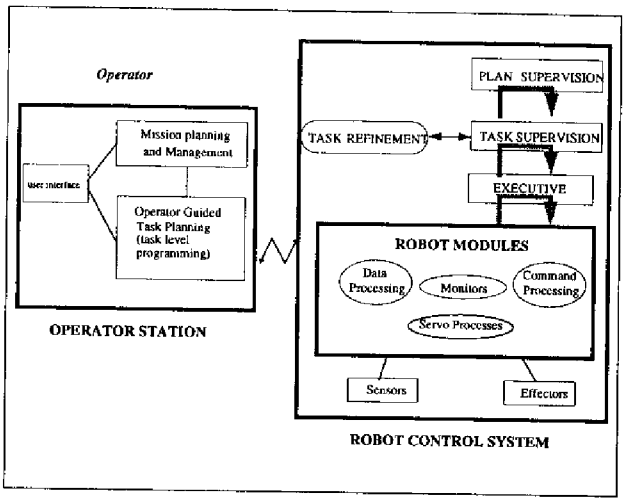


FIG. 4 – *Une architecture pour robots autonomes d'intervention distants*

Elle met en œuvre des processus décisionnels à plusieurs niveaux:

- au niveau de la préparation de la mission à accomplir en générant un plan d'action de niveau d'abstraction élevé pour atteindre l'objectif. Cette fonction est appelée *Fonction de Planification ou de re-Planification de Mission.*

- au niveau de l'affinement du plan de mission obtenu en termes de tâches interprétables par le système de contrôle du robot. Cette fonction est appelée *Fonction de (Télé)Programmation.*

- au niveau de l'interprétation et de l'exécution des tâches par le robot en interaction directe avec son environnement. Cette fonction sera appelée *Contrôle d'exécution.* Elle est assurée par le système de contrôle embarqué. Elle suppose la présence à bord de structures décisionnelles permettant d'interpréter (i.e. d'affiner) les tâches en terme de primitives exécutables par le robot et d'en contrôler l'exécution. Cette interprétation consiste en une planification à granularité plus fine et portant sur des entités directement perceptibles par la machine.

Cette architecture générale appelle les remarques suivantes:

- Le robot est une machine (télé)programmable au niveau tâche. Le degré d'abstraction et de généralité de la tâche fixe les niveaux d'autonomie décisionnelle à concevoir et à implémenter au niveau de la programmation et dans le contrôle de la tâche. En effet, les tâches doivent pouvoir être exécutées alors que l'environnement n'est que grossièrement connu au moment de leur planification. Il est alors impossible de prévoir la totalité des actions à mettre en œuvre mais également impossible à cause des délais de transmission d'avoir un mode directement téléopéré.

- l'homme "reste dans la boucle" mais à un niveau d'abstraction élevé. Il interprète la situation, raisonne sur les objectifs de la mission et programme les tâches à accomplir.

  En effet, notamment dans le cas de robots d'intervention, le robot ne dispose pas de capacités d'interprétation de situation et de génération automatique de mission. Par contre, il peut être doté d'une grande autonomie pour le déplacement (et la perception pour le déplacement) ainsi que pour l'affinement et l'exécution d'une classe de tâches dont la sémantique peut être définie de manière précise (manipulation, tâches spécifiques dépendantes de l'application...)

  De toute manière, tous les processus mettant en œuvre des asservissements ou des commandes gardées sont exécutés de manière autonome. C'est le niveau minimal d'autonomie nécessaire dans ce schéma fonctionnel.

FIG. 5 – *Le robot ADAM réalisant une tâche de navigation en environnement inconnu*

Points marquants:
Cette architecture a fait l'objet de plusieurs instanciations notamment dans le cadre des projets VAP-RISP[6](Robotique mobile d'exploration planétaire)(robotique mobile d'exploration planétaire) puis IARES-EUREKA, AMR-EUREKA[7] (robot d'intervention pour la protection civile), IFREMER (navette de diagraphie NADIA [34]).
Elle a donné lieu également à une implantation partielle sur le robot d'extérieur ADAM [42].
Elle constitue également la base de départ de notre réflexion sur l'interaction entre l'homme et la machine dans des applications telles que l'aide aux handicapés.

II.5 Planification de mission avec prise en compte de contraintes temporelles et du non-déterminisme

II.5.1 La prise en compte de contraintes temporelles

J'ai lancé avec M. Ghallab et R. Chatila une thématique nouvelle portant sur le raisonnement temporel pour la génération et le contrôle d'exécution de plans [6, 28].

En effet, de telles tâches requièrent en général la possibilité de représenter explicitement et de raisonner sur:

- des relations temporelles entre les objectifs à atteindre par le plan, en relatif (réaliser A et B simultanément, B décalé de δ par rapport à A), ou contraint par rapport à une référence absolue (terminer A avant l'instant T);
- des relations temporelles entre les actions élémentaires, en tenant compte de leur durée, et de leurs effets;
- des événements attendus, programmés (à tel instant, après tel effet), ou conditionnels (se produiront s'il y a conjugaison des effets E1 et E2).

Nous avons développé dans ce sens un formalisme et une structure de données, dite IxTeT (Table des Temps Indexée), pour générer et représenter un plan, qui inclut l'instant et l'intervalle comme structures de base. Un algorithme rapide de gestion (recherche et mise à jour) du treillis des temps a été proposé. Un planificateur a également été développé [62, 110].

II.5.2 Planification et non-déterminisme

Ces travaux, récents, constituent un prolongement des choix que nous avons effectués au niveau de l'architecture générique que nous avons proposée (§II.2). On se propose de prendre en compte, au niveau du planificateur, des modèles d'actions non-déterministes (certaines actions de perception notamment) ainsi qu'une description de l'évolution possible de l'environnement à travers l'occurrence d'événements "externes".

L'objectif visé, au-delà l'élargissement des classes de problèmes traités par le planificateur, consiste à doter le planificateur de la capacité d'identifier et de traiter des situations d'échec ou de reprise dites "indésirables" ou "dangereuses" et pour lesquelles il est nécessaire de planifier une réaction *à l'avance*, réaction qui viendrait éventuellement supplanter une réaction par défaut du superviseur [47]. On peut dire qu'un tel système est "robuste" si, pour toute situation nécessitant une réaction rapide, le superviseur dispose de l'information suffisante pour déclencher cette réaction, et surtout pour prendre la "bonne" décision.

Une modélisation et un algorithme de planification appelé NODEP ont été développés [111] avec des résultats prometteurs. NODEP apporte une solution élégante et complète: en n'autorisant la replanification que dans des situations stables, c'est-à-dire dans lesquelles le temps mis à planifier n'est pas contraint, il garantit la robustesse du système. A l'opposé, en permettant la replanification dans de tels cas, il économise le développement de tous les futurs possibles, et donc réduit la complexité et le temps de calcul nécessaire. Si le premier état dans

lequel se trouve le système est lui-même stable, alors sa sécurité est garantie. L'accomplissement de la tâche, par contre, ne l'est pas: il est possible qu'il n'y ait pas de plan, ou bien qu'il se révèle impossible de rejoindre le but depuis une situation stable non développée dès le départ en vertu de la stratégie d'économie.

Points marquants:

- Les travaux sur le planificateur IXTET ont donné lieu à des développements et des applications à la fois en génération et en contrôle d'exécution de plan.
  Citons notamment l'application de ces travaux dans le cadre du projet VAP [77, 79, 83], du projet HILARE [28, 35], d'une convention CIFRE avec la société SHELL (sous la responsabilité de Malik Ghallab) et du projet PADRE (MESR) avec Matra Marconi Space.
  Cette approche constitue également un élément important de notre contribution à un projet collaboratif (LAAS, IRIT, CERT) sur la Planification (cf §B.2) dans le cadre du PRC Intelligence Artificielle et dont j'ai assuré la coordination pendant 2 ans [62].
- Les travaux portant sur NODEP ouvrent la voie à une étude formelle et des algorithmes de planification qui prennent en compte explicitement le caractère évolutif de l'environnement du robot et les capacités dé de décision et réaction disponibles au niveau superviseur.

II.6 La planification automatique de tâches de type Prendre-et-Poser

J'ai dirigé les travaux de développement d'un système - appelé SPARA - de planification automatique de tâche de Prise et Pose pour un robot manipulateur à 6 degrés de liberté (structure bras-poignet) équipé d'une pince à mors parallèles et manipulant des objets polyédriques.

Ce système est similaire, dans ses principes, à HANDEY et SHARP développés respectivement au MIT et au LIFIA. En effet, il est fondé sur une décomposition d'une tâche de prise et pose en une séquence d'étapes "élémentaires" correspondant à des problèmes distincts: grand mouvement, choix de la prise, mouvement d'approche, mouvement de dégagement, etc...

SPARA intègre un ensemble de planificateurs spécialisés traitant de chacun de ces sous-problèmes [21, 25] De plus, il manipule explicitement et propage, en cours de planification, les incertitudes de positionnement relatif des objets, permettant ainsi de produire un programme robuste.

La planification des mouvements d'approche et de dégagement est également traitée de manière originale: en effet, elle permet de produire une trajectoire de la pince prenant en compte une représentation exacte des obstacles locaux.

Une troisième caractéristique essentielle de SPARA est le développement d'une structure de contrôle flexible permettant de prendre en compte explicitement les interactions fortes entre les différentes étapes d'une tâche de Prise et Pose: ainsi, la planification des trajectoires d'approche et de dégagement prend en compte explicitement les contraintes cinématiques induites par le bras porteur; de même il est possible de prendre en compte, au moment de la planification d'une étape de saisie, les contraintes (incertitudes, obstacles locaux...) imposées par une étape de pose ultérieure.

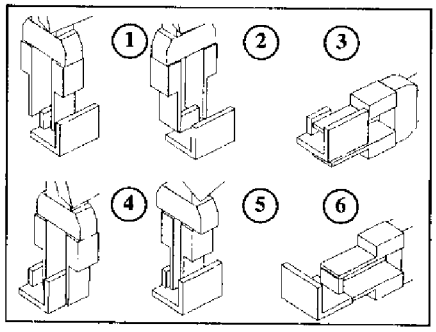


FIG. 6 – *Calcul des positions de prise pour une pince à mors parallèles*

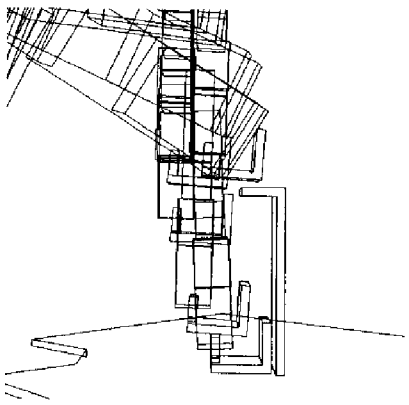


FIG. 7 – *Calcul d'une trajectoire de dégagement*

Le système SPARA de planification automatique de tâches de Prise et Pose a été effectivement réalisé et a permis de produire automatiquement des plans de niveau effecteur exécutés par un robot SCEMI. Il représente le travail de trois doctorants [106, 107, 108] et de plusieurs stagiaires.

Les figures 6, 7 et 8 illustrent une exemple de tâche de Prise et Pose produit par SPARA.

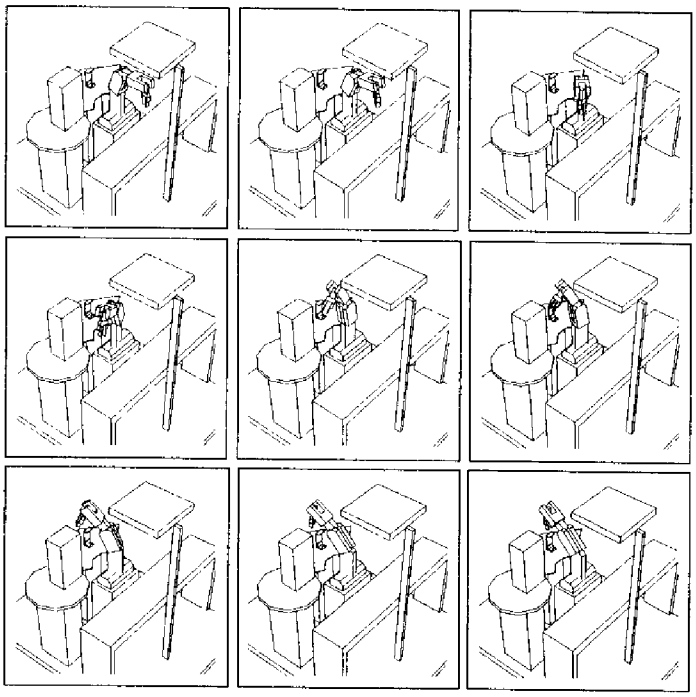


FIG. 8 – *Calcul d'une trajectoire de transfert*

Points marquants:
Notons qu'il s'agit, à notre connaissance, d'un des très rares systèmes de ce type qui aient été effectivement et entièrement implantés et testés dans un environnement réel.
Il a également fait l'objet d'une installation à l'Université de Berkeley pour la planification de tâches d'assemblage (cf. B.1).
Notons enfin que le caractère fortement imbriqué des différentes étapes d'une tâche de Prise-et-Pose ainsi que sa dépendances très forte vis-à-vis de l'encombrement de l'environnement m'a conduit à re-formuler le problème d'une manière plus générale (Cf. §II.7).

II.7 Une formulation générale de la planification des tâches de manipulation

Une tâche de manipulation se caractérise par la capacité qu'a le robot de déplacer certains objets dans son environnement et ainsi d'en modifier la structure. Ainsi un problème de manipulation ne se réduit pas à un simple problème d'évitement d'obstacles.

Alors que la plupart des méthodes attaquant ce problème présupposent une décomposition *a priori* de la tâche en sous-tâches, nous nous sommes posés le problème de l'automatisation de l'étape décomposition elle-même. Il apparaît que le problème dans sa globalité trouve une modélisation formelle purement géométrique. Il s'agit de considérer le produit cartésien de l'espace de configurations de *tous* les corps mobiles de la scène, c'est-à-dire non seulement le robot, mais aussi les objets potentiellement déplaçables. Une tâche de manipulation apparaît ainsi comme un chemin particulier dans cet espace. Mais, de même qu'un robot mobile possède des contraintes qui invalident certains chemins, ici aussi tous les chemins dans cet espace ne correspondent pas nécessairement à un chemin de manipulation. Il suffit pour s'en convaincre de constater que localement le chemin est inclus dans une variété de dimension égale à la dimension de l'espace des configurations du robot (puisque le robot est le seul "agent" de la scène).

J'ai proposé et développé, avec J.P. Laumond et T. Siméon, une formulation générale du problème et un schéma d'algorithme de résolution [60].

D'un intérêt uniquement théorique, cette approche a permis de montrer que le problème de la manipulation restait un problème décidable, mais surtout a permis une bonne compréhension du problème ce qui a eu pour conséquence la résolution de deux instances de ce problème.

Un premier algorithme [71] résout la planification de mouvements d'un disque en présence d'obstacles polygonaux et d'un objet déplaçable circulaire. Outre sa possible application dans le contexte de la robotique mobile, cet algorithme a permis de vérifier l'applicabilité de l'approche générale.

Dans une deuxième application [8] on considère que l'ensemble des prises et des poses possibles des objets est en nombre fini. Dans ce cas l'espace des configurations de toute la scène se réduit à un nombre fini d'espaces de configurations de dimension de l'espace des configurations du robot. Un système général de planification a pu être ainsi construit *sur* un planificateur de trajectoires sans collision. Il produit en sortie, non seulement un plan au sens de la décomposition explicite de la tâche en sous-tâches élémentaires, mais également la séquence des trajectoires sans collision que doit effectuer le robot.

Enfin, ce travail a été poursuivi dans le cadre d'une thèse et appliqué au problème de la planification des mouvements d'un polygone en présence d'obstacles (fixes et déplaçables) polygonaux [36, 10]. La figure 9 illustre un exemple de plan de manipulation produit.

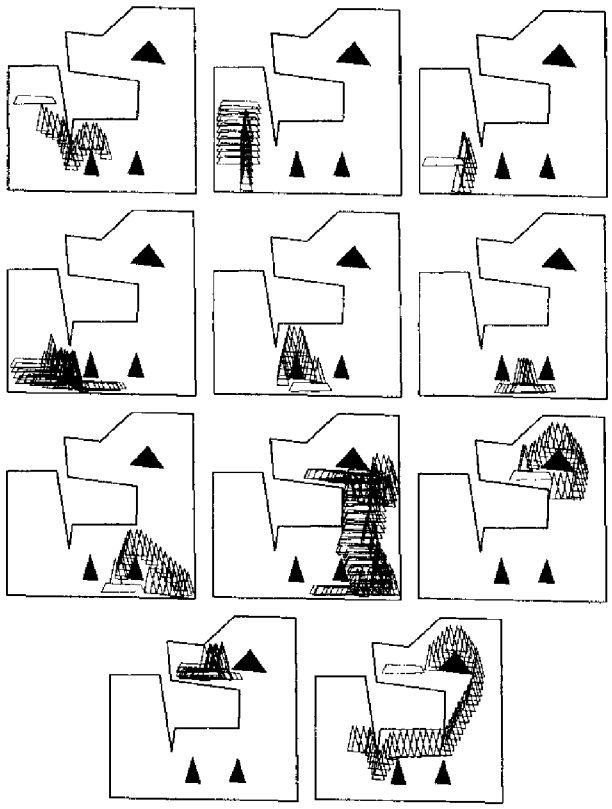


FIG. 9 – *Le planificateur produit les trajectoires ainsi que les positions où il est nécessaire d'effectuer des changement de prise*

Points marquants:
Ce travail est aujourd'hui abondamment cité dans la littérature. Il a jeté les bases d'une nouvelle classe de problèmes de planification géométrique de tâches robotiques. Il est l'objet de plusieurs extensions.
Notons également qu'il est longuement présenté dans un chapitre du livre de J-C. Latombe (professeur à l'Université de Stanford) intitulé "Motion Planning" et qui constitue aujourd'hui une référence dans le domaine.

II.8 Planification de stratégies de déplacement pour un robot mobile en présence d'incertitudes

Une des difficultés majeures posées par l'utilisation d'algorithmes de planification de trajectoires sur des robots réels est liée la non prise en compte par les planificateurs des écarts possibles entre les modèles sur lesquels s'appliquent le raisonnement et la réalité. En effet, il est nécessaire, dans des environnements contraints, de planifier en prenant en compte les incertitudes de manière explicite (incertitudes provenant du modèle géométrique de l'environnement tel qu'il est produit par la perception, mais également des erreurs de contrôle).

J'ai développé, avec Thierry Siméon, un planificateur qui, étant donné une incertitude sur la position initiale du robot, ainsi qu'un modèle de l'erreur de contrôle, permet de générer une stratégie de déplacement (composée de mouvements gardés et de mouvements asservis sur des capteurs externes) garantissant au robot d'atteindre son but avec une précision donnée [63, 64].

Les figures 10 à 12 illustrent les stratégies produites dans différentes conditions.

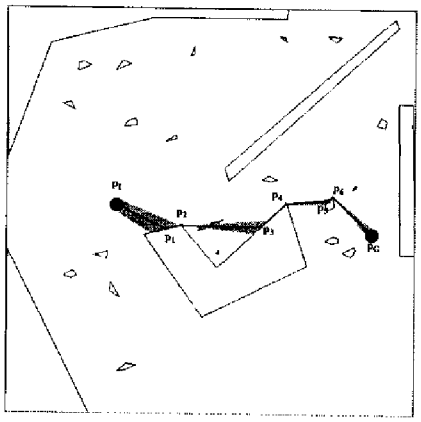


FIG. 10 – *Incertitude faible sur la position de départ et sur le contrôle des déplacements*

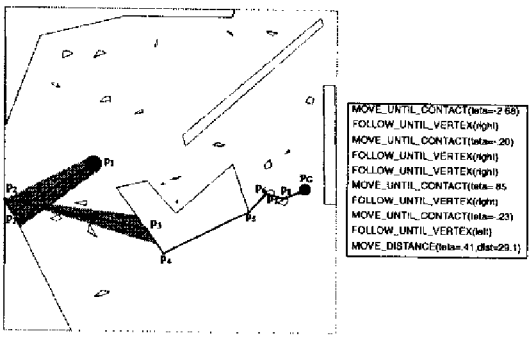


FIG. 11 – *Stratégie différente si l'incertitude sur la position initiale est plus grande*

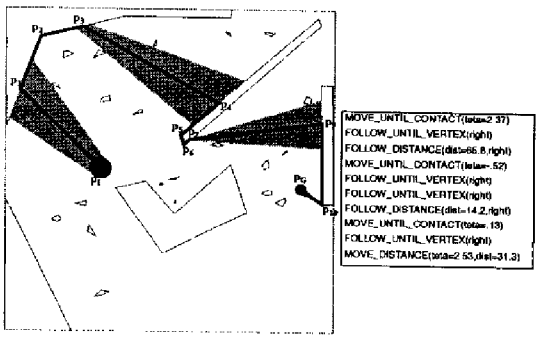


FIG. 12 – *Stratégie différente si l'incertitude sur le contrôle des déplacement est plus grande*

Cette problématique - la génération automatique de stratégies de déplacement avec prise en compte explicite des incertitudes - est aujourd'hui affichée comme une thématique importante du groupe RIA: deux thèses en cours [48]

Points marquants:
Les travaux de recherche sur la prise en compte des incertitudes au niveau de la planification de trajectoire, connaissent aujourd'hui un regain d'intérêt notable.
Notre approche se distingue par le fait qu'elle a été parmi les premières à prendre compte le cas des robots mobiles avec accumulation progressive de l'erreur de position (odométrie).

II.9 Un robot mobile autonome

J'ai dirigé l'implantation complète (1990-1992) sur les moyens expérimentaux du LAAS de l'architecture de Téléprogrammation niveau tâche de robots d'intervention (décrite en II.4) ainsi que la démonstration de plusieurs expérimentations inspirées des scénarios de mission VAP-RISP et AMR-EUREKA.

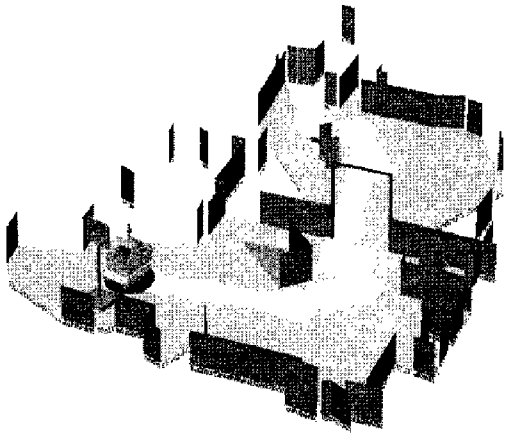


FIG. 13 – *Hilare 1.5 a atteint sa cible*

Un système robotique complet a ainsi été réalisé et mis en œuvre, intégrant un planificateur de mission de haut niveau avec prise en compte de contraintes temporelles symboliques et numériques [6, 28], un système d'affinement de tâches en fonction du contexte d'exécution ainsi qu'une structure de contrôle assurant l'exécution des tâches et la réaction aux événements asynchrones dans des délais compatibles avec la dynamique de l'environnement [30, 31, 35].

Les démonstrations illustrent la capacité du système de planifier et de réaliser des missions en environnement inconnu avec construction incrémentale de l'environnement, identification d'amers, gestion de contraintes temporelles, détection de situations non planifiées et reprises différents niveaux d'abstraction.

La figure 14 représente une trace graphique de la réalisation d'une tâche de navigation autonome dans un environnement d'intérieur inconnu avec la construction incrémentale d'un modèle de l'espace libre.

La figure 13 illustre la situation finale atteinte par le robot ainsi que la connaissance de l'environnement qu'il a acquise en réalisant sa mission.

Points marquants:
Cette expérimentation a constitué la première implémentation complète de l'architecture et des processus décisionnels et fonctionnels permettant l'expression à un très haut niveau d'abstraction d'une mission qui est ensuite réalisée de manière autonome par le robot.
Notons qu'une telle intégration et sa démonstration effective est rarissime.

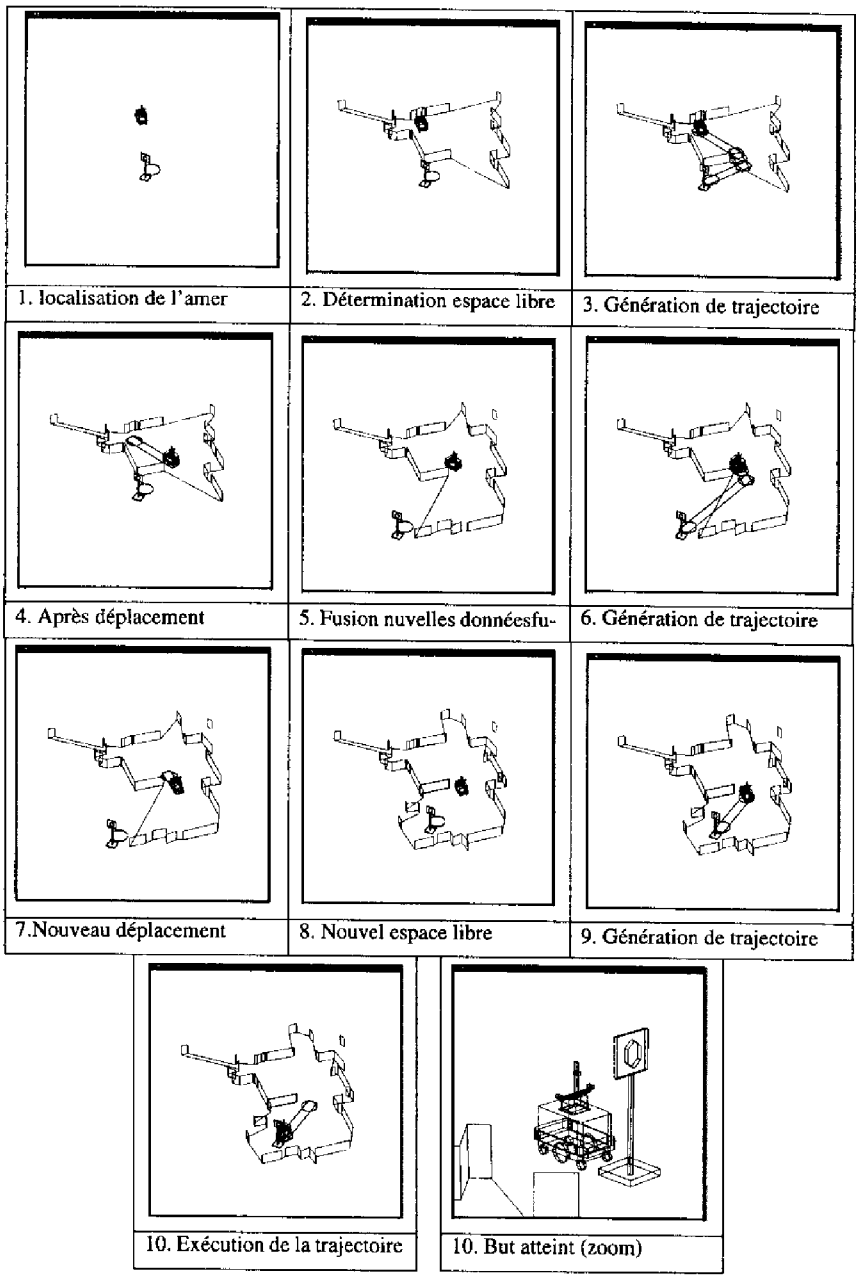


FIG. 14 – *Navigation autonome en environnement d'intérieur inconnu*

II.10 Une approche nouvelle de la coopération multi-robots

II.10.1 Le Paradigme d'insertion incrémentale de plans

Cette problématique nouvelle a été démarrée en accompagnement du projet ESPRIT-III MARTHA (cf. §A.4).

Nous partons de l'hypothèse qu'il est possible (et plus efficace) dans un grand nombre de cas d'éviter une planification unique dans laquelle une station centrale (ou un robot) planifie dans le détail les actions de l'ensemble des robots[8].

Nous supposons que les robots disposent de moyens de communication (avec une portée éventuellement limitée) leur permettant d'interroger les robots dans leur voisinage et d'échanger des informations (buts, plans courants, plans futurs, signaux, état).

J'ai proposé un schéma original de coopération multi-robots fondé sur un *Paradigme d'insertion de Plan* [46].

Dans une première étape, un robot élabore un plan puis tente de l'insérer dans un ensemble de plans en cours d'exécution. Ceci est rendu possible grâce au système de communication qui permet au robot d'interroger les robots voisins et de récupérer leurs plans en cours d'exécution et les synchronisations (éventuelles) déjà établies lors d'une phase d'insertion de plan antérieure.

L'avantage d'une telle approche tient essentiellement:

- au fait que l'exécution peut se dérouler en parallèle (un robot évitant autant que possible de perturber les plans courants);
- au fait que le paradigme est incrémental par construction: il permet de prendre en compte aisément l'insertion ou l'extraction d'un robot dans une coopération en cours.
- au fait que le paradigme se prête à un traitement hiérarchique des conflits
- par rapport à de nombreux schémas proposés dans la littérature, il est correct, n'induit aucun comportement incohérent et permet de détecter toutes les situations d'interblocages nécessitant une planification centralisée.

Deux réalisations de ce paradigme général ont été développées. Elles diffèrent par la nature des plans échangés: plans d'actions avec allocation de ressources [49]ou trajectoires [66]. Elles ont permis de traiter de manière systématique le problème de la navigation d'un ensemble de robots dans un réseau "routier"[50]: voies, carrefours, zones de chargement ou de garage.

II.10.2 Coopération entre plusieurs robots mobiles autonomes

J'assure depuis septembre 1993 l'animation d'un projet interne "multi-robots" et la responsabilité de la réalisation d'une expérimentation multi-robots mettant en œuvre plusieurs robots mobiles (Hilare 1.5, Hilare 2, Hilare 2-bis).

8. Celle-ci peut toutefois s'avérer nécessaire dans certaines situations qu'il s'agira de détecter

Je coordonne, dans ce cadre, les travaux de plusieurs chercheurs (F. Ingrand, T. Siméon, R. Chatila), ingénieurs (M. Devy, M. Herrb), post-doc (S. Suzuki de l'université de Tsukuba, B. Dacre-Wright), doctorants (S. Fleury, F. Robert, L. Aguilar, M. Khatib, H. Bullata)

Chacun des robots est doté d'une implantation nouvelle de l'architecture de contrôle telle qu'elle est décrite en §II.2 et entièrement installée à bord du robot sous VxWorks.

L'expérimentation est aujourd'hui opérationnelle. Elle intègre de nombreux travaux développés par le groupe RIA et démontre concrètement l'approche de la coopération multi-robots que j'ai développé. Cette expérimentation a été notamment retenue comme démonstrative du concept MARTHA §A.4.

Un opérateur ou une station centrale alloue à chaque robot autonome une mission propre exprimée à un haut niveau d'abstraction (ex: transport d'une charge d'un lieu à un autre).

Chaque robot dispose de ses propres moyens de perception. De plus, les robots sont équipés d'un système de communication leur permettant d'une part de communiquer avec la station et d'autre part de communiquer directement avec les robots qui se trouvent dans leur voisinage (communication inter-robots avec portée limitée).

Les robots doivent, de manière autonome, planifier leurs actions et les coordonner et les exécuter, en prenant en compte le contexte d'exécution tel qu'ils le perçoivent au moyen de leurs capteurs et de leur système de communication.

Dans un premier temps, pour valider l'approche de la coopération que j'ai développé. nous avons réalisé un système de simulation qui permet d'exécuter la presque totalité des logiciels sur stations de travail (une station par robot + une station pour la visualisation graphique). Ce système a permis de tester des scenarii réalistes (inspirés de l'application MARTHA) impliquant une quinzaine de robots. Les résultats sont probants et se situent à un très bon niveau international [50].

Le système est également entièrement opérationnel sur trois robots de la famille Hilare (Hilare1.5, Hilare 2 et Hilare 2-bis) dotés de capacités de perception (pour la localisation et la construction de modèles d'obstacles), d'évitement d'obstacles imprévus, de planification de missions et de trajectoires... Le paradigme d'insertion de plan a ainsi pu être démontré non seulement dans des cas "nominaux" mais également dans des situations de reprise après échecs.

Points marquants:
Cette approche s'est avérée à la fois générique et efficace. Elle a été démontrée en simulation sur un nombre substantiel de robots (15) pour des missions de navigations relativement contraintes, tout en garantissant une cohérence du comportement global. En ce sens, elle est aujourd'hui, à notre connaissance, unique. Les autres systèmes connus mettant en œuvre autant de robots, ont été réalisés sur la base de comportements dits "réactifs"et ne peuvent donc prétendre à un comportement global cohérent.
Cette approche s'avère par ailleurs riche d'extensions et de variantes.

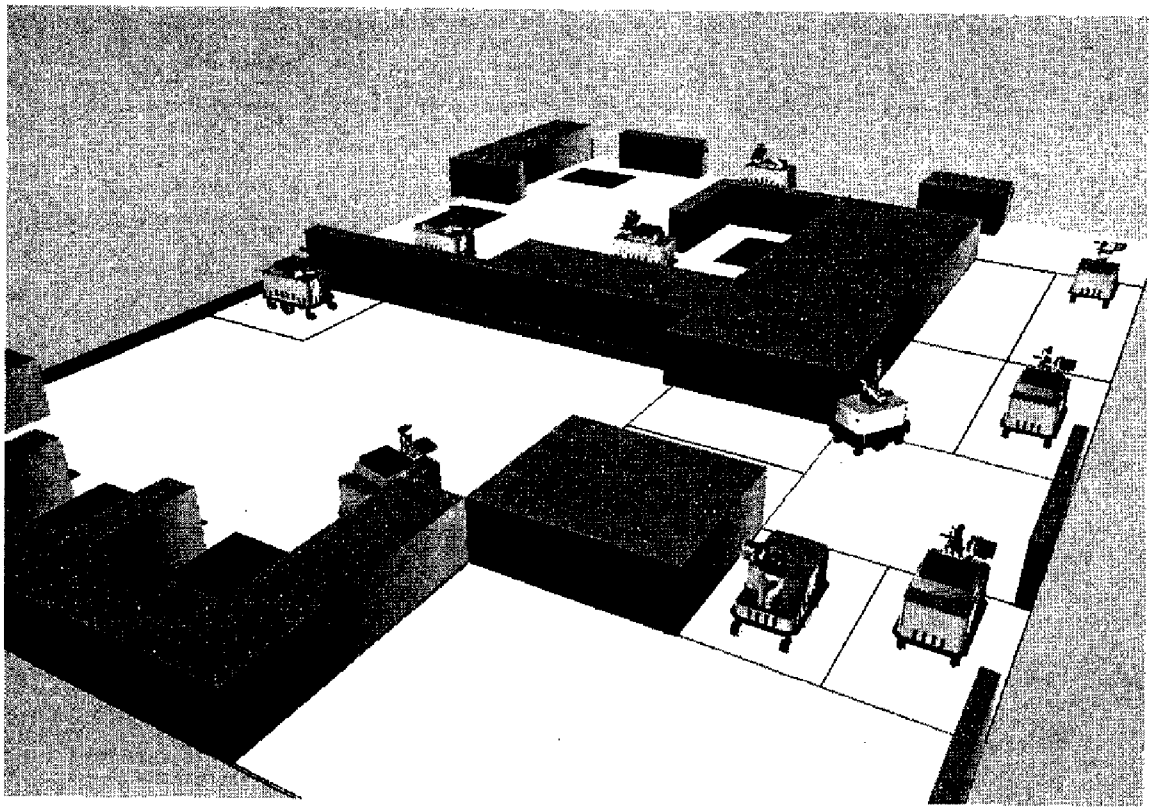


FIG. 15 – *L'une des simulations réalisées pour valider le "Paradigme d'Insertion de Plans": 10 robots mobiles autonomes dans un environnement d'intérieur*

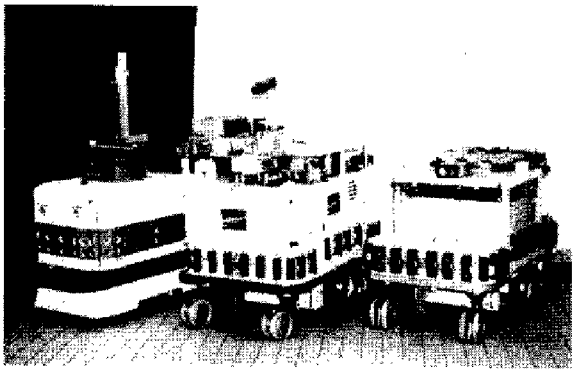


FIG. 16 – *3 robots mobiles autonomes et coopérants: Hilare 1.5, Hilare 2 et Hilare 2-bis*

III PROSPECTIVE

Mon ambition reste toujours d'aller de l'avant dans la conception et la concrétisation des ingrédients décisionnels de l'autonomie en robotique.

L'architecture générique que j'ai élaboré offre un cadre adéquat pour les travaux de recherche et les réalisations en cours.

Elle est notamment intéressante également parce qu'elle fournit une aide pour caractériser les types de plans et, par conséquent, les "planificateurs" et les "superviseurs" d'actions et de tâches. Ainsi, il s'avère aujourd'hui indispensable de concevoir des planificateurs qui prennent en compte explicitement le non-déterminisme (lié aux conséquences des actions du robot et aux résultats des actions de perception) et les incertitudes sur l'état (actuel ou futur) du robot et de son environnement.

Ce travail a déjà été entamé avec des premiers résultats [44, 47]. Je compte le poursuivre et le renforcer.

Un deuxième axe, aujourd'hui important de ma prospective, concerne la coopération multi-robots.

Je compte poursuivre et étendre ma recherche dans ce domaine: approfondissement du schéma de coopération que j'ai développé, et exploration d'autres schémas prenant en compte notamment des interactions entre des robots qui diffèrent par leurs capacités d'action et de perception.

Une autre défi est celui du développement de nouvelles recherches et de l'application des résultats obtenus dans des domaines où l'autonomie décisionnelle et opérationnelle du robot est incontournable. C'est le cas notamment:

- **du robot mobile d'exploration planétaire:**

  l'éloignement du robot, les contraintes sur les communications entre la Terre et Mars (retards de 5 à 20 minutes, débits de l'ordre du Kilo baud), interdisent le recours à la téléopération (à moins d'accepter une évolution du robot d'une lenteur extrême ou des risques considérables lors du fonctionnement de la machine).

- **de la flotte de robots mobiles autonomes:**

  la complexité du problème, la quantité d'informations à transmettre, le flux d'événements, sont tels qu'une station centrale, quelle que soit la puissance de ses calculateurs, ne peut piloter l'ensemble à bas niveau (à moins d'accroitre jusqu'à l'absurde les capacités de transmission et de traitement de l'information).

Ceci est en pleine cohérence avec l'un des axes prioritaires mis en avant par le Département SPI[9]: "les structures et machines intelligentes".

A court terme, de manière prévisible et en prolongement de la situation actuelle, ces travaux

9. Département "Sciences pour l'ingénieur" du CNRS.

s'accompagneront:

- de responsabilités importantes, de tâches d'animation et de collaboration dans le cadre de projets internationaux;
- d'une activité intense de réalisation, d'intégration et de transfert des résultats.

Annexes

A RESPONSABILITES DANS DES GRANDS PROJETS

Je ne rapporte, ici, que sur les grands projets nationaux ou internationaux, dans lesquels j'ai contribué de manière importante et j'ai assumé des responsabilités parfois très exigeantes.

Ces projets, constituent, sur des registres différents, des points marquants dans ma carrière.

A.1 ARA (1980-1986)

Pendant la période 1984-1986, j'ai été chargé de coordonner les travaux de plusieurs chercheurs du groupe RIA et d'ingénieurs de recherche du LAAS visant à l'installation et à la mise en œuvre du site expérimental collectif du Pôle Robotique Générale du programme ARA [56, 57].

Les travaux ont porté aussi bien sur l'installation progressive d'un ensemble expérimental complexe que sur le développement et l'intégration d'un grand nombre de logiciels provenant des équipes de recherche impliquées dans le pôle Robotique Générale d'ARA.

Ceci nous a notamment permis de réaliser plusieurs expérimentations "canoniques" [57] mettant en œuvre des sous-séquences d'assemblage d'une pièce industrielle complexe avec variantes (contacteur électrique fourni par la société Télémécanique).

A.2 EUREKA-FAMOS (1986-1987)

J'ai participé, en tant qu'adjoint de Georges GIRALT (Responsable Technique National), au projet européen EUREKA-FAMOS réunissant, sur la base d'une initiative Franco-Allemande sept pays (Allemagne, Autriche, Espagne, France, Italie, Royaume Uni, Suède)

Il s'agissait d'un projet cadre ou "parapluie", le premier de ce type dans EUREKA. L'objectif essentiel était de susciter des projets EUREKA visant à développer des sites pilotes productifs intégrés flexibles pour l'assemblage industriel dans une perspective de coopération européenne.

Ce travail a représenté une part importante de mon activité pendant la période octobre 1986 - septembre 1987 correspondant à la phase préliminaire du projet FAMOS:

- Participation à plusieurs journées de travail à l'échelle européenne: Londres (novembre 86), Milan (janvier 87), Madrid (mars 87), Toulouse (mai 87);
- Participation à de nombreuses réunions au niveau national: Comité de Suivi et Pilotage, Groupe de Travail FAMOS-France...
- Analyse technique et suivi de propositions de projets EUREKA présentées par plus d'une dizaine d'industriels français;
- Participation à la recherche de partenaires européens pour plusieurs propositions de projets FAMOS;
- Participation à la rédaction d'un "Etat de l'art" des technologies de l'assemblage flexible.

Cette première phase de FAMOS a été un succès complet et exemplaire.

D'une part, neuf projets (dont trois projets d'origine française) issus de FAMOS, ont obtenu le label EUREKA lors de la conférence Ministérielle EUREKA de septembre 1987 (Madrid). D'autre part, il a solidement installé la structure internationale de promotion et de suivi.

En 1992, FAMOS réunissait 20 pays et "abritait" plus de quarante projets.

A.3 VAP/RISP (1989-1993) puis IARES/EUREKA

J'ai assumé des responsabilités au sein du groupement RISP (Robotique d'Intervention sur Site Planétaire – CEA, CNRS, INRIA, ONERA, contrats de partenariat avec le CNES) depuis octobre 1989.

Ce groupement a mené un travail important d'étude d'un véhicule d'exploration planétaire (avant-projet VAP [10]) sur MARS.

J'ai été responsable du thème 10 ("Planification de Mission et Téléprogrammation niveau Tâche") et co-responsable du Thème 6 ("Gestion des Tâches et Structure Décisionnelle") d'octobre 1989 à janvier 1991 [78, 77, 80, 79, 84, 83], puis responsable de l'axe fonctionnel "Structure Décisionnelle et Téléprogrammation Niveau Tâche" depuis janvier 1991. Cet axe fonctionnel couvre les aspects décisionnels et programmation niveau tâche aussi bien au niveau de la station d'opération (au sol) qu'au niveau de la structure de contrôle embarquée.

Il est à remarquer que le projet VAP-RISP a pris une nouvelle dimension par la mise en place en 1993 d'un projet EUREKA (IARES) portant sur la réalisation d'un démonstrateur de robot mobile d'exploration planétaire. Outre la participation de RISP, du CNES et d'industriels français (MATRA-MACORNI SPACE, ALCATEL ESPACE, SAGEM, CYBERNETIX, ITMI, ROL), le projet concerne la Russie, la Hongrie et l'Espagne.

J'ai participé activement à la mise en place de ce projet et j'assure actuellement la responsabilité, pour le LAAS, du "Segment Sol Robotique" (Planification et Téléprogrammation au sol) en liaison avec CYBERNETIX.

A.4 MARTHA (ESPRIT III) depuis 1992

Le projet MARTHA (ESPRIT III, Nr 6668) a démarré le 15 juin 1992. Les partenaires sont: la France (FRAMATOME qui en assure la coordination, la SNCF, la société ROL et le GIP PROMIP [11] par ses composantes LAAS/CNRS et Midirobots), l'Allemagne (Mannesmann Demag Gottwald, Indumat, l'Université de Karlsruhe et l'Aéroport de Francfort), les Pays Bas (le Port de Rotterdam) et l'Espagne (le centre de Recherche et Développement IKERLAN).

L'objectif général du projet (sur 3 ans) est l'étude et le développement des concepts et des systèmes nécessaires à la planification, au contrôle et à l'exécution de tâches de transbordement et transport de conteneurs dans des gares, ports et aéroports par une flottille de robots autonomes dotés de capacités de perception et de décision avancées.

10. Véhicule Automatique Planétaire
11. PROMIP: GIP Productique Midi-Pyrénées

Les applications de ce projet sont de première importance. La présence d'utilisateurs aussi importants que la SNCF, l'Aéroport de Francfort et le Port de Rotterdam en témoigne.

Notons que le Port de Rotterdam (Europe Combined Terminals) dispose depuis quelques mois d'une installation opérationnelle équipée notamment d'une cinquantaine de robots mobiles assurant le transport de conteneurs entre des grues de quai et des stations de stockage elles-mêmes automatisées.

Le projet MARTHA a pour ambition est d'aller vers une plus grande flexibilité en dotant les robots d'une grande autonomie décisionnelle et opérationnelle.

En effet, les solutions actuelles souffrent d'un certain nombre de limitations telles que la nécessité, comme c'est le cas aujourd'hui à Rotterdam d'installer un balisage systématique (coûteux et difficilement reconfigurable) du site de travail des robots.

Par ailleurs, la planification et le contrôle de l'ensemble du système est exclusivement du ressort du système informatique central. Ce dernier fournit des ordres extrêmement détaillés aux robots sous la forme de trajectoires que les robots doivent réaliser de manière exacte et sans être capables de les modifier en cas de détection d'obstacles par exemple. De plus, c'est au niveau de la station centrale que sont gérés l'ensemble des conflits éventuels entre robots.

Ce sont sur ces points que le projet MARTHA porte son effort. En effet, l'approche retenue dans le cadre de MARTHA ne nécessite ni de balisage préalable du site d'évolution des robots, ni des communications très fréquentes entre la station centrale et les robots.

La station centrale, envisagée dans MARTHA, produit et met à jour une mission globale (de haut niveau) pour les robots avec des contraintes temporelles et laisse une grande autonomie aux robots quant à la planification détaillée et l'exécution de leurs tâches en fonction du contexte d'exécution (perçu de manière autonome) et à la gestion des interactions avec les autres robots.

A titre d'exemple, chaque robot reçoit de la station centrale une mission du type:

"Va-à Station-22, Charge conteneur-17 avant 12h45, Va-à Grue-34, Décharge"

Il doit être capable grâce à ses capacités propres de perception et de décision embarquées de mener à bien une telle en mission de façon autonome. Ceci suppose notamment des capacités

- A) de perception de l'environnement en vue:
  - 1) de la détection d'obstacles fixes ou mobiles,
  - 2) de l'élaboration d'un modèle de l'environnement du robot,
  - 3) de la localisation du robot dans son environnement (à partir d'amers non spécifiques tels que les bâtiments)
- B) de communication avec la station centrale mais également de communication directe entre les robots
- C) enfin, des capacités de raisonnement:
  - pour la planification et le contrôle de l'exécution de mission,

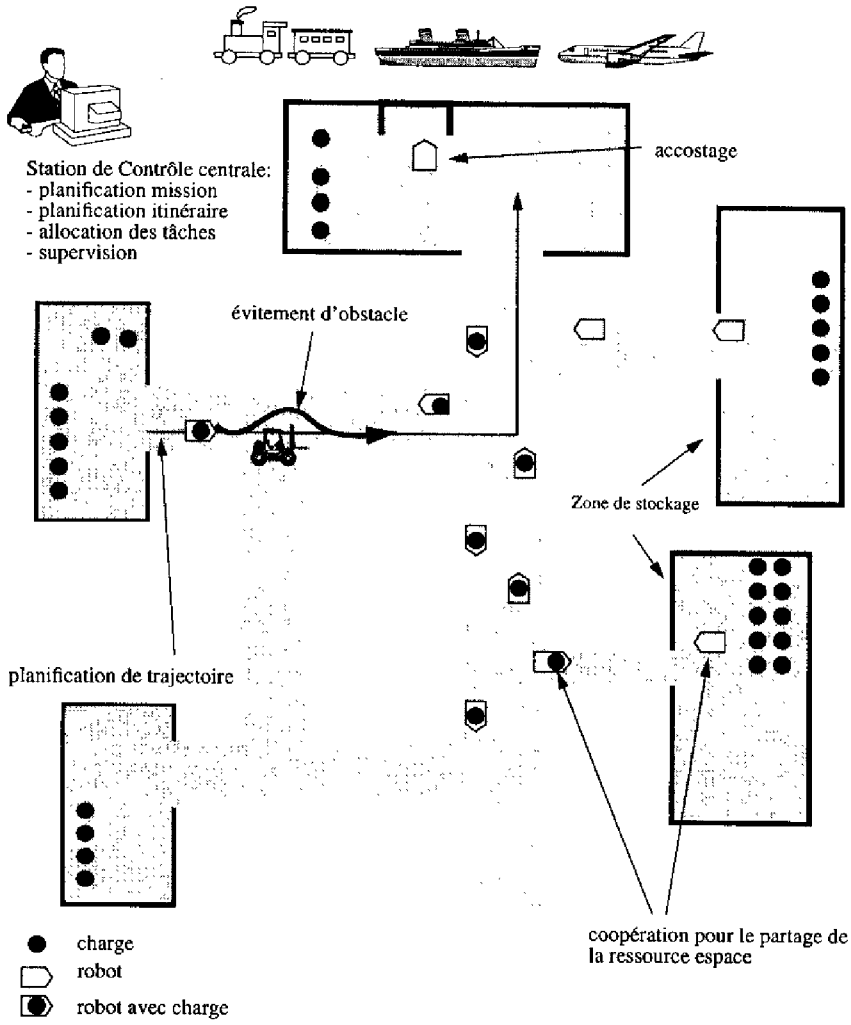


FIG. 17 – *Représentation de la problématique du projet MARTHA*

- pour la coordination avec les autres robots,
- et pour la génération de routes et de trajectoires,

dans un environnement dynamique comprenant non seulement un grand nombre de robots (de l'ordre de la cinquantaine) mais également d'autres véhicules conduits par des hommes et même des piétons, tout ceci dans des conditions climatiques sévères (fonctionnement 24h/24h, pluie, brouillard...).

L'ensemble des résultats obtenus dans le cadre de MARTHA seront démontrés sur deux sites réunissant les conditions des applications réelles: à Trappes (SNCF) et à l'Aéroport de Francfort.

Le rôle de PROMIP (et plus précisément le LAAS et MIDIROBOTS) dans ce projet est la

réalisation d'une architecture logicielle embarquée générique, le développement de l'ensemble des aspects décisionnels incluant ceux permettant d'assurer une coordination autonome entre les robots sans intervention de la station centrale, et la planification de trajectoires [89, 90, 91, 94]

J'assure, depuis le 1er octobre 1992, la responsabilité (Local Project Manager) de la contribution de PROMIP (LAAS/MIDIROBOTS) à ce projet ainsi que la responsabilité (Workpackage Leader) d'un aspect essentiel du projet: l'ensemble des logiciels embarqués.

Ceci se traduit par une charge de travail très importante. En effet:

- la contribution de PROMIP s'élève à 108 hommes-mois sur 3 ans
- il s'agit également d'assurer la coordination entre plusieurs partenaires (FZI, ROL, FRAMATOME, PROMIP, IKERLAN) jusqu'à l'intégration des logiciels
- il s'agit aussi de coordonner le travail de développement mené au LAAS et à MIDI-ROBOTS avec les contraintes de réalisation de logiciels importants devant être portés sur des robots développés par d'autres partenaires: robot COMMUTOR de la SNCF (fig 22, robot développé par INDUMAT pour l'aéroport de Francfort.

En octobre 1994 puis en juin 1995, deux revues détaillées du projet par les experts de la CEE ont été tenues au LAAS. La qualité des travaux de PROMIP a été soulignée.

La figure 18 représente un environnement d'évolution typique.

Les figures 19, 20 et 21 représentent des traces d'exécution (en simulation) d'une résolution de conflit détectée et résolue de manière décentralisée.

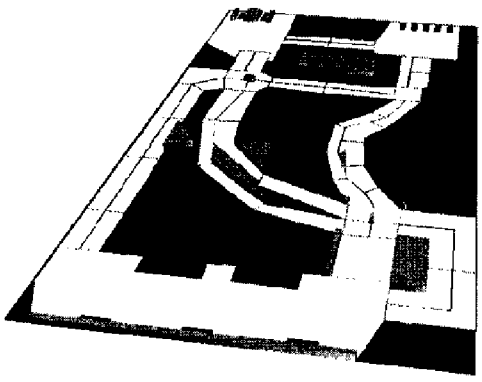


FIG. 18 – *Un environnement de type "réseau routier" (550 m x 250 m)*

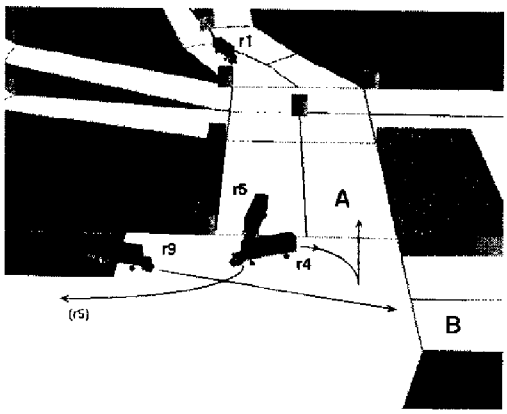


FIG. 19 – *Etape 1*

La dernière phase des travaux a concerné l'intégration (au LAAS) des logiciels de tous les partenaires en vue du portage sur les sites de démonstration.

Ainsi les robots COMMUTOR et FAG sont aujourd'hui dotés d'une architecture de contrôle complète - développée au LAAS sous ma responsabilité - permettant l'affinement et l'exécu-

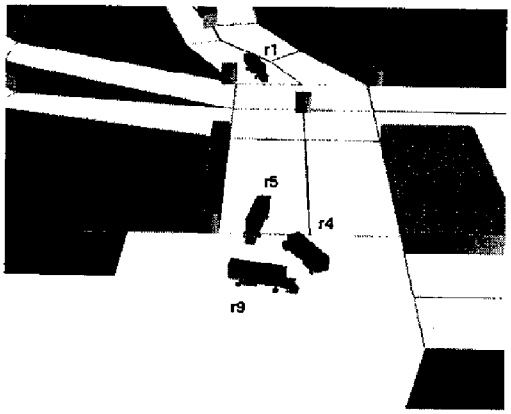


FIG. 20 – *Etape 2*

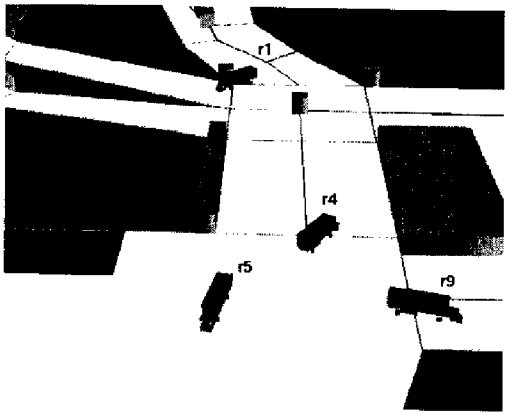


FIG. 21 – *Etape 3*

tion autonome de missions de transport de conteneurs et intégrant des capacités évoluées de coopération inter-robots.

Les démonstrations finales sont prévues pour mars 1995.

FIG. 22 – *Le robot COMMUTOR de la SNCF sur lequel nous avons intégré une architecture de contrôle comprenant un superviseur, un planificateur de trajectoires, un module de surveillance... Une intégration similaire a également été réalisée sur un robot de l'aéroport de Francfort*

B CONTRATS ET CONVENTIONS DE RECHERCHE

B.1 Responsabilités dans des Collaborations scientifiques internationales

Convention avec l'Université de Berkeley (à partir de 1987)

Cette convention porte sur les "Méthodes et les Outils pour la Reconfiguration d'Architectures Informatiques Distribuées et la Programmation Distribuée des Cellules Robotisées Flexibles". J'ai participé à sa mise en place en collaboration avec le groupe OLC du LAAS. Elle a été approuvée et soutenue par la NSF.

Deux doctorants dont j'ai encadré le travail ont effectué des post-doc à Berkeley: Hélène Chochon et Isabelle Mazon. Des logiciels dans le cadre du projet NNS ont été transférés.

Collaboration avec l'université de Pennsylvanie (à partir de 1988)

J'ai élaboré avec R. Chatila un projet de collaboration avec l'université de Pennsylvanie à Philadelphie (Professeur Richard Paul) portant sur la conception et la réalisation d'un système robotisé (plate-forme mobile équipée de deux bras manipulateurs) à grande autonomie destiné à exécuter des tâches de manipulation en milieu sous-marin. Cette proposition a reçu le soutien de la NSF et du CNRS.

Elle a donné lieu notamment à de nombreux échanges: visites, logiciels. Un doctorant dont j'ai encadré le travail a effectué un post-doc à Philadelphie: Thierry Siméon.

Projet ESPRIT-BRA ROCOMI (à partir de janvier 1993)

Il s'agit d'un projet de collaboration (sur deux ans) dans le cadre de l'action Basic Research d'ESPRIT entre trois laboratoires d'Amérique du Sud:

- l'Institut d'Automatique de l'Université de San Juan en Argentine
- le Laboratoire d'Instrumentation et de Contrôle de l'Université de Mar del Plata en Argentine
- l'Institut d'Ingénierie de Caracas au Vénézuela

et deux laboratoires européens:

- l'Institut d'Automatique Industrielle du Conseil Supérieur de la Recherche Scientifique (Madrid, Espagne)
- et le LAAS du CNRS.

J'en assure la responsabilité pour le LAAS. J'ai notamment co-organisé deux workshops (Lisbonne 1993, Madrid 1994).

Convention de Collaboration avec le Québec (à partir de janvier 1993)

Il s'agit d'un Projet de collaboration dans le cadre de la Coopération Franco-Québécoise dans le domaine de l'Enseignement Supérieur et de la Recherche. Il porte sur la Télérobotique Mobile: interface opérateur, perception, modélisation de l'environnement et navigation autonome. Les partenaires sont:

- l' Université Laval (Denis Poussart)
- l'Université Polytechnique de Montréal (Richard Hurteau)
- le Centre de Recherche en Informatique de Montréal (Paul Freedman)
- l'Université McGill (Greg Dudek)
- le LAAS (groupe RIA)

Les chercheurs du LAAS directement concernés par ce projet sont: Rachid Alami, Raja Chatila et Jean-Paul Laumond. J'en assure la coordination pour le LAAS.

Ce projet a donné lieu en 1993 et 1994 à plusieurs échanges (séjour à Québec d'un doctorant de RIA, séjour au LAAS d'un doctorant de l'Université Polytechnique de Montréal, visites de chercheurs dans les deux sens).

B.2 Responsabilités dans des contrats et conventions au niveau national

Projet MRES Programmation Avancée en Robotique d'Assemblage (1988-1989)

J'ai participé à l'élaboration puis assuré l'animation du Projet "Programmation Avancée en Robotique d'Assemblage" soutenu par le MRES. Ce projet porte sur le développement d'*un Système de Programmation Avancé en Robotique d'Assemblage*: Modélisation des processus d'assemblage et modélisation fonctionnelle des cellules robotisées, Modules de Raisonnement Géométrique, Construction d'un Environnement de Programmation complet. Il est le résultat d'une concertation de quatre formations CNRS (LRP, LAAS, LAB, LIFIA).

Projet Planification dans le cadre du PRC-IA (1992-1993)

J'ai coordonné un projet collaboratif (groupe "Robotique et Intelligence Artificielle" du LAAS/CNRS, groupe "Intelligence Artificielle" du CERT/ONERA et équipe "Formalisation du Raisonnement" de l'IRIT) sur la Planification dans le cadre du PRC Intelligence Artificielle.

Les travaux portent sur le choix et l'exploration approfondie d'une représentation des actions et du changement permettant notamment la prise en compte du temps, et offrant un compromis satisfaisant vis-à-vis de la complexité des algorithmiques de planification. Cette représentation doit conduire à la construction d'un "Gestionnaire de Connaissances Temporelles" intégrant un système de maintien des contraintes temporelles et un système déductif,

élément central à partir duquel le projet définira un système de planification, de contrôle d'exécution de plan, d'interprétation de situation et de révision des connaissances.

A noter l'organisation d'une journée de travail sur la Planification (février 1993) avec la participation d'une trentaine de chercheurs du PRC-IA.

Convention de collaboration avec l'IFREMER (1989-1993)

Cette convention portait sur la robotique d'intervention en milieu sous-marin (thèse de Bernard Degallaix).

Projet Recherche et Transfert de Technologie

Je suis responsable d'un projet "Recherche et Transfert de Technologie" soutenu par la règion Midi-Pyrénées. Il implique le LAAS et MIDIROBOTS et porte sur la planification et le contrôle d'exécution d'un ensemble de robots autonomes.

Conventions CIFRE

- **avec MATRA-DATAVISION (1987-1990):** Elle portait sur la modélisation et la propagation des incertitudes de positionnement (thèse d'Isabelle MAZON).
- **avec MIDI-ROBOTS (à partir d'octobre 1992):** Elle concerne la structure décisionnelle embarquée pour robots autonomes de transport de charges (thèse de F. Robert).
- **avec CYBERNETIX (à partir de février 1993):** Elle concerne la Téléprogrammation au niveau tâche pour robots d'intervention autonomes et a pour cadre la participation du LAAS et de la société CYBERNETIX au projet IARES (thèse de J. Perret).

B.3 Participation à des contrats et conventions de recherches

Contrat ADI-MATRA-LAAS (1986)

J'ai contribué à l'exécution d'un contrat ADI-MATRA-LAAS portant sur les capteurs et systèmes associés à un robot manipulateur assurant la distribution d'objets à assembler dans une unité d'assemblage flexible [76].

Convention CNET portant sur la machine MAIA (1986-1988)

Ayant développé par le passé un système LISP [69] [13] [14] [15] [70], j'ai participé à l'utilisation en field-test du prototype MAIA. Par ailleurs, j'ai contribué à l'évaluation des capacités spécifiques de la machine MAIA pour le temps réel dans le cadre des applications robotiques du groupe RIA.

Le projet AMR-EUREKA (1988-1991)

Le projet "AMR" (début en Octobre 1986) porte sur le développement de robots mobiles d'intervention pour la sécurité civile. J'ai participé activement, avec Raja Chatila (qui en assurait la responsabilité pour le LAAS) à partir de 1988. Ceci nous a conduit à travailler en étroite collaboration avec MATRA-ESPACE sur les problèmes d'architecture de contrôle de robots, d'affinement et de contrôle d'exécution de tâches d'intervention en milieu pas ou partiellement structuré. Ces travaux ont donné lieu des transferts de connaissance et de logiciels et à la rédaction en commun de communications. La phase 2 du projet, achevée en 92, a notamment permis la démonstration du robot mobile tout-terrain ADAM réalisant une tâche de navigation en terrain accidenté.

Projet ESPRIT-BRA PROMOTION (à partir de 1992-1995)

J'ai participé à ce projet (dont le responsable est Jean Paul Laumond du groupe RIA du LAAS) essentiellement sur les aspects algorithmiques liés à la planification de stratégies de déplacement pour robots mobiles en présence d'incertitudes (cf §II.8).

A ce titre, j'ai présenté un "Tutorial" sur la planification de trajectoires en présence d'incertitudes lors de l'Ecole de Printemps organisée par le projet en Avril 1993.

C ACTIVITES D'ENCADREMENT

Encadrement de thèses

1. Hélène CHOCHON, *Programmation de tâches d'assemblage robotisées: modélisation et processus décisionnels*, novembre 1986.
2. Ernesto LOPEZ-MELLADO, *Le contrôle d'exécution dans les Cellules Flexibles d'Assemblage*, décembre 1986.
3. Thierry SIMEON, *Génération automatique de trajectoires sans collision et planification de tâches de manipulation en robotique*, janvier 1989.
4. Isabelle MASON, *Raisonnement sur les contraintes géométriques et d'incertitudes pour la planification de tâches de manipulation robotisées*, en mai 1990.
5. Pascal VIOLERO, *SPARA : Un Système de programmation automatique de tâche de manipulation robotisée*, novembre 1991.
6. Bernard DEGALLAIX, *Une architecture pour la téléprogrammation au niveau tache d'un robot mobile autonome*, janvier 1993.
7. Hervé LARUELLE (co-encadrement avec M. Ghallab), *Planification Temporelle et exécution de tâches en robotique*, mars 1994.
8. Jérôme PERRET, *Téléprogrammation au niveau tâche pour un robot mobile autonome d'intervention sur site distant*, novembre 1995.

Thèses en cours

9. Frédéric ROBERT (depuis novembre 1992): Coopération multi-robots et structure de contrôle embarquée de robots mobiles autonomes.
10. Luis AGUILAR (depuis octobre 1993): Planification de tâches de navigation pour robots mobiles dans un contexte multi-robots.
11. Samir QUTUB, (depuis octobre 1995): Coopération multi-robots et Planification Distribuée.

J'ai également accueilli deux postdocs étrangers: P. Freedman (1988-1990, Mc Gill, Canada) et S. Suzuki (1992-1994, Tsukuba, Japon).

Direction de stages et de mémoires de fin d'étude

1. 1984: Ernesto LOPEZ-MELLADO, DEA Automatique, UPS
2. 1984: Sylvie RAVENEAU, ENSEEIHT Informatique
3. 1984: A. GLEYZES et T. LARROQUE, ENSAE
4. 1985: Anourada BONDRE, ENSEEIHT Informatique

5. 1985: Olivier GENELLE, Ecole Centrale de Lyon
6. 1985: J.P. GLEYZES et M. NONON-LATAPIE, ENSAE
7. 1986: Jesus SILVA CASTRO, DEA Automatique, UPS
8. 1987: M. AZEMA, DEA Automatique, UPS
9. 1988: Jean-Christophe HAMANI, DEA Informatique, UPS
10. 1990: Hervé LARUELLE, DEA Automatique, UPS
11. 1990: Christophe DOUSSON, Ecole Polytechnique
12. 1992: François REVILLOD, ENSAE
13. 1995: Samir QUTUB, DEA Automatique, UPS

Participation à des jurys de thèse (extérieurs)

1. Pierre BACQUET, ENSAE, décembre 1988
2. Michel PASQUIER, INPG, janvier 1989
3. Bernadette GASMI, ENSAE, octobre 1990
4. Roger PISSARD-GIBOLLET, Ecole des Mines de Paris, décembre 1993
5. Fouad MEFTOUH, ENSAE (rapporteur), décembre 1993
6. Patrick REGNIER, INPG, décembre 1994
7. Fadi DORNAIKA, INPG (rapporteur), septembre 1995.
8. José Antonio NAJERA MORA INPG (rapporteur), octobre 1995.

Participation à des jurys de thèse (groupe RIA)

1. Ernesto LOPEZ-MELLADO, UPS, décembre 1986
2. Thierry SIMEON, UPS, janvier 1989
3. Isabelle MAZON, UPS, mai 1990
4. Pascal VIOLERO, UPS, novembre 1991
5. Bernard DEGALLAIX, UPS, janvier 1993
6. Hervé LARUELLE, UPS, mars 1994
7. Thierry VIDAL, UPS, septembre 1995.
8. Jérôme PERRET, UPS, novembre 1995.

D PARTICIPATION A COLLOQUES ET CONGRES

Conférences et Workshops Internationaux

- International IEEE Conference on Robotics and Automation, Nagoya, JAPON, Mai 1995.
- ECLA.CIM'94. European Community - Latin America Workshop on CIM. Madrid (Espagne), Novembre 1994 (co-chairman).
- Workshop on Motion Planning, l'Escala (Espagne), Octobre 1994.
- WAFR'94, The First Workshop on the Algorithmic Foundations of Robotics, San Francisco, Février 1994.
- International Conference on Advanced Robotics (ICAR), Tokyo, Novembre 1993 (Organisation avec B. Espiau d'une session portant sur les Architectures de contrôle pour robots autonomes).
- Spring School on Robot Motion Planning, Rodez, Avril 1993.
- IEEE International Conference on Intelligent Robots and Systems, Raleigh, NC, USA, Juillet 1992.
- Workshop on Architectures for Intelligent Control Systems, Nice (France), Mai 1992.
- International IEEE Conference on Robotics and Automation, Nice, FRANCE, Mai 1992.
- Workshop IARP sur la robotique sous-marine, Monterey, USA, Octobre 1990.
- IAS-2, Amsterdam, Pays Bas. Décembre 1989.
- International Symposium on Robotics Research, Tokyo, Japon, Septembre 1989,
- International IEEE Conference on Robotics and Automation, Scottsdale, USA, Avril 1989.
- International Conference on Advanced Robotics (ICAR), Versailles, Octobre 1987.
- Workshop sur la Programmation Automatique de Robots, MIT, Cambridge, Avril 1987.
- 6ème Seminari Sobre Robotica organisé par l'Université de Barcelone, Espagne, Juin 1986.
- International IEEE Conference on Robotics and Automation, San Francisco, USA, Avril 1986.
- International IEEE Conference on Robotics and Automation, Saint-Louis, USA , Avril 1985.
- International IEEE Conference on Robotics and Automation, Atlanta, USA, Mars 1984.
- Conférence Internationale AFCET "Intelligence Artificielle et Reconnaissance des Formes", Paris, Janvier 1984.

Journées Nationales

- Journées Nationales du PRC-IA, Février 1995, Nancy.
- Journée "Plaanification" du PRC-IA, Février 1993, organisateur.
- Quatrièmes Journées Annuelles du PRC-GDR Intelligence Artificielle, Marseille, Octobre 1992.
- Troisièmes Journées Nationales du PRC Intelligence Artificielle, Paris, 1990
- Journées "Robotique", INRIA, Sophia Antipolis, Juin 1987.
- Journées SYSCOROB, ADI, Paris, Décembre 1986.
- Journées Bilan du Programme Automatisation et Robotique Avancée, Paris, Mai 1986.
- Troisièmes Journées Annuelles du Programme ARA, Toulouse, Septembre 1984.

E ACTIVITES DIVERSES

Revue d'articles et de communications

Je contribue régulièrement aux procédures de revue d'articles et de communications pour: l'International Journal of Robotics Research, la Revue d'Intelligence Artificielle, IEEE Conference on Robotics and Automation, IEEE International Conference on Intelligent Robots and Systems (IROS), International Conference on Advanced Robotics (ICAR), Conférence AFCET RFIA.

Groupes de travail du MRT-MRES

- Participation aux réunions du groupe de travail "Cellule Flexible d'Assemblage" et à la rédaction d'un rapport dans le cadre de l'action "Automatisation Intégrée de Production" du Ministère de la Recherche et de la Technologie (mars 1985 - mai 1985).
- Participation aux réunions du groupe de travail "Systèmes de Programmation" et à la rédaction d'un rapport dans le cadre de l'action "Robotique Industrielle" du Ministère de la Recherche et de la Technologie (janvier 1986 - avril 1986).

Expertise et revue de projets

A la suite, de ma participation, en tant qu'adjoint de Georges GIRALT (Responsable Technique National), au projet européen EUREKA-FAMOS, j'ai été amené à participer à l'expertise d'un projet EUREKA portant sur l'automatisation de l'assemblage de lave-linge (juin / juillet 1989).

J'ai également assuré un rôle d'expert auprès de l'ANVAR (Aquitaine) et auprès de la CEE expertise scientifique et suivi de projets ESPRIT Basic Research (à partir de 1990):

- le projet FIRST (3 revues: Leuven, Bruxelles, Oxford)
- le projet MUCOM (2 revues: Bruxelles, Paris)
- le projet SECOND (2 revues: juin 1993 et juin 1994, Leuven)

Diffusion de la connaissance

- Séminaire invité à l'IRIT sur la Coopération Multi-robots, Groupe de travail multi-agent de l'AFCET, Toulouse, Avril 1993.
- Tutorial "Motion Planning with Uncertainties", Spring School on Robot Motion Planning, Rodez, Avril 1993.
- Séminaire invité à l'INRIA (Sophia) sur la programmation au niveau tâche en robotique (mars 1991)

- Séminaire invité au LIFIA (Grenoble) sur la planification automatique de tâches de manipulation (février 1990).
- Séminaire au LAAS sur une approche géométrique du problème de la planification de tâches de manipulation (novembre 1989)
- Séminaire au LAAS sur la prise en compte du temps en planification (novembre 1988)
- Séminaire invité au LIFIA (Grenoble) sur les processus décisionnels pour la programmation de tâches d'assemblage robotisées (mai 1987).
- Exposé sur la conduite d'une Cellule Flexible d'Assemblage. Journée d'Information ADERMIP, Toulouse (mai 1987).
- Séminaire invité à l'université Carnegie Mellon (Pittsburg - USA) sur la programmation et la conduite de Cellules Flexibles d'Assemblage (avril 1987).
- Séminaire invité au CERT (Toulouse) sur les processus décisionnels pour la programmation de Cellules Robotisées (mars 1987).
- Séminaire invité sur la programmation et la conduite de Cellules Flexibles d'Assemblage, Société Hewlett-Packard, Palo Alto, USA (mars 1986).
- Séminaire au LAAS sur les langages de programmation de manipulateurs (décembre 1984).
- Séminaire invité sur l'intégration et la programmation de Cellules Robotisées d'Assemblage, Société Hewlett-Packard, Palo Alto, USA (avril 1984).
- Séminaire invité à l'université de Stanford (USA) sur l'intégration et la programmation de Cellules Robotisées d'Assemblage (avril 1984).

F ACTIVITES D'ENSEIGNEMENT

- Cours d'Initiation au Langage LISP (INSA, Toulouse, 5ème année, Option Informatique Industrielle): 1984 (10 h), 1985 (10 h).
- Cours d'initiation à l'Intelligence Artificielle (ENSM, Nantes, option Robotique): 1987 (12 h), 1988 (12 h).
- Introduction à la Programmation des Robots (UPS, Toulouse, Ingéniorat Robotique et Reconnaissance des Formes): 1988 (4 h) et 1989 (4 h).
- Cours d'Intelligence Artificielle (UPS, DEA Automatique, Option Informatique Industrielle): 10 h par an depuis 1987.
- Cours d'Intelligence Artificielle (UPS, DESS Productique): 10 h par an depuis 1988.
- Cours d'initiation à l'Intelligence Artificielle (ENAC, 2ème année, Toulouse): 1992 (2 fois 8 h), 1993/1994 (4 fois 8 h), 1994/1995 (4 fois 8 h), 1995/1996 (2 fois 8 h).

Syllabus des Enseignements

Introduction à l'Intelligence Artificielle

1. Complexité combinatoire
2. Raisonnement géométrique et spatial
3. Représentation de la connaissance
   - Logique des propositions
   - Représentations procédurales et déclaratives
   - Systèmes à base de règles
4. Représentation et résolution de problèmes
   - Représentation par Graphe d'Etats
   - Représentation par Graphe ET/OU
   - Recherche dans les graphes d'états
   - Recherche dans les graphes ET/OU
5. Applications: planification, perception, systèmes intégrés
6. Introduction et pratique du langage LISP
7. Introduction à la programmation de moteurs d'inférence

Introduction à la programmation des robots

1. Introduction / Les différentes techniques de programmation
2. Les outils d'aide à la programmation
3. Vers la programmation automatique
4. Vers une exécution autonome
5. Trois systèmes en cours de développement

G PUBLICATIONS ET RAPPORTS

Références

Revues Scientifiques

[1] P. Freedman, R. Alami. *The Analysis of Repetitive Sequencing at the Workcell Level: from workcell tasks to workcell cycles.* In *International Journal of Production Research*, Vol 28, Nr 6, pp 1195-1213, Juin 1990.

[2] G. Giralt, R. Chatila, R. Alami. *Task-level Programmable Intervention Robots: the "Remote Operated Autonomous Robot" concept.* In *Robot* Vol 14, Nr 2, pp 1-14, Mars 1992.

[3] T. Siméon, R. Alami *Planning robust motion strategies for a mobile robot in a polygonal world.* Revue d'Intelligence Artificielle, Vol 8, Nr 4, pp 383-401, HERMES, 1994.

Ouvrages de Synthèse (contribution)

[4] R. Alami, H. Chochon. Programming of Flexible Assembly Cells: task modeling and system integration In *ROBOTICS and Industrial Engineering - Selected Readings-* Volume II, 1986.

[5] R. Alami, H. Chochon. Programmation et Contrôle d'Exécution d'une Cellule Flexible d'Assemblage. In *Techniques de la Robotique (Editions HERMES)*, 1988.

[6] M. Ghallab, R. Alami, R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles and B. Roth (Eds), *Robotics Research: The Fourth International Symposium.* pp 431-443, The MIT Press, Massachusetts, 1988.

[7] R. Alami, R. Chatila, M. Ghallab, C. Laugier, J-P. Laumond. Robotique et Intelligence Artificielle. (3ème journée PRC-IA), In *Intelligence Artificielle*, B. Bouchon Ed., Hermés, 1990.

[8] R. Alami, T. Siméon, J.P. Laumond. A Geometrical Approach to Planning Manipulation Tasks. The Case of Discrete Placements and Grasps. In H. Miura and S. Arimoto (Eds), *Robotics Research: The Fifth International Symposium*, pp 453-463, The MIT Press Massachusetts, 1990.

[9] R. Chatila, R. Alami, G. Giralt. Task-level programmable intervention autonomous robots. In *Mechatronics and Robotics I, P. A. MacConaill, P. Drews and K.-H. Robrock Eds, IOS Press*, pp 77-87, 1991.

[10] R. Alami, J.P. Laumond, T. Siméon *Two manipulation planning algorithms* The First Worshop on the Algorithmic Foundations of Robotics, A.K. Peters Pub., Boston, MA. 1994.

[11] R. Alami *A Multi-Robot Cooperation Scheme Based on Incremental Plan-Merging* In, R. Hirzinger and G. Giralt (Eds), *Robotics Research: The Seventh International Symposium*, Springer Verlag, 1996.

Habilitation à diriger des recherches

[12] R. Alami. Habilitation à diriger des recherches *Rapporteurs:* Prof. J.Michael BRADY (Oxford University), Bernard ESPIAU (INRIA Rhone-Alpes), Prof. Richard P. PAUL (Université de Pennsylvanie), *Examinateurs:* Prof. Bernard DUBUISSON (Université de Technologie de Compiègne), Prof. Marc COURVOISIER (Université Paul Sabatier), Jean Louis LACOMBE (groupe MATRA-Hachette), Georges GIRALT (Directeur de Recherche CNRS), Malik GHALLAB (Directeur de Recherche CNRS). Université Paul Sabatier, Février 1996.

Thèse

[13] R. Alami. *Un environnement LISP pour l'intégration et la mise en œuvre de systèmes complexes en robotique.* Thèse de docteur-ingénieur, Institut National Polytechnique de Toulouse, LAAS/CNRS, Toulouse, Novembre 1983.

Conférences avec Comité de lecture

[14] R. Alami. ApocaLisp, un système Lisp pour la robotique. In *4ème Congrès AFCET Reconnaissance des Formes et Intelligence Artificielle, Paris* , Janvier 1984.

[15] R. Alami. NNS, a Lisp-based environment for the integration and operating of complex robotics system. In *IEEE International Conference on Robotics and Automation, Atlanta (USA)*, Mars 1984.

[16] R. Alami, H. Chochon. Programming of Flexible Assembly Cells: task modeling and system integration. In *IEEE International Conference on Robotics and Automation, St Louis (USA)*, Mars 1985.

[17] H. Chochon, R. Alami. NNS, a knowledge-based on-line system for an assembly workcell. In *IEEE International Conference on Robotics and Automation, San Francisco (USA)*, Avril 1986.

[18] H. Chochon, R. Alami. A Knowledge-Based System for Programming and Execution Control of Multi-Robot Assembly Cells. In *'87 International Conference on Advanced Robotics (ICAR), Versailles* , Octobre 1987.

[19] R. Alami, R. Chatila. Decisional Capabilities and Control Architecture for an Advanced Mobile Robot that performs Complex Intervention Tasks in Non-structured Environments. In *2nd Workshop on Manipulators, Sensors and Steps toward mobility*, University of Salford, England, Octobre 1988.

[20] R. Alami, R. Chatila. Robot Manipulation Tasks in Underwater Environments: Decisional Capabilities and Human-Machine Interaction. In *2nd Int. Workshop on Subsea Robotics*, Tokyo, Japan, Novembre 1988.

[21] I. Mazon, R. Alami. Representation and propagation of positionning uncertainties through manipulation robot programs. Integration into a task-level programming system. In *IEEE International Conference on Robotics and Automation, Scottsdale, (USA)*, Mai 1989.

[22] G. Giralt, R. Alami, R. Chatila. Autonomy versus teleoperation for intervention robots? a case for task level teleprogramming. In *Intelligent Autonomous Systems-2, Amsterdam, Netherlands*, Décembre 1989.

[23] P. Freedman, R. Alami. Repetitive Sequencing: from workcell tasks to workcell cycles. In *IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, Mai 1990.

[24] E. Lopez Mellado, R. Alami. A Failure Recovery Scheme for Assembly Workcells. In *IEEE Int. Conf. on Robotics and Automation*, Cincinnati, Ohio, Mai 1990.

[25] I. Mazon, R. Alami, P. Violéro. Automatic planning of pick and place operations in presence of uncertainties. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90)*, Tsuchiura (Japan), Juillet 1990.

[26] R. Alami, R. Chatila, P. Freedman. Task level programming for intervention robots. In *IARP 1st Workshop on Mobile Robots for Subsea Environments, Monterey, California (USA)*, pp 119-136, Octobre 1990.

[27] G. Giralt, R. Alami, R. Chatila, P. Freedman. Remote operated autonomous robots. In *Int. Symp. on Intelligent Robotics, Bangalore, India*, Jan. 1991.

[28] R. Alami, M. Ghallab, H. Laruelle, T. Vidal. Mission Planning and Supervision with Temporal Constraints for Intervention Robots. In *2nd International Symposium on Experimental Robotics,(ISER '91), Toulouse, France*, 1991.

[29] R. Laurette, A. de Saint Vincent, R. Alami, R. Chatila, V. Pérébaskine. Supervision and Control of the AMR Intervention Robot. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pp 1057-1062, Juin 1991.

[30] R. Ferraz De Camargo, R. Chatila, R. Alami. A Distributed Evolvable Control Architecture for Mobile Robots. In *'91 International Conference on Advanced Robotics (ICAR)*, Pisa (Italie), pp 1646-1649, 1991.

[31] R. Chatila, R. Alami, B. Degallaix, V. Pérébaskine, P. Gaborit, P. Moutarlier. An Architecture for Task Refinement and Execution Control for Intervention Robot: preliminary experiment. In *2nd International Symposium on Experimental Robotics, (ISER'91)*, Toulouse (France) 1991.

[32] R. Chatila, R. Alami. *Autonomous task refinement and execution for a planetary rover.* First IARP Workshop on Robotics in Space, Pise (Italie), Juin 1991.

[33] G. Giralt, R. Chatila, R. Alami. Task-level programmable intervention robots: the "remote operated autonomous robot" concept. In *Beijing Workshop on Robotics*, Beijing (Chine), Août 1991.

[34] B. Degallaix, R. Alami, R. Chatila, V. Rigaud. An Architecture for Task Level Programming and Supervision of an Intervention Robot. Application to the NADIA Project. In *4th International Symposium on Offshore, Robotics and Artificial Intelligence (ORIA), Marseille* , Décembre 1991.

[35] R. Chatila, R. Alami, B. Degallaix, H. Laruelle. Integrated planning and execution control of autonomous robot actions. In *IEEE International Conference on Robotics and Automation*, Nice, mai 1992.

[36] B. Dacre-Wright, J-P Laumond, R. Alami. Motion planning for a robot a movable object amidst polygonal obstacles. In *IEEE International Conference on Robotics and Automation*, Nice, mai 1992.

[37] G. Giralt, R. Alami, R. Chatila. Task Level Programming and Robot Autonomy. In *IEEE International Conference on Intelligent Robots and Systems*, Raleigh, NC (USA) Juillet 1992.

[38] R. F. Camargo, R. Chatila, R. Alami. Hardware and Software Architecture for Execution Control of an Autonomous Mobile Robot. In *IECON'92*, San Diego (USA) Novembre 1992.

[39] G. Giralt, R. Chatila, R. Alami. The remote-operated autonomous roobot concept: from deep sea to outer space applications, In *International Planetary Rovers Symposium*, Labège (France) Septembre 1992.

[40] J.-P. Thibault, M. Gallab, B. Degallaix, R. Alami. Execution Control Performed by a Propositional Prover: An Application to an Autonomous Planetary Vehicle. In *Intelligence Artificielle, Robotique et Automatique, Appliquées à l'espace*,CEPADUES Ed., Toulouse, Septembre 1992.

[41] G. Giralt, R. Chatila, R. Alami. Remote Intervention, robot autonomy and teleprogramming: generic concepts and real-world application cases. In *IEEE IROS'93*, Yokohama (Japon), Juillet 1993.

[42] R. Chatila, R. Alami, S. Lacroix, J. Perret, C. Proust Planet exploration by robots: from mission planning to autonomous navigation. In *ICAR'93*, Tokyo (Japon), Novembre 1993

[43] R. Alami, R. Chatila, B. Espiau Designing an intelligent control architecture for autonomous robots. In *ICAR'93*, Tokyo (Japon), Novembre 1993

[44] R. Alami, T. Siméon. Planning robust motion strategies for a mobile robot. In *Proc. IEEE Int. Conf. on Robotics and Automation*, San Diego (USA), mai 1994.

[45] J. Perret, C. Proust, R. Alami, R. Chatila How To Tele-Program a Remote Intelligent Robot, In *IEEE IROS'94*, Munich (RFA), Septembre 1994.

[46] R. Alami, F. Robert, F. F. Ingrand, S. Suzuki. A paradigm for plan merging and its use for multi-robot cooperation. In *IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Antonio, Texas (USA), October 1994.

[47] J. Perret, R. Alami Planning with Non-Dterministic Events for Enhanced Robot Autonomy In *Intelligent Autonomous Systems (IAS-4)*, Eds E. Rembold, R. Dillmann, LO. Hertzberger, T. Kanade, IOS Press, 1995, pp 109-115.

[48] B. Bouilly, T. Siméon, R. Alami A numerical technique for planning motion strategies for a mobile robot in presence of uncertainties In, *IEEE International Conference on Intelligent Robots and Systems* ICRA'95, Nagoya (Japon), Mai 1995.

[49] R. Alami, F. Robert, F. F. Ingrand, S. Suzuki. Multi-robot Cooperation through Incremental Plan-Merging In, *IEEE International Conference on Intelligent Robots and Systems* ICRA'95, Nagoya (Japon), Mai 1995.

[50] L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert *Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment* In, *IEEE IROS'95*, Pittsburgh (USA) 1995.

[51] R. Alami, L. Aguilar, H. Bullata, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, F. Robert *A General framework for multi-robot cooperation and its implementation on a set of three Hilare robots* In *Fourth International Symposium on Experimental Robotics, (ISER'95)*, Stanford (USA) Juin 1995.

[52] F. Ingrand, R. Chatila, R. Alami, F. Robert *Embedded control of autonomous robots using procedural reasoning* In, *ICAR'95*, St Feliu Guixols (Espagne), Septembre 1995, pp 855-861.

[53] F. Ingrand, R. Chatila, R. Alami, F. Robert *PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots* In, *IEEE International Conference on Intelligent Robots and Systems* ICRA'96, Minneapolois (USA), Avril 1996.

[54] R. Alami *A Multi-robot Cooperative Scheme based on Distributed and Incremental Plan Merging* In International Symposium on Robotics and Manufacturing, ISRAM'96, Second World Automation Congress, Montpellier, Mai 1996.

[55] R. Chatila, F. Ingrand, R. Alami *Integration of Deliberation and Reaction for Autonomous Mobile Robots* In International Symposium on Robotics and Manufacturing, ISRAM'96, Second World Automation Congress, Montpellier, Mai 1996.

Colloques avec actes à diffusion restreinte

[56] *Troisièmes Journées du Programme ARA*, Toulouse, Septembre 1984.

[57] *Journées BILAN du Programme ARA*, Paris, Mai 1986.

[58] R. Alami, H. Chochon. Programmation et Conduite de tâches d'assemblage robotisées: modélisation et processus décisionnels. In *Séminaire SYSCOROB: Architecture Matérielle et Logicielle des Contrôleurs en Robotique, Paris* , Décembre 1986.

[59] R. Alami, J-P. Courtiat, M. Diaz, A.M. Druilhe, J.F. Lenotre, V. Mazzola, L. Steffan. Conception en ESTELLE du Pilotage d'une Cellule Flexible d'Assemblage. In *Séminaire Franco-Brésilien*, Brésil, Juillet 1989.

[60] J.P. Laumond, R. Alami. A geometrical approach to planning manipulation tasks in Robotics. In *The First Canadian Conference on Computational Geometry, Montréal (Canada)*, Août 1989.

[61] R. Chatila, R. Alami, R. Prajoux. An Architecture Integrating Task Planning and Reactive Execution Control. *Workshop On Architectures For Intelligent Control Systems, IEEE Int. Conf. on Robotics and Automation*, Nice, Mai 1992.

[62] R. Alami, M. Borillo, F. Garcia, M. Ghallab, H. Laruelle. Représentation et algorithmes pour la planification et l'action. In *Actes des 4èmes Journées Nationales, PRC-GDR Intelligence Artificielle, Marseille, Teknea*, Octobre 1992.

[63] R. Alami, T. Siméon. Planification de trajectoires robustes en présence d'incertitudes. In *Journées Planification et Contrôle de Mouvement en Robotique, PRC-IA, GRECO Automatique, Toulouse*, Janvier 1993.

[64] R. Alami Architectures for the Control of Autonomous Robots. ECLA.CIM'93. European Community - Latin America Workshop on CIM. Lisbonne (Portugal), Novembre 1993.

[65] R. Chatila, R. Alami, S. Lacroix, J. Perret, C. Proust "How to Explore a Planet with a Mobile Robot" IARP Second Workshop on Robotics in Space, Montréal, Canada, Juillet 1994.

[66] R. Alami, L. Aguilar, F. Robert, S. Fleury Multi-robot Navigation through Incremental Merging of Motion Plans ECLA.CIM'94. European Community - Latin America Workshop on CIM. Madrid (Espagne), Novembre 1994.

[67] R. Chatila, F. Ingrand, R. Alami *Mission Planning and Execution Control for Intervention Robots* US/Portugal Workshop on Undersea Robotics and Intelligent Control, Lisbonne (Portugal), Mars 1995.

[68] R. Alami *A Generic Framework for Multi-robot Cooperation and its use for a Fleet of Autonomous Mobile Robots for Load Transfer Applications* IARP, The first worshop on Robotics for the Service Industries, Sydney (Australie), Mai 1995.

Rapports Scientifiques

[69] R. Alami. Manuel ApocaLisp. Rapport de recherche 83044, LAAS/CNRS, Toulouse, Juillet 1983.

[70] R. Alami. ApocaLisp 2.2. Rapport de recherche 85098, LAAS/CNRS, Toulouse, Juillet 1985.

[71] J.P. Laumond, R. Alami. A geometrical approach to planning manipulation tasks: the case of a circular robot and a movable circular object amidst polygonal obstacles. In *Technical Report LAAS 88314*, Octobre 1988.

[72] R. Alami. Manuel utilisateur CPU. Rapport 89259, LAAS/CNRS, Toulouse, 1989.

[73] P. Freedman, R. Alami The distributed control of a robotics workcell: two perspectives Rapport 89424, LAAS/CNRS, Toulouse, Décembre 1989.

[74] B. Degallaix, H. Laruelle, R. Alami, R. Chatila. Téléprogrammation au niveau tâche pour des robots d'intervention. Rapport 91145, LAAS/CNRS, Toulouse, Décembre 1990.

[75] R. Alami, T. Siméon. Planning robust motion strategies for a mobile robot. Rapport LAAS 93005, LAAS/CNRS, Toulouse, Janvier 1993.

Rapports de Contrats

[76] R. Alami, M. Briot, M. Devy. Capteurs et systèmes associés au robot assurant la distribution d'objets à assembler dans une unité d'assemblage automatique flexible. Convention ADI-MATRA-LAAS N.81.195, Rapport final 86030, LAAS/CNRS, Toulouse, Juillet 1986.

[77] R. Alami, R. Chatila - LAAS/CNRS Planification de mission et téléprogrammation niveau tâche Projet VAP, Thème 10, Rapport 1.10.F de Fin de Phase 1, Rapport LAAS 90003, Janvier 1990, 27p.

[78] R. Chatila, R. Alami - LAAS/CNRS Gestion des tâches et structure décisionnelle Projet VAP, Thème 6, Rapport 1.6.F de Fin de Phase 1, Rapport LAAS 90005, Janvier 1990, 17p.

[79] R. Alami, R. Chatila, M. Ghallab, R. Sobek - LAAS/CNRS Planification de mission et téléprogrammation niveau tâche Projet VAP, Thème 10, Rapport 2.10.F de Fin de Phase 2, Rapport LAAS 90149, Mai 1990. 52p.

[80] R. Chatila, R. Alami, R. Prajoux - LAAS/CNRS Gestion des tâches et structure décisionnelle Projet VAP, Thème 6, Rapport 2.10.F de Fin de Phase 2, Rapport LAAS 90150, Mai 1990, 32p.

[81] R. Chatila, R. Alami - LAAS/CNRS Gestion des tâches et structure décisionnelle Projet VAP, Thème 6, Rapport 3.6.I de Fin de Phase 3, Rapport LAAS 901445, Novembre 1990, 10p.

[82] R. Alami, R. Chatila, R. Sobek - LAAS/CNRS Planification de mission et téléprogrammation niveau tâche Projet VAP, Thème 10, Rapport 3.10.I de Fin de Phase 3, Rapport LAAS 90444, Novembr 1990, 11p.

[83] R. Alami, R. Chatila - LAAS/CNRS Planification de mission et téléprogrammation niveau tâche Projet VAP, Thème 10, Rapport 3.10.F de Fin de Phase 3, Rapport LAAS 91021, Janvier 1991, 50p.

[84] R. Chatila, R. Alami - LAAS/CNRS Gestion des tâches et structure décisionnelle Projet VAP, Thème 6, Rapport 3.6.F de Fin de Phase 3, Rapport LAAS 91020, Janvier 1991, 61p.

[85] R. Alami, R. Chatila - LAAS/CNRS Projet VAP, Structure décisionnelle et téléprogrammation niveau tâche, Rapport de fin de phase 4, Rapport LAAS 92030, Janvier 1992, 58p.

[86] G. Giralt, R. Alami. Systèmes intelligents de production Contrat MRT N86.P.07.87, Juin 1992, 324p.

[87] R. Alami, R. Chatila - LAAS/CNRS Projet VAP, Structure décisionnelle et téléprogrammation niveau tâche, Rapport de fin de phase 5, Décembre 1992, 41p.

[88] R. Alami, R. Chatila, S. Fleury, M. Herrb, J. Perret, C. Proust Téléprogrammation niveau tâche et structure décisionnelle embarquée. Rapport LAAS 93273, Juillet 1993, 71p.

[89] F. Ingrand, R. Alami, F. Robert Deliverable D31.1.1. Software Design Specification for the Robot Supervisor MARTHA ESPRIT Project Nr 6668, Novembre 1993, 55p.

[90] R. Alami Deliverable D32.1.1. Software Design Specification for the Motion Planning Sub-system MARTHA ESPRIT Project Nr 6668, Novembre 1993, 38p.

[91] R. Alami Deliverable D32.1.2. User Manual for the Motion Planning Sub-system MARTHA ESPRIT Project Nr 6668, Novembre 1993, 19p.

[92] R. Alami Deliverable D32.2.3. Architectural Design of the Motion Planning Sub-system MARTHA ESPRIT Project Nr 6668, Janvier 1994, 25p.

[93] R. Alami Deliverable D31.2.4. The MARTHA Environment Model and its use MARTHA ESPRIT Project Nr 6668, Janvier 1994, 26p.

[94] R. Alami, F. Robert, F. Ingrand Deliverable D31.2.5. External Interfaces of a Robot Control System MARTHA ESPRIT Project Nr 6668, Janvier 1994, 59p.

[95] R. Alami, F. Robert, F. Ingrand Deliverable D31.2.6. Architectural Design for the Robot Supervisor Sub-system MARTHA ESPRIT Project Nr 6668, Janvier 1994, 35p.

[96] R. Alami, R. Chatila, M. Herrb, F. Ingrand, J. Perret, C. Proust Groupement RISP, Projet I.ARES, Téléprogrammation niveau tâche et structure décisionnelle embarquée, Rapport final. Rapport LAAS 94081, Février 1994.

[97] R. Alami et al. MARTHA-PRO-94-2567-TN Muti-robot Demonstration 1: Demonstration of Planning and Decisional Capabilities including advanced multi-robot cooperation MARTHA ESPRIT Project Nr 6668, Octobre 1994, 29p.

[98] R. Alami, S. Fleury, M. Herrb MARTHA-PRO-94-2551-TN-A, Interactions between the Robot Supervisor and the other Vehicle Sub-systems Rapport LAAS 95290, MARTHA ESPRIT Project Nr 6668, Juin 1995, 69p.

[99] R. Alami, F. Robert MARTHA Environment model syntax Rapport LAAS 95291, MARTHA ESPRIT Project Nr 6668, Mars 1995, 27p.

[100] S. Fleury, M. Herrb, R. Alami Preliminary guide to the RCS Integration Rapport LAAS 95293, MARTHA ESPRIT Project Nr 6668, Mars 1995, 18p.

[101] F. Robert, R. Alami Interface between CCS and RCS Rapport LAAS 95294, MARTHA ESPRIT Project Nr 6668, Mars 1995, 30p.

Diffusion de l'information scientifique

[102] R. Chatila, R. Alami Planification de mission et structure décisionnelle embarquée, Bulletin de liaison de la Recherche en Informatique et en Automatique, Numéro Spécial sur l'exploration de Mars par engin mobile, Nr 139, 1992

[103] R. Alami Gares, ports et aéroports du futur. La Recherche, Nr 258, Octobre 1993.

H THESES ENCADREES

[104] H. Chochon. Programmation de tâches d'assemblage robotisées: modélisation et processus décisionnels. Thèse de l'Université Paul Sabatier, Toulouse, Novembre 1986.

[105] E. Lopez Mellado. Le contrôle d'exécution dans les Cellules Flexibles d'Assemblages. Thèse de docteur ingénieur, Université Paul Sabatier, Toulouse, Décembre 1986.

[106] T. Siméon. Génération automatique de trajectoires sans collision et planification de tâches de manipulation en robotique. Thèse de l'Université Paul Sabatier, Toulouse, Janvier 1989.

[107] I. Mazon. Raisonnement sur les contraintes géométriques et d'incertitudes pour la planification de tâches de manipulation robotisées. Thèse de l'Université Paul Sabatier, Toulouse, Mai 1990.

[108] P. Violéro. SPARA : Un système de programmation automatique de tâche de manipulation robotisée. Thèse de l'Université Paul Sabatier, Toulouse, Novembre 1991.

[109] B. Degallaix. Une architecture pour la Téléprogrammation au niveau tâche d'un robot mobile autonome. Thèse de l'Université Paul Sabatier, Toulouse, Janvier 1993.

[110] H. Laruelle Planification Temporelle et exécution de tâches en robotique. Thèse de l'Université Paul Sabatier (co-encadrement avec M. Ghallab), Toulouse, Mars 1994.

[111] J. Perret Téléprogrammation au niveau tâche pour un robot mobile autonome d'intervention sur site distant. Thèse de l'Université Paul Sabatier, Toulouse, Novembre 1995.

I SELECTION DE PUBLICATIONS

Thématiques

I - Cellules Flexibles d'Assemblage

1 *Programming of Flexible Assembly Cells: task modeling and system integration*

IEEE International Conference on Robotics and Automation, St Louis (USA), Mars 1985. (paru également dans *ROBOTICS and Industrial Engineering - Selected Readings-* Volume II, 1986.)

Auteurs: R. Alami, H. Chochon.

2 *NNS, a knowledge-based on-line system for an assembly workcell.*

IEEE International Conference on Robotics and Automation, San Francisco (USA), Avril 1986.

Auteurs: H. Chochon, R. Alami.

3 *The Analysis of Repetitive Sequencing at the Workcell Level: from workcell tasks to workcell cycles.*

International. Journal of Production Research, Vol 28, Nr 6, pp 1195-1213, Juin 1990.

Auteurs: P. Freedman, R. Alami.

II - Planification d'actions

4 *Dealing with Time in Planning and Execution Monitoring.*

R. Bolles, editor, *Robotics Research: The Fourth International Symposium.* MIT Press, Mass., 1988.

Auteurs: M. Ghallab, R. Alami, R. Chatila.

5 *Planning with Non-Determistic Events for Enhanced Robot Autonomy*

Intelligent Autonomous Systems, Karlsruhe (RFA), Mars 1995.

Auteurs: J. Perret, R. Alami.

III - Architectures de Contrôle pour Robots Autonomes

6 *Integrated planning and execution control of autonomous robot actions.*

IEEE International Conference on Robotics and Automation, Nice, mai 1992.

Auteurs: R. Chatila, R. Alami, B. Degallaix, H. Laruelle.

7 *Designing an intelligent control architecture for autonomous robots.*

International Conference on Advanced Robotics *ICAR'93*, Tokyo (Japon), Novembre 1993

Auteurs: R. Alami, R. Chatila, B. Espiau

8 *Planet exploration by robots: from mission planning to autonomous navigation.*

International Conference on Advanced Robotics *ICAR'93*, Tokyo (Japon), Novembre 1993

Auteurs: R. Chatila, R. Alami, S. Lacroix, J. Perret, C. Proust.

9 *How To Tele-Program a Remote Intelligent Robot.*

IEEE *IROS'94*, Munich (RFA), Septembre 1994.

Auteurs: J. Perret, C. Proust, R. Alami, R. Chatila

IV - Planification de Tàches

10 *Automatic planning of pick and place operations in presence of uncertainties.*

IEEE International Workshop on Intelligent Robots and Systems *IROS '90*, Tsuchiura (Japan), Juillet 1990.

Auteurs: I. Mazon, R. Alami, P. Violéro.

11 *Two manipulation planning algorithms*

The First Worshop on the Algorithmic Foundations of Robotics, A.K. Peters Pub., Boston, MA. 1994.

Auteurs: R. Alami, J.P. Laumond, T. Siméon

12 *Planning robust motion strategies for a mobile robot in a polygonal world.*

Revue d'Intelligence Artificielle, Vol 8, Nr 4/1994, pp 383-401, HERMES, 1994.

Auteurs: T. Siméon, R. Alami

V - Coopération multi-robot

13 *Multi-robot Cooperation through Incremental Plan-Merging.*

IEEE International Conference on Intelligent Robots and Systems (*ICRA'95*), Nagoya (Japon), Mai 1995.

Auteurs: R. Alami, F. Robert, F. F. Ingrand, S. Suzuki

14 *Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment*

IEEE *IROS'95*, Pittsburgh (USA) 1995.

Auteurs: L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert.

15 *A Multi-Robot Cooperation Scheme Based on Incremental Plan-Merging*

In, R. Hirzinger and G. Giralt (Eds), *Robotics Research: The Seventh International Symposium*, Springer Verlag, 1996 (à paraitre)

Auteurs: R. Alami

J.1 Cellules Flexibles d'Assemblage

1 *Programming of Flexible Assembly Cells: task modeling and system integration*

International Conference on Robotics and Automation, St Louis (USA), Mars 1985. (paru également dans *ROBOTICS and Industrial Engineering - Selected Readings-* Volume II, 1986.)

Auteurs: R. Alami, H. Chochon.

2 *NNS, a knowledge-based on-line system for an assembly workcell.*

IEEE International Conference on Robotics and Automation, San Francisco (USA), Avril 1986.

Auteurs: H. Chochon, R. Alami.

3 *The Analysis of Repetitive Sequencing at the Workcell Level: from workcell tasks to workcell cycles.*

International Journal of Production Research, Vol 28, Nr 6, pp 1195-1213, Juin 1990.

Auteurs: P. Freedman, R. Alami.

PROGRAMMING OF FLEXIBLE ASSEMBLY CELL: TASK MODELLING AND SYSTEM INTEGRATION

Rachid ALAMI - Helene CHOCHON

Laboratoire d'Automatique et d'Analyse des Systemes du CNRS
7, Avenue du Colonel Roche
31077 Toulouse CEDEX
France

Abstract

This paper is divided into four sections. The first section discusses Flexible Assembly Cell (FAC) programming requirements. Sections 2 and 3 present an approach to FAC modelling and programming. The intended objective is to build a programming and decisional environment which allows construction of assembly applications, and to endow the on-line system with decision-making capabilities. The last section describes the current state of an actual system which is a first approach to meeting the discussed requirements.

INTRODUCTION

Flexible Assembly Cells (FAC) programming is a major concern as it requires integration of a broad spectrum of techniques into a complete, highly autonomous system successfully interfaced with its environment. This paper presents an approach to FAC programming and reviews results of research on the experimental Flexible Assembly Cell of the French Advanced Robotics and Automation (ARA) joint program.

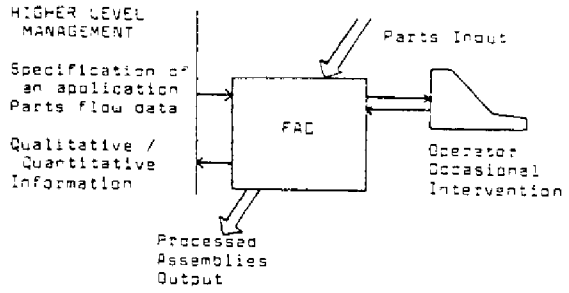
The first section examines various features of Flexible Assembly Cells. It focuses on flexibility and the capabilities it underlies: adaptability, autonomy (during execution time) and configurability, versatility (during setting up and programming time).
Section 2 presents an approach to FAC modelling; three aspects are examined: application specification, functional classification of FAC components, and FAC space organization.
In the third section, this modelling is used to define elementary actions that can be executed, and organize them in a program structure. We show that this modelling provides a suitable framework for embedding different subsystems into a programming and decisional environment for FACs. Moreover, it provides a basis for endowing the on-line system with decision-making capabilities such as task execution monitoring, or interaction with shop floor management systems, or with the operator.
Section 4 addresses FAC on-line system, taking as an illustrative example the FAC developed at Laboratoire d'Automatique et d'Analyse des Systemes.

1- FAC PROGRAMMING REQUIREMENTS

A Flexible Assembly Cell is the place, in an automated factory, where assembly tasks and associated operations are actually performed. It typically includes a number of closely linked components such as manipulators, sensors and peripheral/specialized devices. Such a system can be evaluated in various terms. We will examine herein in some detail questions related to FLEXIBILITY. Two related FAC aspects are to be studied: during execution time, and during programming time.

1.1 During execution time

During execution time, a FAC must behave as an autonomous system (Figure 1), that delivers assembled objects according to some preplanned process while supplied with various parts. Furthermore, it must be able to adapt to dynamic changes in the shop environment and to provide user-interface facilities for occasional operator interventions.
Scheduling, loading and unloading of FACs, adaptive route selection, and determination of various process variables are conducted under the control of the shop floor system[1]. A FAC must respond to various orders and take into account information proceeding from the shop level management system.
A typical interaction is the specification of a task to be performed by the FAC. Here, we must emphasize the fact that a FAC should be programmed not for a single task but for a series of tasks which pertain to the same application. This can be very useful if there exists some functional redundancy between two or several FACs operating in the plant. For instance, an inspection operation that was formerly carried out by a FAC, could be dynamically assigned to another FAC which incorporates a functionally equivalent inspection apparatus.
Other interactions concern managerial information: for instance, a FAC must take into account information about parts input flows.
A FAC should also be able to provide information about its current ability to achieve a task, or qualitative/quantitative data about the task currently performed. For instance, it must be able to inform that it can no longer deal with workpieces of a specified variant (due to a malfunctioning device).



- Figure 1 -

Besides, a cell must provide user-interface facilities and aid for problem diagnosis and error-recovery.
Another key problem is FAC autonomy. There must be provision for 1) management of various local resources, 2) alteration of plans based on more recent information, and 3) coping with various events occuring at the FAC level, in a "transparent" manner with respect to higher level systems. Resorting to higher level systems (including the operator) is done only if the task to be performed cannot be achieved according to the specifications (fatal failure, inability to meet requirements of precision, time-constraints..).

1.2 During programming time

This section examines briefly methods and tools required for programming a FAC. Design objectives involve three major aspects.
First is system integration. The software/hardware architecture must provide facilities for integrating various sub-systems into a programmable system. Addition and suppression of modules - even specialized devices - must be possible with minimum effort. In section 4 a system is described, NNS, which meets such requirements[2].
Second, it is necessary to set up a framework for constructing applications and for accessing and using in a coherent manner, data and programs involving different fields of advanced robotics.
Finally, the structure of the programs and of the FAC on-line system should take into account various aspects previously introduced: FAC state management, interaction with decision-making systems, dynamic changes in planned processes, and resource management.
The next sections discuss a FAC modelling and the associated off-line/on-line system which represent a first approach towards meeting these requirements.

2-FAC MODELLING

Addressing FAC modelling, one has to deal with a real-world environment that comprises various components - including very sophisticated automation systems - which are to be coordinated for performing repeatedly over time complex assembly applications.
As stated before, autonomy of the whole system plays a crucial role. Whereas much of the planning takes place before execution, changes in the environment force the FAC to adapt to various events by making decisions based on its current model and on information exchanged with external systems. We introduce a means of specifying applications achieved by the FAC together with a functional description of every component and a FAC space structuring. These three aspects will serve not only to deal with "on-line" problems such as failure diagnosis, break-down recovery and coping with various events, but also to construct new applications.

2.1 Specifying an application

A FAC application will be described by the set of parts to be processed, their origin, i.e. supplied to the cell by a feeder system or created within the FAC, and their target, i.e. to be delivered by the FAC, to be composed with other parts or processed by a specialized device.
A part-mating or manufacturing* operation and, more generally, every action which results in a change in some characteristics of a part, will be considered as a creation of a new part. For instance, an inspection operation which concludes that a part is defective, results in the "creation" of a new part.
Additional information may be provided in order to simplify FAC programming or to improve FAC behavior. For instance, parts which differ only in a few parameters (colour, size..) are grouped in classes; Likewise, information about flows of parts delivered to the FAC, which can be completely specified, partially specified or arbitrary, may allow to optimize the sequencing of actions and the management of local resources.

2.2 Functional classification of FAC components

FAC components fall into three classes - tools, sensors and effectors- which are examined with respect to their functional properties.
Tools: This term covers a great variety of specialized or peripheral devices: complex machine-tools that perform specific composing or manufacturing operations, and any kind of vices, grippers, feeders, unloading apparatus.
The function of a tool is determined by its ability to hold a part and to (possibly) transform it. Tools may be located at fixed positions or movable using effectors.
A tool is operated through a set of commands, the effect of which is defined in various terms: creation of a new part, establishing/suppressing geometrical relations between the tool and the part and the entailed consequences (degrees-of-freedom of a part in the tool..). Other commands may provide information about the state of the tool.

Sensors: We distinguish three different modes for the use of sensing: consulting mode, monitoring mode and controlling (servoing) mode. A same sensor may be used alternatively in different modes.

* : Through all the paper, "manufacturing" is to be understood as an action of specialized devices, included in the assembly process and that results in part modification: e.g. screwing, riveting...

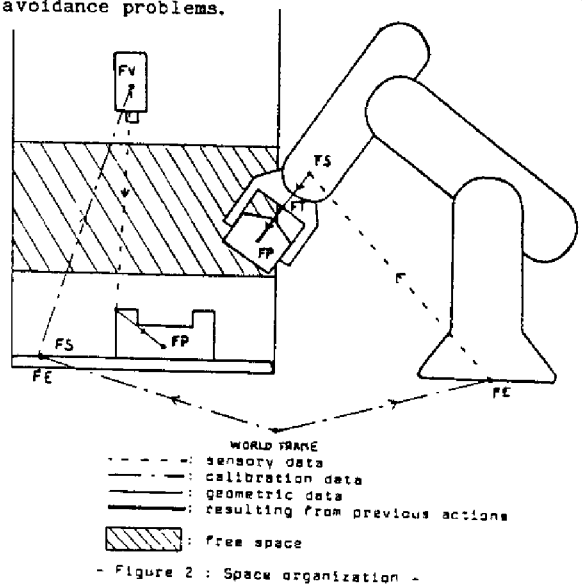
In the consulting mode, sensors provide information that is used to check if an action had the desired effect, or to determine parameters for subsequent actions. Typical applications are part identification, location and inspection, detecting object presence or absence and so forth.
In the monitoring mode, conditions on sensory data are checked at high rate and, if matched, entail execution of predetermined actions. Typical examples are guarded-motions, and action-dependant sensory condition monitoring such as grasping with specified gripping-force.
The third mode involves sensors that are directly linked to an effector (for the sake of efficiency). It is used in situations where manipulator positions are continuously updated in response to sensory data.
As FAC programming involves essentially combining of fairly high-level functions and execution monitoring of the whole system, only the two first modes are directly addressed. Sensor-controlling capabilities will be used within part-mating or special-purpose primitives (see Section 3).

Effectors: Every system able to move parts or tools is an effector. Actions performed by an effector can be more or less complex, according to its number of degrees-of-freedom, its workspace, and capabilities provided by its control system.
We distinguish two effector uses: to transport a part or tool from a workplace to another workplace; and to perform fine motions within a workplace in order to achieve complex operations as part-mating..
General-purpose manipulators allow both uses, whereas other effectors are specially designed for the first use (conveyor belts, rotary tables..) or the second use (fine-positioning wrists..).

2.3 Spatial organization

The functional description of FAC components does not deal with the FAC spatial organization. Simple questions, as where are the parts in the cell, which is the orientation of a part on the conveyor belt, what is the position of a manipulator hand..., involve an organization of the FAC space. To enable this organization, we define the notion of site.
A site is a place in the cell capable to support one workpiece at a time. A site is always tied to a single effector, and may be equipped with various tools and sensors. A coordinate frame FS is attached to every site. FS is independant of the tool and the workpiece present on the site.
Location of sites. A FAC is only composed of apparatus at fixed places; every effector is attached to a reference frame FE, related to the world frame by an invariable transform. These transforms, which enable to link effectors, are obtained by calibration methods. The location of a site is defined as the transform relating FE to FS. It is generally provided by sensory data, which specify the effector configuration, and by a geometric model of the effector.
Orientation and location of parts in the sites. The part orientation in a site includes all the information which specify how a part is held by a tool, i.e. the part faces or edges touching the tool, the degrees-of-freedom of the part in the tool, the position stability.. This information is crucial, as it has consequences on the following actions which can be performed on a part: for instance, depending on its orientation in a gripper, a part can be directly assembled or must be first turned over. Computer Aided Design systems offer facilities to define, automatically or in an interactive way, orientations of parts in the sites.
The part location in a site is given by the transform relating FS to a reference frame FP of the part. This transform is obtained from the location of the tool present in the site, i.e. from the transform relating FS to a reference frame FT of the tool, and from the part orientation in the site, which provides a transform relating FP to FT (Figure 2). When the part location is unknown or partially unknown or when the uncertainties are too important with respect to the allowances, a sensor must provide the part location. It is assumed that we are capable to interpret sensory data so as to obtain the transform relating FS to a known frame attached to the part.

Structuring FAC space. The FAC spatial organization cannot be restrained to a world model with coordinate frames: we need a discretization of the FAC space and we must take into account the space occupied by objects in the cell. This is in particular the case when we deal with collision avoidance problems.



- Figure 2 : Space organization -

The FAC workspace is subdivided into space regions associated to each site. The position of a site tied to a manipulator is given by the name of the space where is the origin of FS. For each of these regions, we distinguish a "free space" and a "constraint space". Indeed this partitioning of site spaces agrees with the two different uses of effectors (see section 2). Gross motions are defined so that the site tied to a manipulator remains inside "free spaces", whereas fine-motions are performed inside a "constraint space". CAD and geometric reasoning systems can be used to specify

site spaces, to generate off-line optimized obstacle-free gross motions[3], and to build fine-motion strategies based on parts geometric description and manipulator kinematics models[4,5,6].
Figure 2 summerizes the relationships between the different frames. In particular, it illustrates how this spatial organization fits well with the processing of inaccuracies, as it distinguishes between the different causes of uncertainty: manipulation or sensors resolution, part tolerances, calibration precision, uncertainty of the geometric models[7].

Finally, with this structuring of FAC workspace, we get a means to define a FAC state which specifies the parts present in each site and their orientation. In the next section, we will examine how this FAC modelling enables to specify elementary actions performed by a FAC and to build a process plan, which constitutes a suitable framework for programming and controlling an application.

3- FAC Programming

3.1 Elementary actions

The FAC state is represented at every moment by the parts that are present on each site and their orientations. Therefore we define an elementary action as a transition of the FAC state, i.e. the change of site and orientation for a part; such actions may result in the creation of a new part when parts are assembled or processed by machine-tools. We next give two examples of elementary actions.
The first example consists of picking a part A on a conveyor belt; the second example is the assembly operation of a part A and a part B on a table, resulting in the creation of a part AB.

ex 1 :
(conveyor part-A orien-1)
(robot-hand - -)
——> (conveyor - -)
(robot-hand part-A orien-2)

ex 2 :
(table-site1 part-B orien-3)
(robot-hand part-A orien-2)
——> (table-site1 part-AB orien-1)
(robot-hand - -)

The elementary actions at FAC level are often complex actions; nevertheless, our purpose is to show how these actions are decomposed by means of primitive functions that may invoke sophisticated procedures. Indeed the process of an elementary action is subdivided into four steps:

1- **Establishing of the initial states of sites involved in the action:** these states are partly known a priori as the result of previous actions, but are to be completely specified ,or simply checked, using sensory information. This step uses identification and location procedures and consulting functions of sensors available in the sites, to set up conditions about the initial state.

2- **Preparation of the sites for the action:** this step consists of changing the tools if necessary, setting the tools ready to act with the right parameters, bringing two sites closer.. At this step, all these primitive operations are independant and can be performed concurrently. They are monitored by sensors in order to control their execution. For instance, there must be a quick reaction if a gripper releases a part during a gross motion or if a vice is broken and cannot be opened.

3- **Execution of the action:** it is the step which carries out the transition of the FAC state. It involves close interaction between parts, tools and effectors. These constraints might require the use of special purpose procedures, involving fine motion strategies, part-mating compliant motion generation.. These procedures are not addressed directly at a FAC-level description of the actions, but are taken into account to provide them with information related to their execution context; they are used through primitive functions.

4- **Validation of the action,** to ensure with sensors that the reached state for each site is the goal state. Like the first step, it requires sensor consulting functions and inspection procedures.

The primitive functions embody the FAC components functional aspects and include the FAC spatial modelling. They use their own knowledge base - generated off-line using specialized decisional modules - to specify an operation and to interface with the low level software structure. Several examples of such primitive functions illustrate how they are implemented and used to program an elementary action:

- **LOCALIZE** takes a site, a part and its orientation as parameters and returns the part location. It requires data such as calibration of the camera on the site, the transform relating the frame located by the camera to the reference frame of the part FP and other parameters of the vision system (threshold, vision window, object models..).

- **MOVE** takes an effector, a site tied to this effector and a position as parameters. It finds a safe trajectory according to the FAC state, and executes it monitored by state-dependant sensory conditions.

- **APPROACH** takes a part with its initial and final states as parameters. It finds and performs an approach motion, related to the part and tools present on the sites.
The elementary action of example 1 can be decomposed as follows:

1st step: (TEST-COND light-beam part-present)
(LOCALIZE conveyor-belt part-A orien1)
2nd step: (MOVE robot robot-hand conveyor-pos)
(PREPARE robot-hand part-A orien2)
3rd step: (APPROACH part-A (conveyor-belt orien1)
(robot-hand orien2))
(MONITOR-COND switches-1 part-present)
(EXTRACT part-A (conveyor-belt orien1)
(robot-hand orien2))
4th step: (TEST-COND switches-1 part-present)

We can notice here that this definition of elementary actions performed by a FAC and their decomposition in steps, enable a simple resource management and offer a basis for concurrent execution (see Section 4).

3.2 Application process plan

The set of elementary actions that can be achieved by the FAC for a given application, constitutes the application process plan. It is represented by a network, whose nodes are part states in the FAC and whose arcs represent possible elementary actions. The network is gradually constructed beginning with the original part states in the cell: from each node, all feasible elementary actions are determined. In section 4, we give an example of such a plan.
The application process plan constitutes a suitable framework for accessing and relating, during programming phase, various specialized systems. It provides relevant information that enables a broad spectrum of possibilities (Figure 3). Typical processing might include studying feasability of the whole process, spatial organization, dealing with uncertainty[8,6], time estimation, simulation, optimization issues[9].
During execution time, it is used to decide which elementary actions to perform, according to FAC state and external events such as new parts arrival. Conflicts that may occur at this stage, are taken into account at programming time. Indeed, the number of possible FAC states could be very important and on-line conflict resolution might lead to inefficiency.In its current implementation, every arc of the network is labeled with a condition about the FAC state, so that deadlock situations are avoided and the process is "optimized" with respect to FAC state.
The on-line system, rather than executing classical programs, interprets a network structure in interaction with the environment. Such a feature is central if one wants some flexibility.

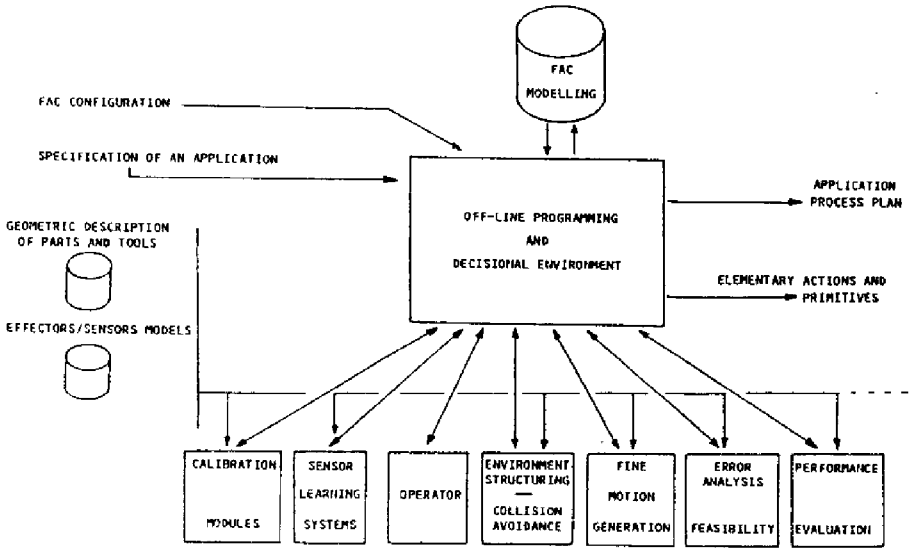
Indeed:
- the managment of the various FAC components is clearly separated from "programs" (process plan) that are to be achieved; this allows a simple definition of software layers as well as it provides an easy means to introduce other software modules;
- this kind of programming allows easy FAC state management, and selection of actions to be performed with respect to the FAC state. For the same reasons, interruption handling and starting again from another state is possible without heavy processing;
- it allows an incremental development of FAC capabilities. This concerns decisional processes that deal with FAC and process state, such as failure diagnosis systems, error-recovery procedures, user interface or interface with higher level managment systems.

4-FAC on-line system

Throughout this section, we will use as an illustrating example the FAC of the ARA research program. We describe successively its actual configuration and an application example that will serve as a guide while discussing the on-line system software layers.

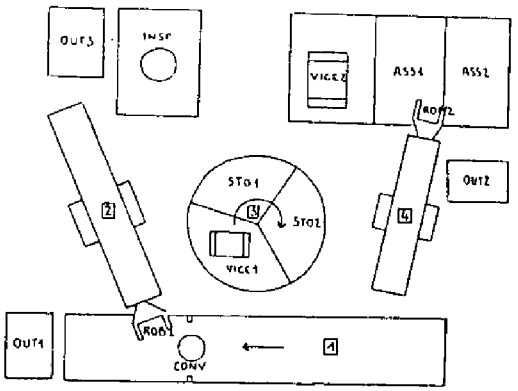
4.1 The FAC of the ARA program

The FAC of the ARA program is intended to serve as a realistic experimental mainframe for testing, validating and developing various contributions dealing with different fields in advanced robotics[10].
The computer environment is composed of a real-time oriented 32-bit GOULD-SEL mini-computer system and various microprocessors. The robotics environment consists mostly of two six degrees-of-freedom manipulators (Renault-Acma TH8, SCEMI).



- Figure 3 -

Various other systems are installed including sensory modules (cameras, proximeters, micro-switches..) and numerous auxiliary devices (conveyor belt, rotary table, vices, special-purpose tools..).
A salient feature of this hardware/software configuration is that it is diversified and in constant evolution. A great variety of instrumental and software tools are to be developed and installed, enlarging the spectrum of possible experiments (various vision systems, force-controlled motion software[11], fine-positioning devices..).



[1] : conveyor belt
[2] : TH8 manipulator
[3] : rotary table
[4] : SCEMI manipulator

- Figure 4 : an application example -

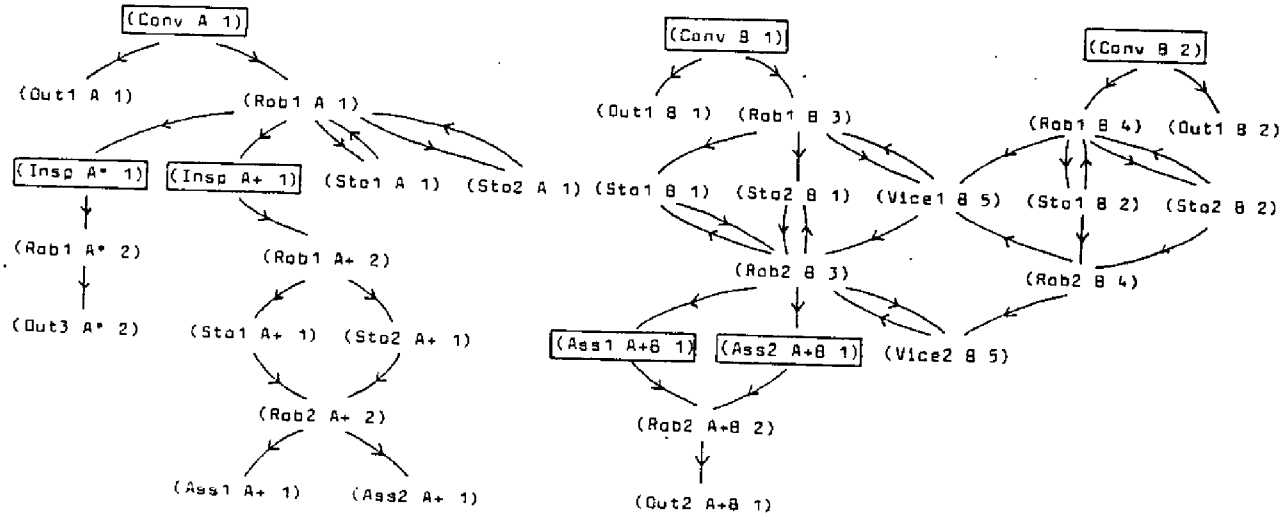
4.2 Application example

Figures 4 illustrates a segment of an application that performs the part-mating of two parts A and B. Parts of type A and B arrive, randomly positioned and in arbitrary order, on a conveyor belt. Parts of type A must be inspected before being assembled with parts of type B.
Two orientations are considered for parts A (<1> for planar surfaces, and <2> in robot grippers), while five possible orientations are defined for part B (<1> and <2> for planar surfaces, <3> and <4> for robot grippers, and <5> for vices). A vision system provides information on the parts delivered by the conveyor belt : their type, orientation and location.

Figure 5 is a representation of the associated application process plan. Parts of type A and B arrive on a conveyor belt (CONV); they are immediatly directed out of the FAC (OUT1) if their introduction may lead to a deadlock state. The inspection operation of parts A results in the creation of parts A+, if it is successful, or of part A*, if the part is defective. Parts A* are put aside (OUT3), whereas parts A+ are stored on the rotary table (STO1 or STO2) before being positioned on ASS1 or ASS2. If parts B arrive on CONV with a "wrong" orientation <2>, they must be turned over - using VICE1 or VICE2 - before being assembled with A+ on ASS1 or ASS2. Parts A+B are delivered by the FAC at OUT2.

A condition on the FAC state is associated to each elementary action, i.e. to each arc in the application process plan. For instance, the condition of the arc (ROB2,B,3)--->(ASS1,A+B,1) is: "site ASS1 contains a part A+ with orientation <1>"; the condition of the arc (STO1,B,1)--->(ROB2,B,3) is: "there is no part in ROB2 and, either one site among ASS1 and ASS2 contains a part A+ with orientation <1> or VICE2 is empty and there is no part B in the FAC that has to be re-oriented (i.e. with orientation <2> or <4>).

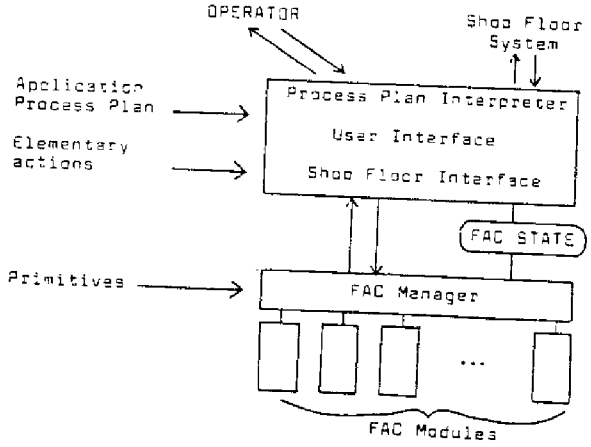
It is worth noting that such a program structure expresses numerous possibilities in a condensed way. For instance, parts B can be re-oriented in various ways : using ROB1 and VICE1, ROB2 and VICE1 or VICE2, parts B can also be momentarily stored before or after being re-oriented. Besides, it allows a great flexibility in dealing with arbitrary parts arrival, and processing several parts simultaneously.



- Figure 5 : Application Process Plan -

4.3 Implementation issues

For a given application, the programming phase provides the on-line system with the process plan, the elementary actions and the primitive functions. The on-line system is composed of several layers (Figure 6).



- Figure 6 : FAC on-line system -

Application process plan interpreter. The first level comprises the application process plan interpreter and other modules such as an interface with higher level management systems, and a user interface.
Upon arrival of one or several parts, the process plan interpreter generates a plan composed of elementary actions that can be performed with respect to the current FAC state. Elementary actions are formulated as sequences of primitive functions that are sent to the FAC Manager. They are scheduled, taking into account FAC components management and concurrent execution (between independant actions, and between operations within steps 1 and 2 for each action).
FAC Manager. The FAC Manager incorporates the FAC model. It executes sequences of primitive functions proceeding from the process plan interpreter and sends back completion results.
It issues commands to various devices and interprets data and completion messages in terms of FAC state changes.
Basic software architecture. The ARA FAC high level software has been implemented using NNS[2], a Lisp-based interactive environment that has been developed to establish a framework for integrating various subsystems into a programmable system.
The NNS system is based on a modular hierarchical implementation, similar to other integrated systems[12,13]. The different subsystems are organized as "Specialized Modules" (SM). A SM consists of all the software and hardware involved in an "activity". SMs can be viewed as entities embodying a set of remote procedures. SMs are implemented as independent tasks running under a multi-tasking operating system or on dedicated processors.

Programming, debugging, and execution are carried out in the same environment, hence facilitating connection between off-line and on-line software systems.

CONCLUSION

This paper has presented an approach to FAC modelling and programming that serves as a basis for the construction of a programming and decisional environment, and for providing FAC on-line system with decision-making capabilities. A key feature is the generation of an application process plan that is interpreted on-line, hence resulting in a great flexibility. Future extensions concern the development of decisional processes that deal with FAC and process state such as failure diagnosis systems, error-recovery procedures, and interface with higher level management systems.

Acknowledgments

We would like to thank Michel Devy and Jean-Michel Valade for numerous hours spent together around the ARA FAC.

REFERENCES

1: **D.A. BOURNE, M.S. FOX,** "Autonomous Manufacturing: Automating the Job-Shop", IEEE computer, sept. 1984

2: **R. ALAMI,** "NNS, a Lisp-based environment for the integration and operating of complex robotics systems", Int. Conf. on Robotics, Atlanta, March 1984.

3: **L. GOUZENES,** "De la perception a l'action en robotique d'assemblage : contribution a la modelisation et a l'algorithmique d'assemblage", These de docteur-ingenieur Universite Paul Sabatier, Toulouse, Nov. 1984.

4: **B. DUFAY, JC LATOMBE,** "An approach to Automatic Robot Programming Based on Inductive Learning", 1st ISRR, Bretton Woods, Sept. 1983.

5: **T. LOZANO-PERES, M.T. MASON, R.H. TAYLOR,** "Automatic Synthesis of Fine-Motion Strategies for Robotics", 1st. ISRR, Bretton Woods, Sept. 1983.

6: **J.M. VALADE,** "Automatic generation of trajectories for assembly tasks", ECAI, Pisa, Sept. 1984.

7: **R.P PAUL,** "Sensors and the Off-line Programming of Robots", '83 ICAR, Tokyo, Sept. 1983.

8: **R.A. BROOKS,** "Symbolic Error Analysis and Robot Planning", IJRR, Vol. 1, Nr 4, Winter 1982.

9: **G. GIRALT,** "Research Trends in Decisional and Multi-sensory Aspects of Third Generation Robots", 2nd ISRR, Kyoto, Aug. 1984.

10: **Troisiemes journees annuelles du Programme ARA,** Robotique Generale, Toulouse, Sept. 1984.

11: **A. GIRAUD,** "Generalized Active Compliance for Part Mating with Assembly Robots", 1st ISRR, Bretton Woods, Sept. 1983.

12: **H. INOUE, M. INABA,** "Hand Eye Coordination in Rope Handling", 1st ISRR, Bretton Woods, Sept. 1983.

13: **A.R. SANDERSON, G. PERRY,** "Sensor-Based Robotic Assembly Systems: Research and Applications in Electronic Manufacturing", Proc. of the IEEE, Vol 71, No 7, July 1983.

NNS, A KNOWLEDGE-BASED ON-LINE SYSTEM FOR AN ASSEMBLY WORKCELL

H. CHOCHON, R. ALAMI

Laboratoire d'Automatique et d'Analyse des Systemes du CNRS
7, Avenue du Colonel Roche
31077 Toulouse CEDEX
France

ABSTRACT

Performing complex assembly tasks requires decisional capabilities available on-line, in order to handle various events (sensor conditions, part arrivals, failure detection...), that occur during execution.

In this paper, we describe NNS, a complete on-line system for multi-robot assembly workcells, that deals with problems of execution monitoring, action scheduling, and failure diagnosis and recovery. NNS is distributed in different decisional layers with their own reasoning methods and state representation; it uses an "execution model" that embeds various knowledge bases built off-line. This approach fits well with requirements of flexibility and efficiency.

1. INTRODUCTION

A Flexible Assembly Cell (FAC) should behave as an autonomous system that is provided workpieces and that performs various physical operations according to some predefined process.

This paper presents an approach to the design of an on-line system for a highly autonomous FAC. First, a brief discussion will focus on the major requirements for such a system. Then, we give a detailed description of NNS, a knowledge-based on-line system that is aimed to be a first step towards meeting flexibility and autonomy requirements.

On-line requirements

In the case of robots working in a well structured environment and performing predefined tasks repeatedly, most decisional problems are addressed off-line. Yet, there remains numerous decisions that must be taken at execution time: opportunistic action scheduling, reacting to unexpected events, specifying an action depending on the execution context. Moreover, an autonomous cell must provide failure diagnosis and user interface facilities for plan repair and error recovery.

Local resource management also plays an important role. In normal operation, several actions requiring common resources (effectors, tools, space regions...) are likely to execute in parallel; the on-line system must be able to distribute cell resources depending on the actions to be performed. There also must be provision for dealing with functional equivalence between several FAC components.

Another key feature is FAC state monitoring using sensors. The on-line system has to detect anomalous events and to react, by generating appropriate emergency actions. Finally, the system must take into account modifications of the FAC state caused by a failure and resume execution.

Considering the above mentioned issues, some design aspects deserve special interest:

* Programming a FAC using a conventional language seems inappropriate as it would be difficult to achieve simultaneously execution and state monitoring, to provide for asynchronous events and to recover after a failure <GIN-85>.

* Such a system has to maintain a FAC state, to reason about it, and to understand action results according to a task-level model.

* The on-line system needs a large amount of information that is supposed to originate from the programming phase. For efficiency reasons the off-line phase must reduce as much as possible the complexity of decisional problems that have to be solved during execution. This is the purpose of various task-level programming systems that are currently investigated <LOZ-84> <LAU-85>. However, such systems must leave sufficient latitude for the on-line phase.

The result will be an amount of pre-processed and condensed information, structured in what we call the "task execution model".

We describe in the following the organization of NNS, an on-line system based on the discussed design issues.

NNS organization

NNS is a knowledge-based system with a hierarchical organization. This hierarchy covers three levels of abstraction: a task level, a functional level and an command level (figure 1).

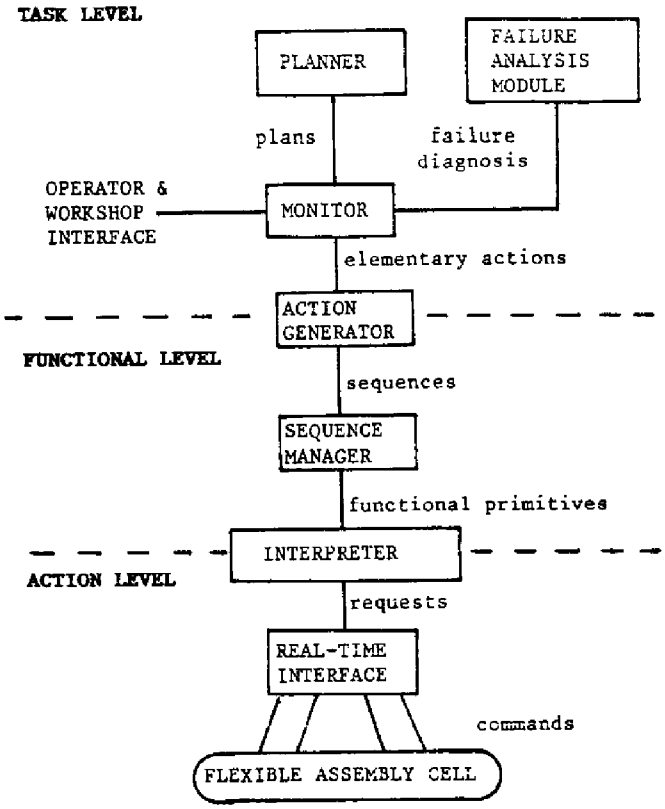


figure 1 : NNS organization

* The task level reasons about what happens in the cell and what the system has to perform. It involves planning, failure analysis, interface with higher level systems (operator, workshop), execution monitoring and action generation. It uses a FAC state representation that will be described in the next section.

* The functional level is in charge of action execution, in terms of functional primitives available in the cell (robot motions, tool operations...). It is composed of two modules: the sequence manager and the interpreter.

* The command level interacts directly with the cell components controllers (manipulators, sensors, various devices ..).

2. FAC STATE REPRESENTATION

A state representation must specify at every moment the situation of a task and allow a prediction of its evolution. All the possible evolutions of the on-line task, are included in a set of **"elementary actions"**.

A complex assembly task performs physical relationships among objects in a known environment; it also gets information about parts, and communicates with the external environment (operator, workshop level). We define the following elementary actions:

- **PICKING** or **PLACING** a part somewhere in th workspace: such actions modify the equilibrium o a part, with the creation of new relationship between the part and the environment,
- **PART MATING** or **DISASSEMBLING**, i.e. creatir or destroying physical links among severa workpieces.
- **GETTING INFORMATION** about parts, such a identification, location or inspection ...,
- **FEEDING** or **UNLOADING**: these operation involve communication with a workshop level asking for new parts or signalling that parts ar ready to be unloaded. The workshop level i responsible for conveying parts to and from th cell.

Elementary actions are considered as stat operators by the planner: the plan does no specify how elementary actions will be performe (in particular, manipulator motions are no addressed in the plan), but it sets up scheduling which represents a projection of th FAC state. The generation of elementary action takes into account opportunities and informatic provided by the execution context.

It is worth noting that the classical PICK&PLAC operation is considered in our representation a two distinct elementary actions, so that ever change in the FAC state is associated to only on modification of physical relationships amon objects. Thus, the system is able to restart fro any state.

Our FAC state definition provides the followin information, that are more detailed in <ALA-85>:

- Where are the parts in the cell? Answerin this question implies that the cell is organize into a finite number of **SITES**. A site is a plac in the cell intended to support workpieces; site may be equipped with effectors, tools, sensors, c specialized devices...

- Where are the sites in the cell? Th workspace is discretized into **REGIONS**, in order t specify the type of effector motions that can b performed within.

- What are the part configurations? To specif the physical relationships of a workpiece in site, we need the three following parameters:

--> the **POSTURE**, i.e. a representation c geometric and dynamic constraints that achieve stable position. The legal postures of eac part are determined off-line, and the way th are achieved is specified in the execution model

--> the **LOCATION** of a part in the site, i.e. t transform between a reference frame associated the site and the reference frame of the part.

--> the **ACCURACY** of part location in the sit This accuracy results from the entire pa followed by the part. Indeed, every elementa action modifies the uncertainty about pa location, according to a model obtained off-lin

group of appropriate concurrent actions, and determines the resulting FAC state for the next step. Let us specify the implemented heuristics that are successively applied for conflict resolution, illustrating our discussion with an example.

First we use a graph representation of the application process (figure 3), which is part of the execution model. The graph specifies all the possible paths of each part in the cell. The nodes are part states (site, part, posture), and the arcs represent elementary actions. Arcs are labelled with conditions about certain components of the FAC state: parts present in the sites, the posture, the location and location accuracy of parts. The conditions are evaluated in order to determine if elementary actions are or are not appropriate. Conditions are used to assure that actions are **feasible** (e.g. Is the accuracy sufficient to guarantee execution success? Is the grasp achievable according to part posture and location?), and **relevant** (e.g. Will the action contribute to bring a part nearer to its target? Will the action free a critical site, i.e. a site that is a mandatory crossing point of some part?). Using the graph partially represented on figure 3, the set of appropriate actions from the initial state, ((in B 2)(p B 1)(vi B 5)(a1 A 4)(a2 A 4)), would be 5, 13 and 14.

Then, the planner groups appropriate actions that may be performed in parallel. To do so, it examines available resources they need (effectors, tools, space regions...). Note that the plan does not impose a concurrent execution of these actions, but it uses this heuristic for a better resource management. In the example, groups of appropriate actions are (5 , 13) or (5 , 14).

At this stage, conflicts may remain and the choice of a group of actions is not easy to make.

An estimation of the durations of actions could be taken into account, but this method requires specifying all the timing and it implies a search with backtracking. Moreover it seems to be sensitive to variations between estimated durations and real execution times.

Another possibility is to defer the decision until plan execution. This opportunistic approach implies generating a conditionnal plan. It may entail a combinatory explosion of the plan, and it does not always provide the best optimization.

As a first approach, the planner selects one of the groups that includes the greatest number of actions, but conflicts are not always entirely solved, as in the example: the group selected is (5 ,13), and after several steps, we obtain the following results:

step 1: **(5 , 13)**
---> ((r1 B 3) (r2 B 6) (vi B 5) (a1 A 4) (a2 A 4) (in))
step 2: **(9 , 17 , 1)**
---> ((p B 1) (a1 AB 1) (vi B 5) (a2 A 4) (in ?))
step 3: **(14 , 2)**
---> ((p B 1) (a1 AB 1) (r2 B 6) (a2 A 4))
step 4: **(18)**
---> ((p B 1) (a1 AB 1) (a2 AB 1))

The goal of the plan is reached when no more actions are appropriate or when the planner decides to postpone the remaining steps to a next planning.

The plan is then provided to the monitor: it specifies the order of elementary actions related to each site. In the example, the plan generated would be:

site	I	actions
in	I	5, 1, 2, planning
r1	I	5, 9
p	I	13, 9
s	I	
vi	I	14
r2	I	13, 17, 14, 18
a1	I	17
a2	I	18

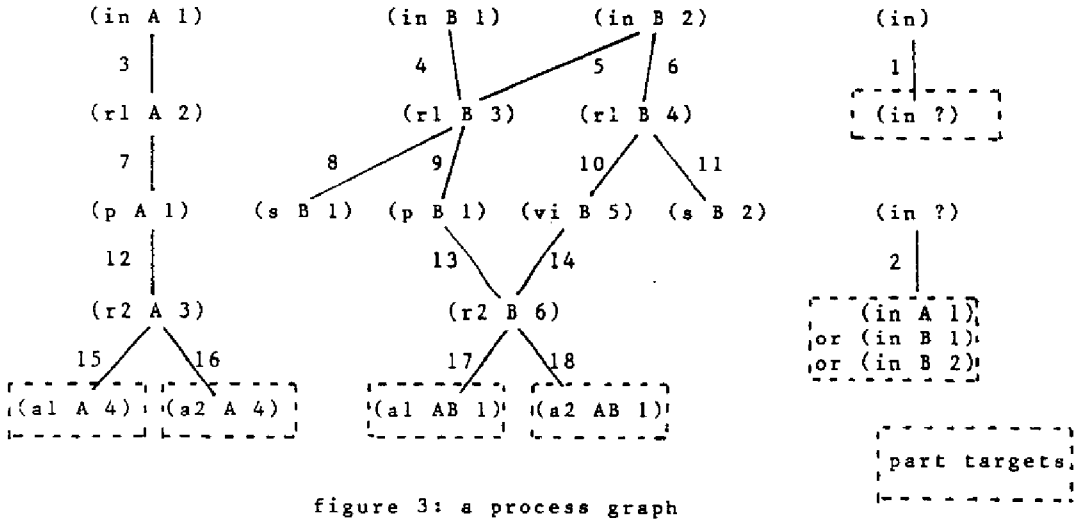


figure 3: a process graph

The graph gives all its efficiency to the planner; it expresses in a concise manner all the potentialities of the cell. Methods for graph analysis are studied in order to establish off-line the conditions associated to each arc.

3.2 The monitor

The monitor coordinates the execution, planning and failure analysis activities. It knows about the past, the present and the near future of the task: it maintains information about the history of parts present in the cell, the elementary action in progress and the next actions to be performed on each site.

The monitor asks for a new plan in some particular situations that have been examined above (part arrival, unexpected events, end of the previous plan).
It provides a "valid" initial state for planning, i.e. a state completely specified and that belongs to a set of acceptable states. Indeed, after a failure detection and analysis, the FAC state may become partially undetermined or it may lead to a deadlock situation: the monitor has then recourse to an operator intervention in order to start again (e.g., removing an unknown part from a site where no camera is available to identify it).
Generally, the initial state provided to the planner does not agree with the current FAC state. It is an impending state that assumes a successful completion of ongoing actions; thus, planning times are partially hidden.

The monitor interacts with a workshop level, exchanging signals (part arrival or part ready to be unloaded), and information (asking for missing parts, notifying its ability to process the task...).

The monitor is informed of the end of elementary actions: it updates a task execution history and uses the plan to determine what are the next actions that can be started.

When the real-time system detects a mismatch between the FAC state managed by NNS, and the actual state monitored by sensors, it generates some predefined emergency actions and issues an interrupt to the monitor. The monitor reacts first by interpreting the interrupt, by determining the sites that were present in the region where the unexpected event occured, and by cancelling all ongoing actions that are likely to be affected by the event. For instance, if a part, held in a vice, falls down as a robot was about to grasp it, the picking action is aborted. Then, the monitor sends to the failure analysis module circumstances about the event: current FAC state, sensory conditions that were monitored, sensor values, elementary actions in progress, history of the execution.

3.3 The failure analysis module

In the NNS system, the purpose of the failure analysis module is precisely defined as to establish the FAC state which results from a failure. Note that this module is not in charge o the emergency actions following a failure, nor o execution resumption.

After the occurence of an anomalous event, th validity of the FAC state will be considere doubtful; the failure analysis module will use th execution context and the task history provided b the monitor, to establish a diagnosis of possibl causes. Sometimes, the diagnosis implies mandatory modification of the task execution mode (e.g. a new estimation of the accuracy require for an action). The failure analysis module wil then use its hypotheses to modify the set o available resources (e.g. device, sensor o effector malfunctioning), and to update the FA state (e.g. to free a site which has released part).

The failure analysis module is a key element o sophisticated on-line systems. It raises man design problems as it needs a large knowledge o the task, on the cell functions, and on assembl errors... We are developing research on such topic. In the current implementation, there is n true diagnosis. We use a very simple approach which consists in invalidating the state of al sites that were present in the space region wher the event occured. Then the system makes use o sensors to reacquire information and asks th operator for missing information. Even with thi simple implementation, the system requires minimu operator intervention for execution resumption.

3.4 The action generator

The action generator embeds the transition betwee the task level and the functional level. I determines how an elementary action can b performed.

A set of intermediate objectives is associated t each type of elementary action (pick, place identification...). Action generation consists i instanciating these objectives and building th appropriate sequences of FAC functiona primitives.

For instance, the PICK or PLACE elementary actior involve the following objectives:

-a- bringing two sites in the same region, wher the whole action will be performed. This ste consists in manipulators gross motions and othe effectors positioning (rotary table, conveyor).

-b- preparing tools in order to receive t part, according to its next posture (opening gripper or a vice).

-c- approaching in order to achieve t geometrical relationships involved in the ne state of the part.

-d- achieving the state transition, by applyi new forces (closing a gripper) and then releasi the part (opening a vice). The FAC state and t associated sensory monitoring conditions a updated at this step.

-e- Moving sites apart, so that gross motions for a next action can be performed.

The system exploits the potential parallelism in steps -a- and -b-, and produces several independant sequences. On the other hand, steps -c- -d- and -e- are strictly ordered and performed in a unique sequence, synchronized with the end of the other sequences.

Instanciating these steps takes advantage of some information given by the execution context. For instance, the following decisions, using predefined heuristics, have to be taken:

- Choosing a region to perform the action: for example, picking a part on a rotary table requires to determine the positioning of this effector.
- Selecting a path for gross motions, that avoids to cross a region currently locked by another action.
- Choosing among a number of predefined fine-motion strategies, according to the accuracy of the part location.

Sequence generation consists in selecting functional primitives, specifying their parameters and determining resources that are required for their execution. Note that information provided by the task execution model is of critical importance for sequence generation. Sequences cannot be entirely generated off-line; all complex decisionnal problems are addressed off-line (space structuring <GOU-84>, fine motion generation <VAL-85>) and results are provided to the action generator module in the task execution model.

3.5 The sequence manager

The sequence manager provides a multi-tasking environment that allows to run sequences in parallel.

Sequences consist of functional primitives, requests to update the FAC state, and resource management operations. Indeed, the planner cannot deal with cell resources that are only necessary to achieve an intermediate step of an action (space regions to be crossed, effectors linked to several sites...).
Likewise, FAC state is updated inside the sequences (not after the completion of elementary actions), in order to reflect more accuratly the actual state and to change the monitoring of sensory conditions that are issued to the real-time system.

Sequences may be suspended after a call to a functional primitive, waiting for the receipt of a completion status; any error will cause the whole sequence to be abandoned.

Sequences can also be aborted when the monitor orders it.

3.6 The interpreter

It is composed of all functional primitives, such as effector motions, part identification and localization, state checking and monitoring and other task-specific functions (complex assembly primitives, screwing operations...).

These primitives establish a link between the cell's functions and their actual implementation: primitives will be translated into execution requests that are submitted to the real time interface. In the opposite direction, data proceeding from the real-time system will be interpreted in terms of the primitive's execution status or in terms of events that will be addressed to the monitor.

3.7 The real-time interface

This layer contains all implementation-dependent information. It accepts execution requests from the interpreter and issues commands to the physical FAC components (robot controllers, vision systems, peripheral devices ..).

Requests are of three types: action, sensing and monitoring.

- **Action requests** involve robot movements, gripper opening... Such actions may be interrupted if some monitored sensory condition is verified. For instance, the motion of a robot approaching a part must be stopped, if a sensor detects that the part has been displaced. Note that, as the example illustrates, there are numerous situations where an event not directly linked to an ongoing action, may have consequences on the feasability of this action.

- **Sensing requests** involve acquisition of information from sensors: obtaining the identity and the location of a workpiece... Note that an other use of sensors is embedded in monitoring requests.

- **Monitoring requests** consist of the activation and cancellation of predefined rules, intended to monitor the FAC state or to perform simple guarded commands. A monitoring rule contains a condition on the value of some sensors, which will be evaluated periodically, and a set of immediate actions that must be executed when the condition is verified.

4. IMPLEMENTATION ISSUES

NNS is implemented as a set of processes running on various processors. Communication is performed mainly by interprocess messages.
The system architecture is based on a hierarchical structure composed of various software layers.

The low level - termed "real-time level" - is composed of **"specialized modules" (SM)**. Each module controls one of the main cell components (robots, vision systems, peripheral devices...) and is implemented as an independant low level interpreter. SMs may access to a common memory, which contains the current status of the cell (ongoing requests, immediate actions to be taken when exceptions are detected, processes state ..).

The second level is implemented in LISP. The Lisp system we use provides facilities for exchanging messages with other processes and for having access to the cell common memory. Likewise, we use a multi-tasking package that allows to run, within one Lisp system, multiple processes in a pseudo-parallelism mode. This pseudo-parallelism produces a true parallelism for physical actions; indeed, execution requests are sent to SMs running on dedicated processors. The sequence manager makes use of these mechanisms: every generated sequence is associated to an independant Lisp process.
Higher level layers (monitor, planner...) are also implemented in LISP.

The hardware architecture is based on a mini-computer linked to various microprocessors and specialized processors. The SMs currently installed are two 6 d-o-f manipulators, an industrial vision system, an experimental multilevel vision system <DEV-85>, a system for monitoring sensory conditions <LOP-86>. This software runs on an independent processor and controls various sensors (microswitches, potentiometers, infrared cells, ultrasonic sensors...) and peripherals (rotary table, vices...). It dynamically accepts rules, composed of a sensory condition that are checked at high rate and of some associated actions to perform when the condition is verified.

A first version of NNS has been fully implemented and used for several experimental assembly tasks. We have obtained some reasonably fast performances for the execution of complex assembly tasks (four types of parts arriving in arbitrary order on a conveyor with different postures and manipulated with two robots): the decision-making time was entirely hidden by the execution time of action requests. Moreover, the system proves to be of flexible use: debugging and simulating facilities have been integrated within NNS.

CONCLUSION

We have examined design issues of on-line systems for Flexible Assembly Cells and shown how flexibility and autonomy are considerably improved by introducing some decisional capabilities during the execution phase.

Our approach is based on a multilevel organized system, provided with a "task execution model". The main decisional activities are involved in planning for action scheduling, failure diagnosis and action generation.

It is worth noting that this approach can be viewed as a method for building complex assembly tasks, because it clearly distinguishes between what should be prepared off-line (programming and learning phases) and what should be left to on-line decision making.

Finally, we consider it as a suitable specification for the output of a task-level programming system, that we are currently investigating .

REFERENCES:

<ALA-85> - R. ALAMI, H. CHOCHON, "Programming of Flexible Assembly Cells: Task modeling and system intergation", IEEE Conf. on Robotics, St. Louis, April 1985.

<DEV-85> - M. DEVY, "Identification d'objets plans par recherche de caracteristiques locales et globales", COGNITIVA 85, Paris, June 1985.

<FOX-85> - B. R. FOX, K. G. KEMPF, "A representation for opportunistic scheduling", 3rd ISRR, Paris, October 1985.

<GIN-85> - M. GINI and al, "The role of Knowledge on the Architecture of a Robot Robust Control", IEEE Conf. on Robotics, St Louis, April 1985.

<GOU-84> - L. GOUZENES, "Strategies for Solving collision-free trajectories problems for mobile or manipulator robots", Int Journal of Robotics Research, Vol 3, No 4, Winter 1984.

<LAU-85> - C. LAUGIER, J. PERTIN-TROCCAZ, "SHARP A system for Automatic Programming of Manipulation Robots", 3rd ISSR, Paris October 1985.

<LOP-86> - E. LOPEZ MELLADO, "Un systeme de surveillance en Temps Reel pour les systemes d'Assemblage Robotise", LAA Internal Report, February 1986.

<LOZ-84> - T. LOZANO-PEREZ, R. BROOKS, "A approach to Automatic Robot Programming", USA-France Robotic Workshop, Philadelphia, November 1984.

<VAL-85> - J. M. VALADE, "Geometric Reasoning and Synthesis of Assembly Trajectory", 8 ICAR, Japan, September 1985.

INT. J. PROD. RES., 1990, VOL. 28, NO. 6, 1195–1213

The analysis of repetitive sequencing at the workcell level: from workcell tasks to workcell cycles

PAUL FREEDMAN†‡ and RACHID ALAMI‡

We begin with the part-oriented definition of a workcell *task* in terms of generic *elementary actions* to be performed. These actions are then mapped to a workcell-element-oriented description of the *operations* to be executed to perform the task. When the workcell task is *repetitive*, the notion of a workcell *cycle* becomes important, and we can associate with each workcell element a repetitive *sequence* of operations. In this paper, we shall demonstrate that workcell programming is more than just a matter of sequencing, since there may exist alternative cycles inherent in the same specification of the workcell tasks. These alternatives are then compared in terms of cycle time, task throughput, task-processing rate, and machine utilization. But even for simple workcell tasks, the space of possible time evolutions of workcell behaviour can be very large, with many different workcell cycles. To automate such sequence analysis, we therefore present a Prolog-based decision-support system called SAGE (sequence analysis by generalized enumeration). Three simple examples are used to illustrate our approach.

1. Introduction

According to Wemmerlöv and Hyer (1987), there are two aspects to the problem of workcell-level design:

(1) issues related to *system structure*, such as the selection of workcell elements (robots, computer vision systems, conveyors, etc.), their physical layout, and the definition of their programming primitives; and
(2) issues related to *system operation*, such as the definition of the workcell tasks, and the sequencing of these tasks.

In this paper, we shall concentrate on the second aspect. In particular, when the workcell must function in a *repetitive* way, the notion of a workcell *cycle* becomes important, and we can associate with each workcell element a repetitive sequence of *operations*. (Our use of terminology is given in the next section.) The importance of repetitive behaviour has been emphasized by Hall (1988): 'the key is to develop as repetitive a process as possible in any kind of manufacturing' (p. 458). This has two important consequencies. First, operational aspects are easier to analyse and, therefore, easier to improve. Second, the balancing of production flows at the factory level is easier to perform, and a 'lead' (master) schedule becomes easier to construct. Then 'the goal is not only to complete units to ship on schedule, but to also maintain a steady cycle time so that all of the operations keying from a lead schedule can stay together' (p. 466).

Received 5 May 1989.
† Partially supported by a NATO Postdoctoral Fellowship provided by the Canadian government.
‡ Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), 7, Avenue du Colonel Roche, 31077 Toulouse Cedex, France.

3.1

For our purposes, sequence analysis begins once the workcell operations are defined. Thereafter, at least two kinds of optimization related to system operation can be studied. First, if there is some functional redundancy among the workcell elements, the workcell may be capable of performing the same task in different ways, e.g. by using a robot instead of a conveyor system to load/unload parts. Then if the durations of these 'equivalent' operations are different, we will obtain different repetitive behaviours.

The second kind of optimization is more fundamental. Given the definition of a task as a set of workcell operations, there may be several ways of performing the task, i.e. there may exist several different partial orders of operations for the workcell elements. Since the workcell operations are typically strongly coupled, the partial order selected for a given workcell element will strongly influence workcell-level behaviour. In this context, the selection of the partial order for each workcell element which minimizes the time to perform the task could be important (Giralt 1984).

It is this second kind of optimization which is missing from current workcell *programming* systems. Indeed, the constraints associated with precedence and shared resources which link the workcell operations make it difficult to analyse workcell behaviour in a manual way. According to a recent study (Meredith 1987), programming (or perhaps one should say 're-programming') difficulties at the workcell level severely limit the flexibility of so-called flexible manufacturing. Indeed, 'software is the major technical problem during implementation... [but] the implementation process really never stops' (p. 1502).

Therefore, it seems evident that programming *aids* at the workcell level are of critical importance. In particular, the problem of *constructing* a sequence of operations for each workcell element which respects the constraints associated with precedence and shared resources is computationally intractable (Garey and Johnson 1979). Informally, this implies that there exists no known efficient algorithm for generating a solution. Therefore, problems of this nature are ordinarily solved by searching the space of possible solutions, possibly with the aid of application-dependent heuristics.

To this end, we note the increasing popularity of logic programming in the management science literature (Bharath 1986, Lee and Miller 1986). Unlike conventional programming which emphasizes an algorithmic or prescriptive approach, programming in logic, e.g. using the programming language Prolog (Clocksin and Mellish 1981), has a *descriptive* feel because a program consists of relationships among values. Each definition (rule) defines a relationship, and each clause in the definition specifies a condition of *pattern* for which the relationship holds (Davis 1985).

Predicate logic and the mechanical theorem proving of Prolog together provide a powerful framework for solving planning problems in a manufacturing environment. One example is a knowledge-based system for the 'operational control' of a production facility (Ben-Arieh, Moodie and Nof 1985). Routing decisions of parts between machines are derived in real time from policies encoded as decision rules in Prolog.

A Prolog system called MASCOT has been presented by Erschler and Esquirol (1986) to schedule operations on machines in a job shop. Each workcell operation is described by duration, earliest start time, latest finish time, resources required and the amounts consumed. The system uses the objectives associated with resources to construct a partial order which characterizes feasible solutions. Bensana, Bel and Dubois (1988) have described a planning system called OPAL based on MASCOT

3.2

which generates a job-shop schedule from the partial order using heuristic knowledge supplied by the shop-floor manager about 'technological constraints'.

Finally, a knowledge-based planning system has been presented by Shaw (1988) to dynamically schedule a set of tasks (jobs) for a robotics workcell. The state of the workcell is represented in a symbolic way via a 'world model'. Each workcell operation is characterized by a duration and the set of required resources; pre-conditions and changes (additions, deletions) to the workcell are defined in terms of the world model. Tasks are then defined as sets of required operations. At first, each task is planned separately and then precedence constraints are 'added' to manage the mutual exclusion associated with shared resources.

However, these systems are best described as production (or process) *planners* since *goals* are associated with the scheduling problem. Indeed, they emphasize the resolution of different kinds of scheduling requirements (which may be conflicting) at different levels in the factory.

In this paper, we present SAGE, (sequence analysis by generalized enumeration) an application-*independent* Prolog-based decision-support system for workcell sequencing analysis (Freedman and Malowany 1988a). Given the definition of the operations required to perform the intended workcell tasks, SAGE first constructs a Timed Petri net (Ramchandani 1974) to verify that the workcell program has a repetitive form. Alternative repetitive workcell behaviours are then found by searching the space of possible time evolutions of the workcell. Since the workcell operations are strongly coupled, this space is of manageable size, even for medium-sized applications. (The use of Petri nets in flexible manufacturing has been reviewed by Barad and Sipper (1988).

In contrast to the AI-oriented scheduling systems described above, the work to be presented here has a completely different emphasis, since we view sequence optimization as part of the *off-line* development of the workcell program, rather than as part of the runtime control. For example, we make no attempt to 'resolve' runtime conflict; rather, we simply explore the consequences of resolving such conflict in different ways. In this sense, SAGE is designed simply to help the workcell programmer evaluate alternative runtime 'strategies'. Note too that SAGE performs no 'planning' in the sense that there are no goals to be achieved (except the obvious one that the intended workcell task must be performed).

To the best of our knowledge, the only other research dealing with time in the context of flexible manufacturing is that on FORBIN (Miller, Firby and Dean 1985). Here the workcell consists of a single mobile robot tending several CNC machines, and the set of travel times between machines is provided to the system. Since the coupling among the workcell operations is restricted to relations between the robot and the CNC machines, the planner seeks to optimize the robot motions by assigning to each task the time interval which results in a plan of minimal execution time. The resulting search space is factorial in the number of tasks, and heuristics (e.g., deadlines on the robot motions) are invoked to make the analysis possible; thus, the 'best' plan found may be sub-optimal. Note too that parallel streams of operations are not treated.

In the next section, we present a part-oriented programming paradigm along with the required terminology. Next, a classification of workcell applications in terms of runtime uncertainty, or *non-determinism* (Freedman 1989) is developed to understand better what repetition means at the workcell level. Then a characterization of workcell cycles is described to compare alternative cycles, based on workcell time, task throughput, and machine utilization. Finally, the sequencing analysis is illustrated using three small examples of increasing complexity.

3.3

Work with SAGE was begun by the first author at McGill University (Freedman and Malowany 1988 a, Freedman and Malowany 1988 b), and is now part of continuing research at LAAS with the workcell programming environment NNS (Alami 1984, Alami and Chochon 1986, 1988).

2. Assembly workcell programming

In the context of automated assembly, a workcell *task* is a part-oriented description of the time evolution of a part of part set as it is 'processed' by the workcell. The flow of control is implicit here; the task takes the form of a set of part *states* and state transitions.

A *workcell program* is a set of tasks to be concurrently performed. For example, the workcell might be programmed to produce multiple sub-assemblies, or perhaps perform distinct inspection and repair tasks on the same part set. In what follows, we shall sometimes refer to the execution of a program task as an 'instance' of that task.

Associated with each state transition is an *elementary action* which describes a modification of the part state. Thus, the set of elementary actions of a given task characterize how the task is to be performed by the workcell. In this sense, the set of elementary actions are the functional primitives of the workcell. Elementary actions may be classed as either 'deterministic', when there is a single outcome, e.g. a robot motion, or 'non-deterministic' when there are multiple outcomes, e.g. the inspection of an assembly with outcomes {pass, fail}. (It is at this level that treatment or errors is best introduced, by associating error-related outcomes with the elementary actions and by defining extra elementary actions for error handling.)

Finally, we associate with each workcell element a set of *operations*. Each operation describes a modification of the workcell state (not to be confused with part state) in terms of changes in the workcell resources. Each operation takes the form of a set of enabling pre-conditions, an activity (command) to be executed, a set of post-conditions, and the *duration* of the activity. Thus, the set of all workcell operations defines the execution primitives of the workcell, i.e. 'what' the workcell can do. In general, an elementary action may be realized by several different operations (or sequences of operations).

We shall further classify the workcell operations as follows. Those operations which figure in the definition of the workcell task are called *required*; they are associated with transformations of the parts (e.g. loading, unloading, inspection and mating). The other operations are called *auxiliary*; they are entirely associated with motions (of the robots, conveyor systems, etc.) within the workcell. Such operations may or may not be executed as part of the workcell task.

Since we shall be concerned with time minimization in this paper, we shall assume that the duration of an operation is independent of its execution context, i.e. independent of the operation previously executed, independent of the operation to be executed next, and independent of the other operations to be executed in parallel. (If this assumption is invalid, then the granularity of the operation is too coarse and the single activity associated with the operation must be decomposed.)

For example, the assembly task 'mate parts A and B' might have the following description:

$0 \rightarrow$(A, new), i.e. a new part of type A is in the workcell
$0 \rightarrow$(B, new)
(A, new)$\rightarrow$(A, aligned)

3.4

(B, new)→(B, aligned)
(A, aligned) and (B, aligned)→(AB, new) i.e. the aligned parts are mated and a new assembly is obtained
(AB, new)→(AB, +), i.e. the assembly was correctly performed
(AB, new)→(AB, −), i.e. the assembly was incorrectly performed
(AB, +)→store result
(AB, −)→scrap result

Each 'step' is an elementary action; note that the operations required to perform the elementary action are not specified. For example, parts may enter the workcell as the result of different workcell operations, e.g. a robot motion, or the motion of a conveyor system. The inspection of the assembly might be performed by a computer vision system in a single image-processing step, or perhaps in several steps. Indeed, the set of actions required is not unique when the workcell exhibits functional redundancy. We will return to this issue shortly.

A key feature of the NNS workcell programming environment is the runtime representation of the workcell program in the form of a condition/event graph called the *execution graph*. Nodes are the part states, and the arcs are the operations required to perform the desired workcell tasks. To perform the sequencing analysis based on time, the execution graph is re-cast as a timed Petri net so that the durations of the operations can also be modelled.

3. Workcell tasks: a classification

Clearly, repetition implies the absence of runtime uncertainty or *non-determinism*. This non-determinism can take two forms. The simplest kind of workcell task can be described as *deterministic* when: (i) the outcome of each of the associated workcell operations can be completely predicted; and (ii) the operations follow one another in a fixed way. For example, the task of a CNC machine is nominally deterministic. Since the possibility of failure at each machining step is not considered, programming is a matter of sequencing subject to the precedence constraints on the machining steps. But even when all the workcell operations have predictable outcomes, there can arise instances when mutually-exclusive operations become possible to perform at the same time. For example, consider a single robot responsible for parts transfer between several CNC machines. If multiple machines finish their required operations simultaneously, then when the robot is free, multiple part transfer operations are possible. Such conflict can be resolved by examining the consequences of the alternative robot operations so as to optimize the usage of the CNC machines. This might mean first tending to the machine with the operation of longest duration.

Therefore, we will say that a workcell task exhibits *internal non-determinism* when: (*a*) the operations have predictable outcomes; and (*b*) there arise instances when mutually-exclusive operations become enabled at the same time (due to resource sharing). Note that here, non-determinism takes the form of runtime conflict; we use the word 'internal' to emphasize that it is embedded in the description of the task itself. Each instance of internal non-determinism must be resolved by examining the consequences of the alternative operations. In this sense, programming means respecting precedence constraints *and* optimizing the usage of shared resources.

But when we move beyond the definition of operations to consider the workcell environment, another kind of runtime uncertainty becomes apparent. The workpieces

3.5

might not be uniform, and they might arrive at the workcell with different orientations, in different orders, etc. Of equal importance, the operations are subject to failure, due to progressive wear/failure of robotic components, and to unexpected human interference.

In general, this uncertainty describes non-determinism associated with the *outcome* of a robotic operation. Such *external non-determinism*, i.e. non-determinism external to the user description of the workcell task, can be handled in the following way. For example, consider a workcell configured to perform the repair of printed circuit boards with varying defects. A vision system might be used to inspect each board to determine the nature of the defect (and the kind of repair to be performed). The visual inspection could be modelled as a single sensing operation with alternative mutually exclusive outcomes {no_defect, defect_type1, ...} 'competing' as the consequence of the sensing. Similarly, the possibility of failure could be modelled by adding an extra outcome to each operation to represent failure (along with extra operations associated with the recovery mechanism). Of course, this approach requires that all the outcomes associated with the external non-determinism can be anticipated and suitably defined.

Stated in another way, internal and external non-determinism represent, respectively, runtime uncertainty associated with conflict among mutually exclusive operations, and runtime uncertainty associated with on-line events related to the robotic environment. In the balance of this paper, we shall restrict our attention to workcell tasks which exhibit internal non-determinism. (See Freedman (1989) for the treatment of external non-determinism.)

Given this classification of workcell tasks, we still require a means of *comparing* alternative sequences (and, therefore, schedules). This is the topic of the next section.

4. Comparing repetitive behaviours

One common metric used to compare sequences (or more properly, schedules derived from sequences) in the job-shop literature is the *completion time* of each job (Baker 1974). In this sense, we say that a schedule S *dominates* another schedule S' if for each job j_i, the completion time $c_i(S) \leqslant c_i(S')$, i.e. each job completes sooner or no later in S than in S'.

In the context of repetitive manufacturing, we can replace the 'time to complete' by the 'time between successive completions'. And when the workcell operations are strongly coupled, we will expect to find that the time between successive completions of different workcell operations will be the same.. We shall call the period of repetitive behaviour the workcell *cycle time*.

Using our terminology, *makespan* is the total time required to execute a task. When the workcell program consists of just one task, then makespan = cycle time. But since a typical workcell program consists of several tasks, the cycle time $\geqslant$ the makespan of each task since for any given task, the cycle time may include task 'idle' time due to the inter-leaving of the different tasks.

But time, by itself, tells just part of the story. To compare different repetitive sequencies more carefully, we introduce two additional metrics: the task throughput, i.e. the number of times each task in the workcell program is executed per cycle; and machine utilization, i.e. the working time of the workcell elements as a percentage of the cycle time.

3.6

The operational information described by these metrics might be used in the following ways.

(1) As described by Hall (1988), cycle time is key to the simplified analysis and improvement of operational aspects, and to the balancing of production flows at the factory level.
(2) Task throughput can be used to determine the appropriate amounts and mixes of parts for the input and output pallets associated with different tasks. In a sense, this represents a fine-tuning of the resource allocation performed at the factory level. If each task in the workcell program involves the assembly of a different part set, then the processing *rate* of the task is simply the task throughput divided by the cycle time. For example, if $task_i$ is executed n_i times during a workcell cycle with cycle π, then its processing rate would be n_i/π.
(3) Machine utilization time can be used in two ways: to help select the operations required to perform the tasks by identifying workcell elements which are under-used, and to help formulate policies for periodic inspection and maintenance, as suggested in (Wemmerlöv and Hyer 1987).

As we shall see, the three metrics are *not* simply related when the workcell program consists of multiple tasks. Such a situation is typical of automated assembly applications.

We have demonstrated previously (Freedman and Malowany 1988 b) that a workcell task which is 'meaningful' in the sense of 'repetitive' will always have a finite state space. A simple cycle† in this space therefore represents one possible form of repetitive behaviour. Since the space is finite, all simple cycles must have finite length. (At worst, a cycle will contain every node.)

To simplify the enumeration of this state space, we introduced a simple heuristic: as soon as a workcell element completes an operation, assign to it a new operation from among the operations now enabled. At the time, we recognized that this incremental or 'local' minimization of machine idle time may not be optimal in the 'global' sense, i.e. optimal in terms of the workcell cycle thus obtained (as shown by Carlier and Chrétienne (1985) for the general case of timed Petri nets). But for workcell applications dominated by synchronization, the heuristic greatly reduces the complexity of the sequence analysis at little 'cost'.

When this heuristic is suppressed, a sense of *forced waiting* is added to our analysis. We shall say that a workcell element is 'forced to wait' if it is now idle and at least one of its operations to be performed is now enabled. According to the classification of workcell tasks presented above, forced waiting is therefore a kind of *artificial* non-determinism associated with the addition of 'do nothing' operations of varying duration. Clearly, the addition of forced waiting increases the size of the state space of workcell behaviour, and adds new cycles to the sequence analysis.

At this point, the following question comes to mind: Depending upon the mix of durations of the workcell operations, does there exist a workcell cycle of smaller cycle time than can be found using the heuristic to locally minimize machine idle time? As we shall see, the answer to the question is 'sometimes'. Indeed, the answer critically depends upon the structure of the workcell application, and upon the mix of durations of the workcell operations. But even when the answer is 'no', a sequence with forced

† A cycle is simple if each arc in the cycle is traversed just once.

3.7

waiting might be preferred, since the workcell throughput, or the distribution of machine utilizations might be somehow preferred at the factory level. Indeed, Hall (1988) noted that when the cycle time of a workcell must be reduced in order to balance production flows at the factory level, but the processing rate of a particular machine cannot be varied, that machine must lie idle for part of the workcell cycle.

We can therefore pose a second question as follows: Depending upon the mix of durations of the workcell operations, does there exist a workcell cycle with a superior processing rate for some task that can be found using the heuristic to locally minimize machine idle time? In the following section we present various examples to make this clear. The first example involves a single robot tending two CNC machines, while the second and third concern two robots performing co-operative assembly.

5. A first example

Consider a simple workcell consisting of two CNC machines and a single robot. The machines work independently on different tasks, and on different part types A, B. The robot is responsible for loading new parts and unloading finished parts at each machine. This leads to the following task descriptions.

Task A
0→(A, new)
(A, new)→(A, machined)
(A, machined)→0

Task B
0→(B, new)
(B, new)→(B, machined)
(B, machined)→0

Using these elementary actions, we now define the workcell operations using the notation op_{ij} for the jth operation of the ith workcell element, where element number 1 is CNC 1, element number 2 is CNC 2, and element number 3 is the robot:

op_{11}: (A, new)→(A, machined)
op_{21}: (B, new)→(B, machined)
op_{31}: 0→(A, new) and (A, machined)→0
op_{32}: 0→(B, new) and (B, machined)→0

with the following durations: $\tau_{11} = 3$ s, $\tau_{21} = 1$ s, $\tau_{31} = 2$ s and $\tau_{32} = 1$ s.

Using the heuristic of local minimization of machine idle time, a single repetitive sequence is found, as shown in Fig. 1 in the form of a Gantt chart. Note that operation op_{11} appears twice in the Gantt chart although it is executed exactly once (in two parts) during the cycle.

Since operations op_{11} and op_{31} are each executed just once per cycle, the throughput of task A is just 1. However, since operations op_{21} and op_{32} are each executed twice per cycle, the throughput of task B is 2. This result can be generalized as follows.

Theorem 5.1: All the operations associated with the same task must be executed the same number of times.

Proof: By contradiction. First let's look at the case where the workcell program consists of just one task. Now suppose that for some cycle, at least one operation

3.8

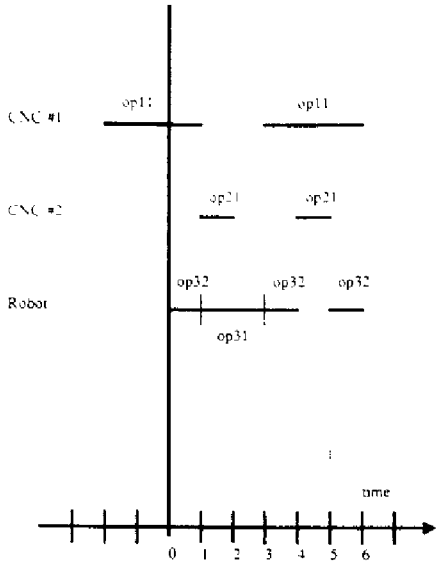


Figure 1. The repetitive sequence found via the heuristic for the first example, in the form of a Gantt chart.

Cycle time	Throughput		Processing rate		Machine utilization (%)			Forced waiting
	Task A	Task B	Task A	Task B	1	2	3	
5	1	2	0·20	0·40	0·60	0·40	0·80	No
6	1	1	0·16	0·16	0·50	0·17	0·50	Yes
7	1	3	0·14	0·42	0·43	0·43	0·72	Yes
7	1	1	0·14	0·14	0·43	0·15	0·43	Yes
10	2	1	0·20	0·10	0·60	0·10	0·50	Yes
12	2	2	0·16	0·16	0·50	0·17	0·50	Yes
12	2	2	0·16	0·16	0·50	0·17	0·50	Yes

Table 1. Characterizing the seven different repetitive sequences found with and without forced waiting, where the operations have the following durations: $\tau_{11} = 3$ s, $\tau_{21} = 1$ s, $\tau_{31} = 2$ s and $\tau_{32} = 1$ s.

is executed fewer times than the other operations. But by definition, each cycle represents some form of (strictly) repetitive behaviour in the space of possible time evolutions of the workcell. Therefore, we must be able to 'glue' the execution of a cycle to the next execution of the same cycle. But if some operation remained outstanding, we would be unable to 'add' it to the repetitive behaviour characterized by the cycle. Therefore, all the operations must be executed the same number of times during the cycle.

When the workcell program consists of N tasks, there will be N sets of associated operations, and the previous result must hold for each set.

The data associated with the repetitive sequence found via the heuristic are presented in the first row of Table 1. (The corresponding state space has nine nodes and

3.9

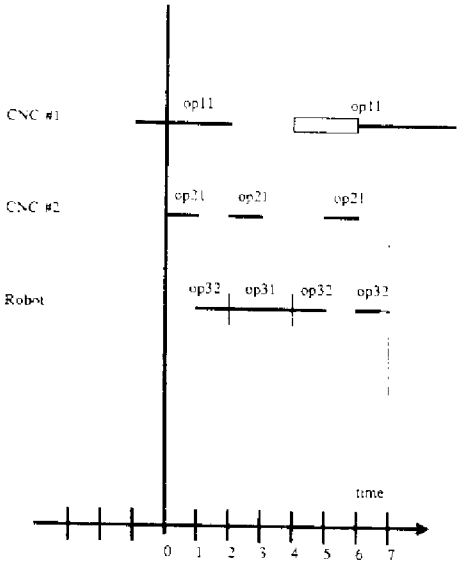


Figure 2. One of the repetitive sequences found via forced waiting for the first example, in the form of a Gannt chart. The shaded area denotes the interval of forced waiting. (Although operation op_{11} is enabled by the completion of op_{31}, it does not begin until 2 s later.)

11 arcs.) By suppressing the heuristic, six extra sequences involving forced waiting are found, as shown in the same table. (The corresponding state space now has 21 nodes and 54 arcs.) Clearly, the last two sequences (with duration 12 s) are merely permutations of the second sequence (duration 6 s) executed exactly twice.

To illustrate better how the forced waiting appears, the third sequence in Table 1 (cycle time 7 s) is presented in Fig. 2 in the form of a Gantt chart. The shaded area denotes the interval of forced waiting; although operation op_{11} is enabled by the completion of op_{31}, it does not begin until two seconds later.

Of particular interest are the first, second, and third sequences in Table 1. Although the second sequence is the time-optimal one with forced waiting, its cycle time is 1 s longer than the cycle time of the first sequence found via the heuristic (6 instead of 5 s), and the throughput of task B is actually reduced from 2 to 1. Indeed, both task processing rates are inferior since the cycle time is greater. But by increasing the cycle time by 1 s more to 7 s (the third sequence in Table 1), the throughput of parts of type B jumps from 1 to 3, and the utilization of CNC 2 and the robot (workcell elements 2 and 3) increases from 0·17 to 0·43 and from 0·50 to 0·72, respectively. The reader may wish to consult Figs. 1 and 2 to appreciate better how the sequencing of the workcell operation changes. Note that the third repetitive sequence might be preferred to the first one if a cycle time of 7 instead of 5 s was preferred at the factory level to better balance production flows between workcells, or if a greater throughput of task B (3 instead of 2) was desired at the expence of task A.

But the addition of forced waiting has important consequences on the time complexity of the sequencing analysis, as shown in Table 2. The last column is the sum of the CPU time† to enumerate the state space, identify all cycles, construct the

† Timing measurements are for a Sun-3·60 workstation.

3.10

Case	Number of nodes	Number of arcs	Number of sequences	CPU time (s)
With heuristic	9	11	1	5·5
Without heuristic	21	54	6	40·2

Table 2. Comparing the task spaces and time complexity of the sequencing analysis with and without the heuristic of local minimization of robot idle time for the first example.

Cycle time	Throughput		Processing rate		Machine utilization (%)			Forced waiting
	Task A	Task B	Task A	Task B	1	2	3	
8	1	2	0·12	0·25	0·62	0·17	0·75	No
9	1	1	0·11	0·11	0·56	0·23	0·45	Yes
12	1	3	0·08	0·25	0·42	0·50	0·67	Yes
14	2	1	0·14	0·07	0·72	0·15	0·43	Yes
18	2	2	0·11	0·11	0·56	0·23	0·45	Yes
26	3	4	0·11	0·15	0·58	0·31	0·54	Yes

Table 3. Characterizing six different repetitive sequences where the operations have the following durations: $\tau_{11}=5$ s, $\tau_{21}=2$ s, $\tau_{31}=2$ s and $\tau_{32}=2$ s.

corresponding repetitive sequences, and generate the statistics associated with the three metrics. It is clear that, at least for this example, the workcell analysis is rendered almost 10 times more complex by the suppression of the heuristic and the concomitant addition of forced waiting. (Not shown in Table 2 is the space (memory) required by the analysis, since we are primarily concerned with time complexity.)

To what extent are these figures representative if the actual durations of the workcell operations differ from the values used in the sequencing analysis? Since the operations are typically strongly coupled, changing their relative durations will, in general, change the repetitive behaviour of the workcell (except when the workcell tasks are strictly sequential).

For example, when all four operations have the same duration, e.g. $\tau_{11}=\tau_{21}=\tau_{31}=\tau_{32}=2$ s, a single repetitive sequence is found via the heuristic, with cycle time 4 s and each task executed once. (The corresponding state has five nodes and seven arcs.) By suppressing the heuristic, five other repetitive sequences are found with forced waiting (cycle times 4, 6, 8, 8 and 12 s) which are permutations of the time optimal sequence. (The corresponding state space is larger, composed of 11 nodes and 30 arcs.)

Now by changing just the duration of op_{11} to 5 s, a single repetitive sequence is found via the heuristic with cycle time 8 s, task A is executed once, and task B is executed twice. (The corresponding state space has eight nodes and nine arcs.) By suppressing the heuristic, 14 other repetitive sequences are found with forced waiting, with cycle times ranging from 9 to 29 s. (The corresponding state space is again much larger composed of 31 nodes and 77 arcs.) Results for a subset of these sequences are given in Table 3. Again we observe that by selecting a repetitive sequence which is not time optimal (in the sense of minimal cycle time), the respective throughputs of tasks A and B can be radically altered, along with the percentage utilization of the workcell elements.

For all mixes of durations of the operations for this example, the time-optimal sequence can always be identifed using the heuristic. Therefore, the other sequences

3.11

'discovered' by forced waiting simply demonstrate how cycle time can be traded off against task throughput and machine utilization. In the following section, we present a different example for which the time-optimal sequence is not discovered using the heuristic.

6. A second example

Consider the example of two robots R1 and R2 performing a single task: cooperative assembly. The assembly consists of two parts, p_1 and p_2. Each part must be pre-processed in some way (e.g. inspected or aligned) and then attached to a common base. In addition, part p_1 must be re-positioned after the pre-processing before it can be attached to the base. We shall define the workcell operations to perform the assembly task as follows:

op_{11}: unload the finished assembly at OUTPUT, and load from INPUT new parts p_1, p_2
op_{12}: pre-process p_2, and attach to base
op_{13}: re-position p_1
op_{21}: pre-process p_1
op_{22}: attach p_1 to base

with the following mix of durations: $\tau_{11} = 1$ s, $\tau_{12} = 3$ s, $\tau_{13} = 3$ s, $\tau_{21} = 1$ s and $\tau_{22} = 3$ s.

Using the simple heuristic, a single repetitive sequence is found, cycle time = 10 s. (The corresponding state space has four nodes and five arcs.) The associated data are given in the first row of Table 4. Now by suppressing this heuristic, nine other repetitive sequences are found; the results are summarized in the same table. (The corresponding state space is much larger, with 21 nodes and 33 arcs.) Clearly, the time-optimal repetitive sequence (second row) is superior in every way to that found via the heuristic (first row) even though it incorporates forced waiting.

The reader may be puzzled by the existence of four repetitive sequences with forced waiting which have the same cycle time (10 s) as that of the repetitive sequence without forced waiting. This is due to the fact that both robots are under-utilized by the assembly task, and there are exactly four different ways of forcing them to wait without increasing the cycle time of the repetitive sequence.

The consequences of adding forced waiting to the sequencing analysis are summarized in Table 5. Once more, the CPU time required is increased about 10 times.

Cycle time	Number of sequences	Throughput	Processing rate	Machine utilization (%)		Forced waiting
				1	2	
10	1	1	0·10	0·70	0·40	No
8	1	1	0·12	0·75	0·75	Yes
10	4	1	0·10	0·70	0·40	Yes
11	4	1	0·09	0·63	0·36	Yes

Table 4. Characterizing ten different repetitive sequences where the operations have the following durations: $\tau_{11} = 1$ s, $\tau_{12} = 3$ s, $\tau_{13} = 3$ s, $\tau_{21} = 1$ s and $\tau_{22} = 3$ s.

3.12

Case	Number of nodes	Number of arcs	Number of sequences	CPU time (s)
With heuristic	4	5	1	2·0
Without heuristic	21	33	10	20·0

Table 5. Comparing the task spaces and time complexity of the sequencing analysis with and without the heuristic of local minimization of robot idle time for the second example.

Cycle time (s)	Number of sequences	Throughput	Processing rate	Machine utilization (%)		Forced waiting
				1	2	
4	1	1	0·25	0·50	0·75	No
4	2	1	0·25	0·50	0·75	Yes
5	3	1	0·20	0·40	0·60	Yes

Table 6. Characterizing the different repetitive sequences with and without forced waiting, where the operations have the following durations: $\tau_{11} = 1$ s, $\tau_{12} = 1$ s, $\tau_{21} = 1$ s and $\tau_{22} = 2$ s.

7. A third example

To decouple partially workcell operations, sites for temporary parts storage within the workcell may sometimes be used, in order to obtain more robust behaviour or to enable the concurrent execution of multiple instances of the same task. As described in an earlier section, we associate the motions to/from these sites with *auxiliary* workcell operations. In this third example, we demonstrate how the consequences of adding such sites and auxiliary operations to the workcell can be evaluated.

Here we begin with a variation on the previous example. Two workcell sites are defined: s_1 for the input and alignment of parts of type p_1; and s_2 for the input of parts of type p_2 and for the assembly. The workcell operations are defined as follows:

op_{11}: load from INPUT1 a new part p_1 at s_1, and add alignment pin
op_{12}: move p_1 from s_1 to s_2
op_{21}: load from INPUT2 a new part p_2 at s_2
op_{22}: assemble p_1 and p_2, and move to OUTPUT

with the following mix of durations: $\tau_{11} = 1$ s, $\tau_{12} = 1$ s, $\tau_{21} = 1$ s and $\tau_{22} = 2$ s.

Using the simple heuristic, a single repetitive sequence is found, with cycle time = 4 s. (The corresponding state space has four nodes and five arcs.) Now by suppressing this heuristic, a total of five other repetitive sequences with forced waiting are found, with cycle times of 4 and 5 s. (The corresponding state space is much larger, with 12 nodes and 20 arcs.) All these results are summarized in Table 6. Clearly, the extra cycles with forced waiting are no better than the time-optimal cycle with no forced waiting.

Note the existence of two repetitive sequences with forced waiting which have the same cycle time (4 s) as that of the repetitive sequence without forced waiting. This is due to the fact that robot R1 is under-utilized by the assembly task; therefore it can be commanded to wait 1 s (at exactly two different time instants) without increasing the cycle time of the repetitive sequence.

3.13

7.1. *Adding a buffer*

Now we shall add to the workcell one buffer with capacity one part, to be used by R1 to temporarily stock a part of type p_1 while R2 is busy (and site s_1 is occupied). Therefore, a third workcell site b (buffer) is defined, along with the following operations:

op_{13}: move p_1 from s_1 to b
op_{23}: move p_1 to s_2

with the durations $\tau_{13} = 1$ s and $\tau_{23} = 1$ s.

Using the simple heuristic, two repetitive sequences are found with cycle time = 4 s; must one sequence makes use of the buffer. (The corresponding state space has 13 nodes and 17 arcs.) The data for these two repetitive sequences are presented in the first two rows of Table 7. Since the second sequence involves the buffer, R2 must execute an additional operation op_{23}; therefore; its utilization increases from 0·75 to 1·0. However, the per cent utilization of R1 does not change since op_{13} replaces op_{12} and these operations have equal duration.

Now by suppressing this heuristic, a total of 44 extra repetitive sequences with forced waiting are found, with cycle times ranging from 5 to 22 s. (The corresponding state space is much larger, with 34 nodes and 82 arcs.) Data for a subset of these repetitive sequences are also presented in Table 7. It is clear that, for this example, the presence of the buffer is key to the formation of longer cycles.

The data in Table 2 also demonstrate how machine utilization is reduced as the cycle time increases. Indeed, machine utilization is no longer important for comparing repetitive sequences when auxiliary operations are executed, since these operations cannot directly affect the task throughput and, therefore, the processing rate of the workcell. However, machine utilization could still be used to help schedule periodic maintenance.

The relative complexities of the sequence analysis for the various cases are summarized in Table 8. When the buffer is not considered, the addition of forced waiting increases the CPU time by a factor of 5 for this example (as opposed to a factor of 10 for the two previous examples). When the buffer is added, even the CPU time

Cycle time (s)	Throughput	Processing rate	Utilization (%)		Buffer used	Forced waiting
			1	2		
4	1	0·25	0·50	0·75	No	No
4	1	0·25	0·50	1·0	Yes	No
5	1	0·20	0·40	0·60	No	Yes
5	1	0·20	0·40	0·80	Yes	Yes
6	1	0·16	0·33	0·67	Yes	Yes
9	2	0·22	0·44	0·78	Yes	Yes
10	2	0·20	0·40	0·70	Yes	Yes
11	2	0·18	0·37	0·73	Yes	Yes
15	3	0·20	0·40	0·73	Yes	Yes
17	3	0·17	0·35	0·65	Yes	Yes
21	4	0·19	0·38	0·67	Yes	Yes
22	4	0·18	0·36	0·64	Yes	Yes

Table 7. Characterizing a subset of the different repetitive sequences for the cases of with and without the workcell buffer.

3.14

Case	Number of nodes	Number of arcs	Number of sequences	CPU time (s)
No buffer, heuristic	4	5	1	2·0
No buffer, no heuristic	12	20	5	9·3
Buffer, heuristic	13	17	2	8·8
Buffer, no heuristic	34	82	44	131·9

Table 8. Comparisons of the task spaces and time complexity of the sequencing analysis with ant without the heuristic of local minimization of robot idle time, for the cases of with and without the workcell buffer.

required using the heuristic is greatly increased. But this increase is much smaller than that incurred with forced waiting. Although the size of the task space does not grow inordinately, the number of repetitive sequences discovered increases 20-fold. This is due to the fact that the buffer partially de-couples the operations of the two robots, thereby leading to many more possible sequences.

The comparative analysis presented here could also be used to study other kinds of operational variations. For example, when several workcell elements share the same workcell volume, it is sometimes convenient to command them to wait at a particular location outside the shared volume whenever they are idle, in order to simplify the execution of the other operations which involve the shared volume. Clearly, the consequences of 'adding' such locations (and their associated motions) could also be investigated via our sequencing analysis.

8. SAGE

In its current form, SAGE is a collection of 14 small programs written in C-Prolog (Pereira 1984) (comprising some 3500 lines of source code) which execute under the Unix operating system on either VAX or Sun families of workstations. Only the graphics interface (Brachman 1985) is particular to Sun, since it makes use of the SunCore graphics library.

SAGE creates a representation of the repetitive runtime behaviour called the *task space* from a description of the workcell application as a timed Petri net. The core of the analysis is performed by three programs, as described below.

8.1. *Generation of the task space*

The first key program generates the task space in a breadth-first manner, as a set of nodes and arcs. When the workcell application is strictly sequential, each operation has a unique successor operations and the nodes and arcs form a single cycle. However, most practical sequencing problems exhibit some degree of internal non-determinism which gives rise to branching in the task space, i.e. when multiple *competing* operations become enabled for the same workcell element, SAGE explores the consequences of each alternative by creating a new node for each permutation of elements in the lists of candidate successor operations.

Unlike conventional simulation in which time advances by some fixed interval at each iteration, the time interval here between iterations changes as the task space is explored. At each iteration, we 'leap' ahead to the first time instant at which a condition now pending will become true. Experiments with workcell applications of moderate

3.15

size and typical variation in the durations of the operations have shown that this adaptive timestep can reduce by two thirds the computer time required to generate the task space.

The time complexity of the task space generation and analysis is difficult to measure, since it depends upon how the operations are related and their durations. If at some time there are N active elements $\{i\}$ which have $\{n_i\}$ successor operations, then the number of possible permutations to consider is $n_1 \times n_2 \times \ldots \times n_N$. As previously noted, each permutation becomes a new branch in the task space to explore when this number is small. But when this number is large, the heuristics must be invoked which introduces extra time complexity. Similarly, the extra complexity associated with the calculating the adaptive timestep can only be justified when the durations of the operations vary substantially; otherwise, time could be more simply advanced by a fixed value (such as the greatest common divisor of all the durations) which could be calculated just once at the state of the task space generation.

8.2. *Finding cycles in the task space*

A closed path in the task space consists of a cycle characterizing some repetitive behaviour, plus some transient behaviour associated with start-up from the initial workcell state. Although we shall now restrict our attention to just the cycle, it must be noted that the transcience is also part of the runtime 'program', since it describes how the workcell is driven from the initial state to 'induce' the repetitive behaviour.

Cycles in the task space are found in two steps. First, a spanning forest is generated using depth-first search. Then, the extra arcs in the task space which are not in the spanning tree are processed; these arcs are called 'chords'. In this way, we obtain simple cycles which are elementary, i.e. no node is visited more than once. Finally, elementary cycles which share nodes are combined to obtain new cycles which correspond to the 'interleaving' of independent workcell tasks (see Section 2).

8.3. *Cycle analysis*

Although cycles in the task space correspond to repetitive sequences, not all sequences are *complete*, i.e. each of the required workcell operations is executed at least once. (Auxiliary operations may or may not be executed.) For instance, the task space of the first example described above for two CNC machines and one robot contains two incomplete cycles consisting of just one CNC machine and the robot working together while the other CNC machine lies idle. Clearly, this means that just one of the two tasks in the workcell program is performed by the workcell.

The cycle analysis has three steps, and is performed by two programs. First, cycles which are incomplete are deleted from further consideration; second, the repetitive sequences based on the remaining cycles are created; and, finally, the three metrics are determined for each sequence.

The time complexity of the cycle analysis is simply ($O(N)$, where N is the number of cycles in the task space, since each sequence is examined just once at each step.

9. Conclusions

In this paper, we have shown that when the workcell tasks are repetitive, the notion of a workcell *cycle* becomes important and we can associate with each workcell element a repetitive *sequence* of operations. But there may be multiple repetitive sequences inherent in the mapping of the tasks to workcell operations. This is the motivation for

3.16

our decision-support system called SAGE (sequence analysis by generalized enumeration). First, the state space of all possible time evolutions called the task space is enumerated. Cycles in this space represent repetitive sequences, and those which are 'complete', i.e. which contain all the required operations, can be used by the workcell runtime system to perform the intended tasks. Using cycle time, task throughput, task-processing rate, and machine utilization, we demonstrate how the alternative sequences could be compared.

In contrast to AI-oriented scheduling systems, our work has a completely different emphasis since we view sequence optimization as part of the *off-line* development of the workcell program, rather than as part of the runtime control. For example, we make no attempt to 'resolve' runtime conflict; rather, we simply explore the consequences of resolving such conflict in different ways. In this sense, SAGE is designed simply to help the workcell programmer evaluate alternative runtime 'strategies'. Note too that SAGE performs no 'planning' in the sense that there are no goals to be achieved (except the obvious one that the intended workcell task must be performed).

To simplify the sequence analysis, a simple *application-independent* heuristic was introduced, to locally minimize the idle time of each workcell element. In this sense, just a subset of the complete task space was searched; therefore, just a subset of possible workcell cycles were identified. By suppressing the heuristic, the complete task space is searched and different repetitive sequences with *forced waiting* were found. Sometimes such a sequence with forced waiting was 'better' than the sequences without forced waiting. We are now looking for ways of recognizing this 'sensitivity' to forced waiting, within the context of our ongoing work with timed Petri nets.

However, the suppression of the heuristic greatly increases the time complexity of the sequence analysis. For example, this might mean several minutes (instead of seconds) of calculation on a Sun-3/60 workstation. However, we see this as no real obstacle since the sequencing analysis described in this paper is meant to be performed *off-line*.

Clearly, the simple examples presented here are not representative of typical workcell tasks. Nonetheless, the state space of even these examples is much too large to be analysed in a manual way. To handle larger problems, other pruning mechanisms must be used. For example, a desirable task throughput could be specified by the user as part of the specification of the workcell program. Then when a partial path (not yet a cycle) in the task space exceeds this throughput, it would be deleted from further consideration. In the case of our first example, if the throughputs or tasks A and B were to be limited to one, then five of the seven possible repetitive sequences would be pruned (see Table 1).

Of course, the C-Prolog implementation itself limits the speed of analysis and the size of analysable problems. This was the motivation for the development of a second implementation of SAGE for a different Prolog which offers the possibility of compilation. We report new results (Freedman and Alami 1990) which indicate that CPU time required for the analysis is now two orders of magnitude less, making it possible to treat workcell programs of a more practical size.

Although not treated in this paper, a restricted amount of external non-determinism related to runtime sensing can be analysed when the workcell program consists of a single task. For example, consider a computer vision system performing inspection of printed circuit boards; the outcome of the inspection operation is either 'pass' or 'fail'. By 'pretending' that each outcome always occurs, we can construct two families of cycles, one for 'pass' and the other for 'fail' (Freedman 1989). Each cycle

3.17

would correspond to some repetitive workcell behaviour 'anchored' by the inspection operation. The various cycles could then be compared using the metrics developed in this paper (cycle time, task throughput, etc.) and an optimal cycle would then be chosen from each family. These two cycles would form the basis of two distinct workcell runtime programs. The global behaviour of the workcell could then be summarized by averaging their metrics, e.g. $\pi(\text{average}) = (\pi(\text{pass}) + \pi(\text{fail}))/2$. If the frequencies of the outcomes can be *a priori* estimated, e.g. 90% pass, 10% fail, then a weighted average could be calculated, e.g. $\pi(\text{average}) = 0{\cdot}90 * \pi(\text{pass}) + 0{\cdot}10 * \pi(\text{fail}))$.

References

ALAMI, R., 1984, NNS, a Lisp-based environment for the integration and operating of complex robotics system. *IEEE International Conference on Robotics and Automation, Atlanta (USA)*, March 1984.

ALAMI, R., and CHOCHON, H., 1986, Programming of flexible assembly cells: task modelling and system integration. *Robotics and Industrial Engineering—Selected Readings*, Vol. II.

ALAMI, R., and CHOCHON, H., 1988, Programmation et contrôle d'exécution d'une cellule flexible d'assemblage. *Techniques de la Robotique* (Editions Hermes).

BAKER, K., 1974, *Introduction to Sequencing and Scheduling.* (New York: Wiley).

BARAD, M., and SIPPER, D., 1988, Flexibility in manufacturing systems: definitions and petri net modelling. *International Journal of Production Research*, **26**, 237–248.

BEN-ARIEH, D., MOODIE, C., and NOF, S., 1985, Knowledge-based control for automated production and assembly. *8th International Conference on Production Research*, pp. 285–292.

BENSANA, E., BEL, G., and DUBOIS, D., 1988, Opal: a multi-knowledge-based system for industrial job-shop scheduling. *International Journal of Production Research*, **26** 795–820.

BHARATH, R., 1986, Logic programming: A tool for ms/or? *Interfaces*, **16**, 80–91.

BRACHMAN, B., 1985, *G-Prolog User's Manual* (Department of Computer Science, University of British Columbia).

CARLIER, J., and CHRETIENNE, P., 1985, Studies of solutions if scheduling problems associated with timed petri nets. *Technical Report 57* (MASI, Institut de programmation, Université de Paul et Marie Curie).

CLOCKSIN, W., and MELLISH, C., 1981, *Programming in Prolog* (Berlin: Springer-Verlag).

DAVIS, R., 1985, Logic programming and prolog: a tutorial. *IEEE Software*, 53–62.

ERSCHLER, J., and ESQUIROL, P., 1986, Decision-aid in job shop scheduling: a knowledge-based approach. *IEEE International Conference on Robotics and Automation*, pp. 1651–1656.

FREEDMAN, P., and ALAMI, R., 1990, Repetitive sequencing: from workcell tasks to workcell cycles. *IEEE International Conference on Robotics and Automation*, Cincinnati, Ohio.

FREEDMAN, P., 1989, Characterizing repetition in workcell applications and the implications for sequence optimization. *4th International Conference on Advanced Robotics (ICAR)*, Columbus, Ohio.

FREEDMAN, P., and MALOWANY, A., 1988 a, The analysis and optimization of repetition within robot workcell sequencing problems. *IEEE International Conference on Robotics and Automation*, pp. 1276–1281.

FREEDMAN, P., and MALOWANY, A., 1988 b, Sage: A decision support system for the sequencing of operations within a robotic workcell. *Decision Support Systems*, **4**, 329–343.

GAREY, M., and JOHNSON, D., 1979, *Computers and Intractability: a guide to theory of NP-completeness* (New York: Freeman and Co).

GIBBONS, A., 1985, *Algorithm Graph Theory* (Cambridge: Cambridge University Press).

GIRALT, G., 1984, Research trends in decisional and multisensory aspects of third generation robots. *Robotics Research: The Second International Symposium*, edited by H. Hanafusa and H. Inoue (Cambridge, MA: MIT Press).

HALL, R., 1988, Cyclic scheduling for improvement. *International Journal of Production Research*, **26**, 457–472.

LEE, R., and MILLER, L., 1986, A logic programming framework for planning and simulation. *Decision Support Systems*, **2**.

3.18

MEREDITH, J., 1987, Automating the factory: theory versus practice. *International Journal of Production Research*, **25**, 1493–1510.

MILLER, D., FIRBY, J., and DEAN, T., 1985, Deadlines, travel time and robot problem solving. *Proceedings of 9th IJCAI 1985*, pp. 1052–1054. (IJCAI).

PEREIRA, F., 1984, *C-Prolog User's Manual* (Department of Architecture, University of Edinburgh).

RAMCHANDANI, C., 1974, Analysis of asynchronous concurrent systems by petri nets. *Technical Report* (L.C.S. M.I.T.).

SHAW, M., 1988, Knowledge-based scheduling in flexible manufacturing systems: an integration of pattern-directed inference and heuristic search. *International Journal of Production Research*, **26**, 821–844.

WEMMERLÖV, U., and HYER, N., 1987, Research issues in cellular manufacturing. *International Journal of Production Research*, **25**, 413–341.

3.19

J.2 Planification d'actions

4 *Dealing with Time in Planning and Execution Monitoring.*

R. Bolles, editor, *Robotics Research: The Fourth International Symposium.* MIT Press, Mass., 1988.

Auteurs: M. Ghallab, R. Alami, R. Chatila.

5 *Planning with Non-Determistic Events for Enhanced Robot Autonomy*

Intelligent Autonomous Systems, Karlsruhe, Mars 1995.

Auteurs: J. Perret, R. Alami.

Dealing with Time in Planning and Execution Monitoring

Malik Ghallab, Rachid Alami, Raja Chatila

LAAS-CNRS, 31077 Toulouse cedex, France

Two generic classes of robotics applications, the structured environment class, and the unstructured environment class, are considered with respect to planning and execution monitoring. The paper analyzes the common and distinct requirements in these 2 classes for representing time and dealing with time and real-time in planning and execution monitoring tasks. An original approach is proposed. It is base on a hierarchical representation of temporal relations between goals, events, actions and their effects, together with an efficient management of the time lattice. This approach is developed in the context of an experimental project: a flexible assembly cell.

1 Introduction

Robotics researches focus usually on one of the two generic classes of robotics applications:

- the structured environment class, and
- the unstructured environment class.

A good paradigm of the first class is that of flexible assembly cells, whereas autonomous mobile robots, for example in public safety applications, are prototype elements of the second class. Although there are some applications that fall in between (e.g. a flexible maintenance and repair cell), these two classes present a significant spectrum of research problems that have to be solved in third generation robots.

There are several common points and differences between the two classes, for example in space representation and geometric reasoning, in control problems, or in perception systems. We will be concerned here solely by planning and execution monitoring aspects involved in these two classes.

The Robotics Group at LAAS has been developing for several years two experimental projects, one in each class: HILARE, a mobile robot [16] [8], and NNS an environment for managing a flexible assembly cell [1] [11]. Two distinct approaches to planning and execution monitoring have been implemented in these two projects. Common points were mainly rule based formalisms and tools, such as production rules compilers. None of our previous approaches was able to represent explicitly and deal with time and real-time at the planning level, and at the action and reaction level.

The first goal of this work was thus to analyse the common and distinct requirements in these 2 generic classes for representing time and dealing with time and real-time in planning and execution monitoring tasks. A subsequent goal was to come up with a satisfactory approach for meeting these requirements, and to apply it to our experimental projects.

This paper reports on our findings and the actual state of the work. The next section analyses the temporal structures required, surveys known methods, and develops the proposed approach that relies on an original data structure, called an *Indexed Time Table* (IxTeT). An *IxTeT* is a hierarchical representation of temporal relations in a plan (relations between goals, events, actions and their effects), together with an efficient management of the time lattice. Section 3 develops and illustrates through an example of realistic complexity the embodiment of the proposed approach for one of the 2 generic classes considered: the flexible assembly cell environment.

2 Temporal structures for planning and acting

Different types of actions may involve different kind of resources (space, tools, sensors, particular abilities of the robot, programs and computer resources...). But every action involves time. Information describing an action exhibit a rich temporal structure: the time at which the action has been (or will be) considered and decided, when it could be carried out, when it did (or may) start, its duration, when each of its effects held (or will hold), and the various relations (before, during, at the same moment...) between these time points or intervals. Events that take place in a dynamic environment have similarly a temporal structure (except that they are not under the robot control).

Such knowledge is difficult to represent and manage, it lacks precision and certainty, and leads to huge combinatorics. It is however absolutely required if a robot has to act in a dynamic environment coherently and efficiently towards some goal, and to react sensibly and in real-time to external events. Let us analyse the requirements for representing explicitly and reasoning on temporal knowledge in planning and execution monitoring tasks: what we would like to express and do concerning time.

2.1 Temporal knowledge required in planning

Planning for a goal, i.e. deciding in a given situation what to do and when to do it, is a projection over a desirable future. Such projection relies on the knowledge of past and current states of the world, and results from the analysis of several possible futures where goals and expected events have been tentatively set and tried to be met or dealt with through actions. It is thus mainly a reasoning on time: a processing of the temporal structures of goals, actions and events.

Goals: in general they may be given with both relative and absolute time links. The "conjunctive goals" planning problem [7] is just a particular case of the situation where goals are partially ordered. For example: "pack up bolts A and assemble sub-system B then bring back both".

Goals may also be linked by synchronisation, overlapping and duration constraints, e.g. "feed parts A and B simultaneously" ; "keep

spring A loaded while inserting shock absorber". Similar relations may be used to set goals relatively to expected events, e.g. "heat until but no more than the red point" ; "unload wagon A during its next stop".

Achievement of goals may also be specified with regards to absolute time bounds: "reach location A before sunset" ; "fix the leak or leave area B before end of count-down timer" .

Thus in general a goal corresponds to an interval of time (or a time point) during which a property should hold, and one may need to constrain the length or position of this interval relatively to other goals, events or absolute references.

Actions: they take place during a time span that has a constrained length (duration). Their effects and conditions should be located with respect to this interval. Some effects hold from the beginning of an action, others during it or when the actions ends; indirect effects may lag well behind the action.

A decomposition operator (or a skeleton of plan) may describe a structure of elementary actions together with their relative position in time and their compound effects.

Actions that share resources or are done by the same agent may have additional time constraints that do not result from their elementary description (such as overlapping, disjoining or synchronization relations).

Events and world description: the current state of the world is just a part of what needs to be described in a dynamic environment where changes may result from other causes then the robot own actions.

Past events and states could be important for future decisions. This is the case for events without effects on the present but that may have future results; for example an intermittent failure, or a fixed failure that could have a delayed effect.

Information about the expected future are evidently essential in planning: what will or may happen in the environment if the robot does not act. This concerns:

- **scheduled events:** bound to happen at known time (relative or absolute); e.g. "workshop garbage is collected at 07:00" ; "feeding cart comes every 20 minutes".
- **conditional events:** will happen when some conditions are met (immediately or after a delay); e.g. "overflow of a bin or storage place at some threshold", "failure in computers room cooling equipment, if not fixed, will results approximately 1 hour later in an automatic shut down of all computers".

A robot may plan to act in order to prevent some expected events (the above shut down). It may also plan to take profit from their known effects to reach its goals, thus synchronising its actions with world changes. Processes that take place in the environment may be considered just like events, except that they have durations that could be constrained.

In summary the various knowledges required as input by a planner, i.e. goals, actions, events and world properties, have time intervals that can be constrained in duration, are related to each others and to absolute references.

The output of a planner should be a conditional plan: several sets of actions, temporally structured, and whose projected effects achieve the given goals and their temporal constraints, taking into account current and past states of the environment together with the expected events. This conditional plan should be generated together with an execution model that specifies:

- the conditions and rules that will be evaluated at execution time to resolve non deterministic choices,
- the properties to be monitored for checking direct or indirect effects of actions,
- the sensorial informations to focus attention on in order to detect expected events.

The corresponding temporal knowledge would be: when and for how long checking, monitoring and focusing should be done. The plan execution model should also detail synchronization steps and other critical phases, taking into account the real duration of actions, delays of their effects, length of processes...

As it will be argued and illustrated later, the differences between a structured environment planner and an unstructured one cannot be grasped into a unified approach unless planning is considered as a hierarchical process that is pursued deeper opportunistically as conditions (knowledge, time for planning...) permit or require.

2.2 Time in Execution Monitoring

Execution monitoring is used here in a broader sense than the restricted literal interpretation. In fact we are interested in most decision making aspects related to acting and reacting.

Acting requires necessarily a plan. There is however a large overlapping between planning and acting. Plans need refinement at execution time: an action considered elementary at planning level may require at execution level a further decomposition chosen such that it fits the current situation. Conditional plans involve choices between alternatives, eventually only one of which (the most likely one) has been fully pursued at planning time but another one may need to be developed and resubmitted to the planner at execution time. Unexpected events may require partial modification or rejection of current plan. Even current goals may have to be discarded or postponed for more urgent ones.

Tasks devoted to a robot execution monitor are thus: monitoring, keeping track of the state of achievement of current plan, refining adequately actions and implementing them, chosing among alternatives, focusing the attention of sensory systems, deciding about short terms reactions to unexpected events. Those tasks involve mainly 2 temporal aspects: how one keeps track and defines permanently the present time, and how one deals with real-time.

The present is defined relatively to the projected future. It evolves normally as planned, or may "jump" backward in the plan, or forward, or even out of the plan. The advance of time is discontinuous and driven by asynchronous events (although a robot may have a clock that gives the absolute time). Each expected event when observed instanciate a particular future: if P and Q are expected next, Q happening before P reduces uncertainty for what should come later. Observing the occurrence of an event expected much later may change locally or significantly the rest of the plan, e.g. if it was to result from a forthcoming action, the purpose of this action has to be reconsidered. A "positive" event (forward jump in the plan) may achieve directly part of the goals, e.g. a motor that resumes operations while the robot is in its way to reach it for repair. On the opposite a "negative" event (backward jump) may cancel the effects of a previous actions and make them necessary again or show them ineffective and require another plan.

The real-time requirement is mainly due to the dynamic feature of the environment, i.e. to unexpected changes and events that happen asynchronously, at a speed not under the robot control, but that require from it adequate reactions. Such events should be perceived, identified, and at least partially understood before a decision about how to react can be taken. We are talking here about unexpected events that the robot knows about, and can deal with.

To perceive and identify an unexpected event a robot should be looking for its eventual happening. This awareness or focus of attention relies on execution monitoring: critical actions or phases of a plan

should trigger particular monitorings. The real-time reaction to an event should correspond to a set of actions hierarchically ordered by response time, e.g. with the following 3 levels:

- an immediate reflex action: most of the time a direct mapping from the event identification;
- a short term adaptation to the new situation: to permit full assessment, goals evaluation and replanning;
- a long term reaction according to the new plan

For example: a fast autonomous vehicle faces a closed path

- reflex: it brakes to avoid collision
- short term adaptation: it parks safely and replan
- long term reaction: it takes another itinerary (this can overlap with the previous level).

2.3 Time in structured and in unstructured environment applications

A scenario for a structured environment application is that of a multi-sensory, multi-arm, flexible cell for robotics assembly that has to carry out the same task for a short time (few days) with the maximum degree of robustness, error recovery and efficiency (see detail in section 3.1)

A scenario for an unstructured environment application is that of mobile robot working in a hazardous area that is required to go to some place, to find and fix a leak, or to leave that place before an automatic shut down.

The following table sketches the main differences of interest to us between this two classes:

	structured	unstructured
- variability of the environment	low	high
- variability of goals	low	high
- degree of knowledge modelization and programming	high	low
- degree of autonomy	low	high
- degree of parallelism	high	low

How time is involved in this two classes of applications ?

Structured environment:

- off-line tasks : time is involved only at the reasoning level
- on-line tasks :

  -each agent in this distributed multi-agent system has the same share of time (other resources are shared differently);

  - at a macroscopic level time is seen as periodic (repetition of the task);

  - time has to be optimized

Unstructured environment:

- there are almost no off-line tasks (except high level learning), most planning has to be done in real-time;
- there are more constraints on a unique agent, less parallelism and almost no repetitive tasks
- a clock giving the absolute time is required

These differences have several consequences for planning and execution monitoring. In the structured case most of the planning can be done off-line. Giving the a priori knowledge about the environment and its low variability, one may aim for a conditional plan, or a scheme of plans, that foresees a large number of possible futures, and organize this set of plans into a detailed execution model.

In the unstructured case a detailed plan, even if it can be generated, would be useless (not achievable). The task at hand could be decomposed into a plan skeleton that specifies the main subgoals, constraints and choices involved in this task. Further refinements will be carried out at execution time.

Planning being done on-line in the unstructured case, it is just like any other action: it has a duration (unknown, but constrained) and overlaps with other actions. As time permit, planning can be pursued into deeper levels of detail, or stopped at just a sketchy skeleton together with the first action required to start the plan. Thus planning should be a hierarchical process. Planning and execution controlling are both involved at action time. Unexpected events that make the plan non achievable are easily checked, those not involved in the plan but that make it obsolete require a limited replanning.

In the structured case the on-line system relies on the execution model to instanciate a particular plan, among the schema of plans, that fit the current situation. No simultaneous use of the off-line planner and the on-line system is required.

2.4 Temporal Knowledge in known Planners and Execution Monitors

Now that we have freely prospected what knowledge we would like to express and what should be done with it, let us briefly survey what can be done with known planning and execution monitoring technics.

Situation calculus and state-space representations: The paradigm here is STRIPS [13]. There is no explicit representation of time in such systems, just a linear sequence of states. Actions are modeled by a triple <pre-conditions, delete-list, add-list>. They do not have a duration, and their effects take place in a single instantaneous transition from state to state. The goal is a single state. No coming event can be taken into account. The world is assumed to be static but of the only actions planned for (thus there is no real time).

Some discrepancy with planned states can however be taken into account at execution time. Execution monitoring relies on the Triangle Table method [12]. A Triangle Table is a data structure that summarises all conditions and effects of a sequences of STRIPS-like actions and their relationships. The "kernel" of the table defines what has been consistently achieved so far. If permanently computed at execution time it will enable to locate the present state of the world in the plan and to inform the execution monitor about what to do next [19] and [20]. Replanning will be required if the kernel is empty.

Procedural or task networks: They have been developed in NOAH [21] and NONLIN [23]. Planning is a hierarchical process based on task decomposition. For efficiency reasons and in order to avoid unnecessary constraints and backtrackings ("early commitment") concurrent sub-tasks are considered. A decomposition is a partial order of actions and sub-tasks to be achieved by further decomposition. Ordering constraints are later on added by a critical assessment of the network (if not possible a backtracking is performed). The planner's output is thus a partial order of instantaneous actions, opportunistically ordered at execution time.

Windows and durations: They are temporal concepts used in DEVISER [25] to extend the task network approach. A window is one or 2 numerical bounds on a forthcoming time point. Goals are constrained relatively to absolute references, e.g. (goals ((window between 10 50) (duration 1000)) (P) (Q)) requests that properties

P and Q should be achieved simultaneously sometime between t=10 and t'=50, and should hold for at least 1000 time units. Actions are represented by the usual triples <pre-conditions, del, add> in addition to a fixed duration. All effects take place at the end of an action. Unconditional events scheduled at times bounded by known windows can be expressed. Planning is performed as in NONLIN by node decomposition, this is done in addition to a propagation of numerical inequalities along the task network, that constrain further allowed windows and may lead to backtracking. The planner's output is similar to a PERT chart.

Thus DEVISER represents time as points, bounds and values on the real axis, that are processed only when given numerical values. No relative relationships (between goals, events, actions and their effects) are allowed, neither are conditional events.

Temporal logic of intervals: It has been developed in [2] and [4], and proposed for planning in [3]. It is a general world model where each assertion is temporally qualified by a symbolic interval over which it holds. An interval is related to another one by a temporal relation or a disjunction of (mutually exclusive) relations, such as: during, start, overlap. Each interval is represented as a node in a complete graph whose consistency is maintained by transitive closure propagation, e.g. adding "A overlaps or meets B" propagates to all other intervals related to A and B, thus adding new constraints that propagate at their turn... eventually leading to a contradiction. Past and current properties of the world are directly expressed in this temporal graph. Even non temporal properties are represented as quantified expressions on intervals.

Future events are temporally related to the conditions that trigger them. Goals and their relationships appear like expected events except that they should result from actions to be added in the graph.

An action is also an interval temporally linked to the intervals of its conditions and effects intervals, e.g.

```
"if action STACK(x,y) occurs over interval I-stack then
  I-stack finishes the interval I-clear-y over which
CLEAR(y) holds, and
  I-stack meets the interval I-on-xy over which
ON(x, y) holds, and
  I-stack is during the interval I-clear-x over which
CLEAR(x) holds".
```

The planner's output is a consistent temporal graph that includes the one given as input, and where every goal results from (has been causally explained by) the effects of planned actions.

This approach has some nice features: time is explicitly dealt with at the symbolic level, general qualitative relations can be expressed, intervals are very flexible (they are implicitly stretched or moved to allow insertion of other actions as planning progresses), synchronized, overlapping and parallel actions can be generated.

But the temporal logic of interval has also several drawbacks and restrictions:

- at the knowledge representation level: it is a strictly relative model that does not allows scheduled dates, deadlines, durations, or absolute delays (thus a solution given by this approach may not be feasible);
- at the programmer's level: a large number of intervals have to be explicited and described;
- at the planning level: the plan output is non conditional, i.e. it is a single projected future of what will happen if every thing goes well;
- at the problem solving level: the homogeneous representation of all knowledge as relations on temporal intervals leads to a very large graph, the management of which is an exponential process (that also requires a vast amount of memory); this does not reduce the usual complexity of planning (backtracking is still required), thus making the practicality of the approach questionable.

A recent paper [22] unifies that approach with that of [18] into a more rigorous theory (based on time points instead of intervals). It formalizes clearly the syntax and semantics of formulas in this temporal logic. However it does not address the above mentioned problems.

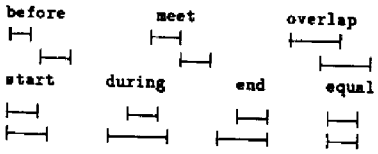
2.5 The Indexed Time Table approach to Planning and Execution Monitoring

This section introduces a knowledge representation formalism and structure, called an *Indexed Time Table* (IxTeT) that will enable us to express absolute and relative temporal relations between properties, events, goals and actions. It will be argued through simple examples and informal algorithms that the proposed representation is a promising approach towards filling the requirements previously set for planning and execution monitoring.

2.5.1 A formalism for actions, goals and events

We need to express intervals and time points, together with their relationships:

- qualitative relations between 2 intervals: before, meet, overlap, start, during, end, equal, and their inverses (after, is-met-by... see [2]):



- qualitative relations between an interval and a time point: before, during, start-at, end-at, after. Other relations can be expressed by combining those above, e.g.
  (disjoin A B) is equivalent to
  (or (before A B) (meet A B) (after A B))
  (start-before A T) is equivalent to
  (or (during A T) (end-at A T) (before A T))
- quantitative relations between 2 time points or extremities of intervals, e.g.
  (> (span (beginning Interv1) (termination Interv2))
  (span Interv3))
  Some absolute time references can be used, e.g.
  (clock Value-A), (end Timer-B).

Planning being a hierarchical process, we will rely on decomposition operators to specify the steps required to achieve a task. Such operators will be described as in the following example that defines a 3 actions and 2 pre-conditions operator:

```
(to (achieve (on ?x ?y))
    (when    (on ?x ?z))
    (do   steps  s1 (achieve (clear ?x))
                 s2 (achieve (clear ?y))
                 s3 (pickup ?x ?z)
                 s4 (holdmove ?x (position ?y))
                 s5 (putdown ?x ?y)
          such-that (meet s1 s3)
                    (meet s2 s5)
                    (meet s3 s4)
                    (meet s4 s5)))
```

-4

The "when" field specifies the context of decomposition; the "do" field can be a single action or task, e.g.

```
(to (achieve (clear ?x))
    (when (and (on ?y ?x) (clear ?z)))
    (do   (achieve (on ?y ?z))))
```

Actions and their affects are described independently of decomposition operators. For example:

```
(describe (pickup ?x ?y)
          (effects  met-by (clear ?x)
                    meet   (clear ?y)
                    end    (on ?x ?y))
          (duration 20))

(describe (putdown ?x ?y)
          (effects  met-by (clear ?y)
                    start  (on ?x ?y)
                    meet   (clear ?x))
          (duration 20))
```

A "when" field can be added to specify different effects and durations in various situations. The duration field can be any evaluable expression on the action variables. Quantitative delays can be added to temporal relations between an action and its effects. Note that the preconditions of an action that are not affected by it do not appear in its description: they are subtasks in a decomposition operator. This clean separation between decomposition operators and the description of actions (as in [23]) leads to a modular and powerful formalism (there are different ways to combine a "to" operator with its actions).

Expected events, scheduled or conditional, are expressed by the following operator:

```
(expect event
        (when context)
        (effects temporal-relation1 property1
                 temporal-relationN propertyN))
```

The context in the "when" field can be:

- a scheduled time, e.g. (starts-at (clock 0700)); or
- a conjunction of conditions that trigger the event together with temporal links to it.

A duration field can also be associated to an event.

Finally the required goals are defined by an achieve operator:

```
(achieve goals goal1 absolute-constraint1
               goalN absolute-constraintN
         such-that (temporal-relations between goals)* )
```

Absolute constraints are temporal relations between the required goals and known intervals or time points.

2.5.2 The Indexed Time Table

Temporal relations in the above formalism link implicit intervals; this make the knowledge easier to specify and read. But this knowledge cannot be used unless an internal representation make the corresponding intervals explicit: the same property may have several instances, each one can be true and false at different intervals (e.g. there are 6 instances of (clear ?u) in the above example). Thus an analysis of each decomposition operator and its actions is needed in order to explicit and match consistently the right intervals.

The output of this preprocessing analysis is represented as a time table, called an *OTT*, that summarizes action positions and truth values for all properties involved in the operator (effects and subgoals) along a sequence of symbolic time points. The time table corresponding to the previous example is given in *Figure 1*. Three contiguous actions, thus 4 time points are needed for this operator. Few simple temporal relations have been propagated to check possible matching of intervals, e.g. effect (clear ?x) of putdown cannot be the same interval as that of (clear ?x) of step s1 otherwise we would have a loop. All logical relations (non temporal) between properties involved in the operator are taken into account, e.g.
$\forall$ x,y (clear ?x) $\longleftrightarrow$ (not (on ?y ?x)). The usual hypothesis for negation by failure for closed world is assumed.

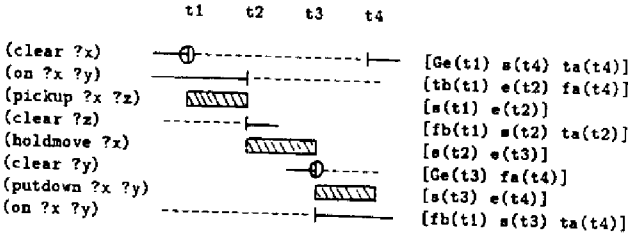


Figure 1

In this table a solid line denotes a true property, a broken line a false property, a rectangle is the interval of an action. End points of intervals are marked whenever known. Those corresponding to subgoals are indicated by circles. A complete symbolic representation of an *OTT* (given on the right of the figure) needs specification of truth values together with temporal relations. We use the symbols **tb, ta, fb, fa** for respectively true before, true after, false before and false after; **s** stands for start-at (was false, became true), **e** stands for end-at (from true to false), **G** denotes a subgoal. Notice for example how step s1 and the relation (meet s1 s3) is translated in the first row as **Ge(t1)**, whereas the 3rd effect of action **putdown** corresponds for the same row to **[s(t4) ta(t4)]**.

An important property of this structure: it is not allowed to introduce an interval in a row between 2 time points one of which being an **s** or an **e** not prefixed by a **G**, e.g. an new interval for (clear ?x) is not allowed between t1 and t4. The corresponding interval is said to be **non breakable** for that property.

An Indexed Time Table (*IxTeT*) is a structured organization of several *OTT*s, that is generated by a planner and corresponds to a complete plan together with its time structure. It starts initially from a time table describing current and past properties of the world, expected events and required goals together with their temporal relations. A goal is expanded using an appropriate decomposition operator, its *OTT* is inserted in the current *IxTeT*. This satisfies one or several pending goals (at least the expanded one), and introduces new subgoals to be expanded later.

The most important and difficult step here is the insertion of an *OTT* into an *IxTeT*. Time points have to be located with respect to those of the *IxTeT*: subgoals and effects in the *OTT* should be matched to properties in the *IxTeT* taking into account and checking several temporal and logical relations.

Let us illustrate some aspects of this process through a simple example (borrowed from [3]) using the above *OTT*. A 2 arms robot has to permute the first 2 parts of a stack of 3 parts, e.g. starting from

(and (on b a) (on a c)),

we require the un-ordered conjunction of 2 goals:

(and (on a b) (on b c)).

The initial *IxTeT* is thus (Ist and Last are first and last time points):

```
(on b a) : [tb(Ist) fa(Last)]
(on a c) : [tb(Ist) fa(Last)]
(on a b) : [fb(Ist) Gta(Last)]
(on b c) : [fb(Ist) Gta(Last)]
```

Goal (on b c) is expanded: an instance of the above OTT is simply inserted between time points Ist and Last, giving the sequence (Ist t1 t2 t3 t4 Last). Two interesting rows in the current $IxTeT$ are:

```
(clear b) : [tb(Ist) Ge(t1) s(t4) ta(t4) fa(Last)]
(on b c)  : [fb(Ist) fb(t1) s(t3) ta(Last)]
```

Replacing the goal (on b c) at Gta(Last) in the initial $IxTeT$ by ta(Last), and removing ta(4), makes the interval [t3 Last] non breakable: no other interval for this property can be inserted in it unless a backtracking in this goal decomposition is performed. Similarly subgoal (clear b), when later on expanded, will be immediately satisfied if no other interval for that row is inserted between Ist and t1 (note however that the interval [Ist t1] remains breakable since the goal is still pending).

In the next step goal (on a b) requires insertion of another instance of the same OTT (with time points t'1 through t'4):

- t'1 is characterized by the subgoal (clear a), this property is true in the current $IxTeT$ starting at t2, thus we should have: t'1 after t2 ;
- t'3 requires the subgoal (clear b) that is true either between Ist and t1 (but this will make the previously pending subgoal harder to meet), or after t4, thus : t'3 after t4 ;
- t'2 is not constrained by a subgoal, but it provides the effect (clear c) that is a pending subgoal at t3, thus: t'2 before t3 ;
- t'4 is just constrained to be between t'3 and Last.

Luckily in this example all insertion constraints can be met and lead to a completely ordered sequence of time points. The corresponding $IxTeT$ is given graphically in *Figure 2* (matched intervals have been drawn separately).

Remaining subgoals are immediately satisfied without further expansion. Notice how interactions between overlapping actions have been explicitly and quite simply taken into account (as a mater of comparison, the same example requires the explicit definition of 23 input intervals and propagation of 945 temporal constraints in the approach of [3]).

2.5.3 Planning with an $IxTeT$

The basic principles have been explained above along with the definition of the $IxTeT$. Several essential topics remain however to be developed. Let us go through some of them here.

How one should manage the agenda of pending goals ? The idea is to follow the hierarchy of decomposition: first all goals in the initial $IxTeT$, then those pending from the first level of decomposition, and so son. However if a goal can be immediately satisfied, without decomposition, by making an interval non breakable, we will leave it pending and put it back at the end of the agenda (unless all goals are in the same situation). This avoids constraining unnecessarily the $IxTeT$: a relation such as [tb(t) Ge(t')] is easily checked in the table, and the planner will chose if possible insertions that do not break it, otherwise nothing has been invested yet in decomposing this goal (and no backtracking will be required if it has to be decomposed).

Such strategy solves the classical problem where starting from:

(and (on c a) (clear b)), we require

(and (on c a) (on a b)).

Goal (on c a) will simply stay unexpanded until expansion of (on a b) needs (clear a), that in turn makes expansion of (on c a) necessary.

This strategy also enhancesthe planner robustness with regards to goals ordering in the agenda.

Another important issue is where in the $IxTeT$ to insert an OTT when there are several alternatives. In addition to the time position of the goal being expanded and to the subgoals and effects of the decomposition operator used, one has also to consider the decomposition context (the "when" field) that may restrict possible insertion points.

A first heuristic here is to leave whenever possible variables of the context non instanciated. This can be done for variables that are required for achieving the OTT subgoals. For example the "to" operator for (clear ?x) can be used with variable ?x non instanciated (but it requires variable ?y). This again avoids unnecessary constraints in the $IxTeT$: remaining variables are opportunistically matched when needed to satisfy a pending subgoal.

A second heuristic in this issue is to introduce only the constraints that may solve pending subgoals. Most of the time this should lead to an $IxTeT$ whose time points are partially but not completely ordered.

The efficient management of this partial order relies on one of the two indexing schemes proposed for the $IxTeT$ approach. Let us introduce an example using the above OTT. Starting from
(and (on a b) (on a d) (clear c)) we require
(and (on b c) (on a b)) . Expansion of the first goal is done as before. The 2nd instance of the OTT for goal (on a b) is however inserted in the current $IxTeT$ with less constraints:

- t'1 requires (clear a) : t'1 is before t2
- t'3 requires (clear b) : t'3 is after t4
- t'2 does not provide any needed effect, its is not constrained by this insertion; neither is t'4. We have thus the following partial order (*Figure 3*):

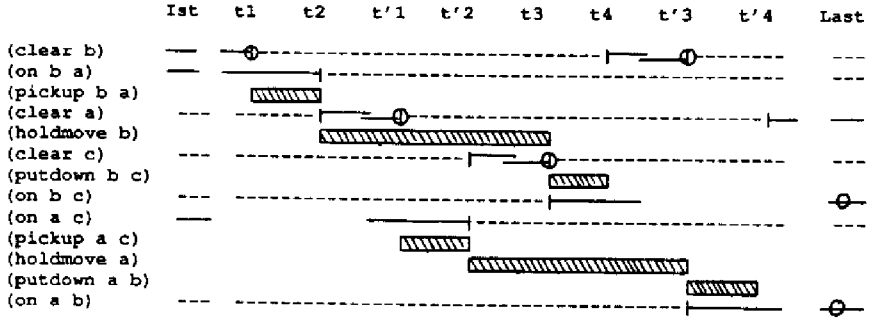


Figure 2

t-6

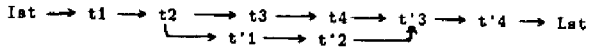


Figure 3

Two interesting rows in the corresponding *IxTeT* (that cannot be drawn graphically) are given bellow:

```
(clear b) : [tb(Ist) Ge(t1) e(t4) Ge(t'3) fa(Lst)]
(clear a) : [fb(Ist) fb(t1) e(t2) Ge(t'1) e(t'4) fa(Lst)]
```

The important property here is, although the *IxTeT* time points are partially ordered, those used in any given row correspond to a complete order. From the point of view of a single property, time is non ambiguously seen as a simple sequence. We know when the property is required to be true or false, when a new interval can be inserted for it, and whether it corresponds to a pending goal that can be readily satisfied or that request expansion.

The algebraic structure of a set of time points in an *IxTeT* is thus a lattice. It can be represented as a directed acyclic graph with a root (Ist) and a sink (Last). Two basic operations have to be performed on this time lattice:

- finding if two points are ordered and their relative position;
- inserting time points and links consistently (i.e. without loops).

The obvious method for both operations relies on path finding, but it is too costly for such a heavy use. The proposed approach relies on a hierarchical indexing scheme.

Let R be any convenient range of numbers (integers or reals) with a conventional Min and Max; i, j, $i_1 \ldots, i_k$ are elements of R. Each time point t in the lattice is given an index: a sequence noted $t(i_1.i_2 \ldots i_k)$, of one or several elements of R. Ist and Last are indexed Ist(Min) and Last(Max). The rest of the lattice is indexed such that if t is immediately before t' (no other point is between t and t') then:

- either $t(i_1. \ldots .i_k.j)$ and $t'(i_1. \ldots i_k.j')$ are such that $j < j'$ and there is no other index $(i_1. \ldots i_k.j")$ in the lattice with: $j < j" < j'$
  (i.e. t' is the next sibling of t in the index hierarchy) ;
- or $t(i_1. \ldots .i_k)$ and $t'(i_1. \ldots i_k.j)$ are such that there is no index $i_1..i_k.j'$ with: $Min < j' < j$
  (i.e. t' is the first son of t in the hierarchy).

For example we would have (*Figure 4*):

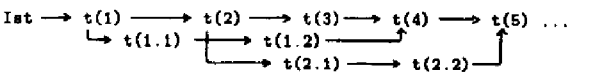


Figure 4

Indexes as stored as a tree with a pointer from each index to the corresponding time point. A node in the lattice has pointers to its immediate successors.

The relative position of $t(i_1. \ldots .i_n)$ and $t'(j_1. \ldots .j_m)$ is found as follow:

- if $(n < m)$ and $(i_1. \ldots .i_{n-1}) = (j_1. \ldots .j_{n-1})$ and $(i_n < j_n)$ then t is before t' (e.g. t(3.4.5) is before t(3.4.7.2) ; t(6) is before t(9.1.3)), in this case t and t' are said to be directly comparable, and we note $t < t'$
- otherwise successor nodes of t are followed along the same level of the indexing hierarchy until a node t" at a higher level is found:
  - if $t" < t'$ then t is before t'
  - if $t' < t"$ then nothing can be said: we restart from t' and try to compare its higher successors to t
  - otherwise the path from t" is pursued higher up until a direct comparison can be made.

If no direct comparison can be made at the highest indexing level up from t, then we try up from t'. If it does not succeed then t and t' are not ordered.

A direct comparison involve few operations (of constant complexity). A path up in the indexing hierarchy is of logarithmic length. Thus it should be a very fast algorithm. The same procedure provides a loop checking test: there is a cycle between t and t' iff t is before t' and t' is before t.

More work is needed to insert time points and links in the lattice while keeping the indexing structure as defined. The main idea is to have nodes in the lattice as high as possible in the indexing hierarchy (to "flatten" up the lattice). For example we may want to insert in the previous lattice (*Figure 4*) a time point t after t(2.2) and before t(1.2). Previously to this insertion we had the path
$t(2.2) \longrightarrow t(5)$, after the insertion we would have
$t(2.2) \longrightarrow t \longrightarrow t(1.2) \longrightarrow t(4)$. But $t(4) < t(5)$: the insertion adds a constraint into the lattice that has to be taken into account by changing pointers and indexes. Pointer t(2.2) is removed (less constrained), t is indexed t(2.3), t(1.2) is re-indexed t(2.4), thus giving (*Figure 5*):

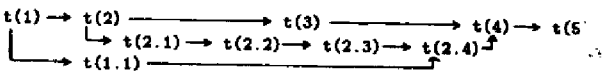


Figure 5

The general procedure for inserting a link (or a sequence of nodes) between t and t' would be:

- if t' is before t then the link cannot be inserted (it would make a loop);
- if the path up from t goes through a node t" such that t' is before t" then the link adds a constraint in the lattice, nodes in the 2 involved pathes and their offsprings have to be re-indexed;
- otherwise we just add the new link or sequence of nodes as offsprings of t up to t' (if $t(i_1. \ldots .i_k)$ has already a son node $t"(i_1. \ldots .i_k.i_{k+1})$, we just add a dummy node indexed $(i_1. \ldots .i_k.i_{Min})$ between them and make it father of the newly inserted nodes).

The complexity of this procedure has not yet been analyzed.

With the time lattice being defined we are now ready to take into account absolute constraints on the duration of actions and position of time points for goals and events. The lattice is an additive graph with durations as positive or null labels on arcs. The maximum cumulative sum of arc labels between t and t', over all existing pathes from t to t', corresponds to the minimum required time to go from t to t'. This minimum will eventually be compared to absolute constraints attached to the 2 time points t and t'. A non satisfied constraint will lead to rejecting the faulty path. PERT-like technics such as those implemented in DEVISER [25] will be used to check the consistency of each insertion of an *OTT* into the *IxTeT*. A good processing of the uncertainty and imprecision necessarily attached to such information has however to be added to such technics.

A final point about the structure of an *IxTeT*: its temporal dimension is indexed through the time lattice, it would improve the planner efficiency to also index its second dimension. We would like to find easily the rows to look into and those to be added to an *IxTeT*, while inserting an instance of an *OTT*. We would also need to have all logical relations concerned by those rows, and the rows that will be

checked through those relations. The indexing scheme proposed here relies on a unification tree, a structure that results from the compiling of a set of patterns [14].

2.5.4 Execution Monitoring with an *IxTeT*

This topic will be discussed in more detail in the next section, in the context of a flexible assembly cell application. Let us survey briefly here what could be the main benefits for monitoring a plan expressed as an *IxTeT*.

As time advances, the lattice is reduced to an ordered sequence. This is achieved either:

- through an opportunistic choice made by the Execution Monitor, or
- through the occurrence of an event.

For example we just passed time t. In the *IxTeT* t is followed in any order by:

- t', defined as the end of action A synchronized with the beginning of action B; and
- t", defined as the occurrence of a scheduled event.

If t' happens before t", the corresponding relation is propagated through the 2 pathes. Eventually this will add constraints in the lattice, and resolve future choices.

Another issue is when to expect a conditional event: the indexing of the *IxTeT* enables to find easily when a conditional event may occur (its effects have been taken into account at planning level). The observed (or chosen) ordering of time points non previously ordered will enable to tell (or decide) if the conditions triggering the event do or do not hold simultaneously.

How an *IxTeT* can be of any help to reacting to an unexpected event? We assume that such event is relevant to the task at hand but does not require postponing or canceling current goals. The event and its logical consequences are put into the *IxTeT* at the current time. Any contradiction with what was expected is compared to the past and to the previously projected future:

- properties required later for a forthcoming action or goal but that are not true any more (were true initially or resulted from some previous action) should be considered as new subgoals inserted in the rest of the *IxTeT* and decomposed by the planner;
- properties that were planned to hold only later on, as a result of some forthcoming actions, will lead to reconsider the need of such actions and those before them.

Detailed examples will be given in the following section.

3 A Structured Environment Application: The NNS project

In this section, we discuss how the proposed approach could be used to tackle problems of *Task Planning* and *Execution Control* for a project developed at LAAS that represents a structured environment application: the NNS project [5] [6] [10].

The NNS project aims at developing methods and tools for the programming and the robust execution control of complex manipulation tasks to be performed by a multi-robot flexible assembly cell.

Clearly, such a system involves decision-making capabilities available off-line and on-line. The originality of our system lies mainly in the fact that the *Off-Line System*, instead of producing a "rigid" program, generates a more "flexible" structure called *Task Execution Model* that will be used on-line in order to take into account sensor information, to interpret what happens in the workcell and to react to various asynchronous events *(Figure 6)*.

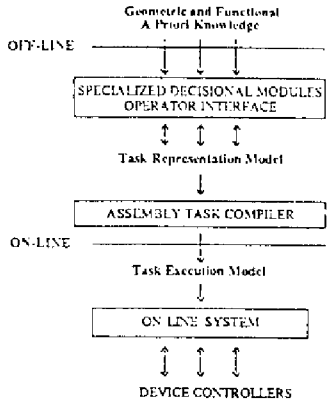


Figure 6: NNS organization and Knowledge Representation

The *Off-Line System* includes various planners based on geometric reasoning in order to structure the space of the workcell, to synthesize the various actions that can be executed [17] [24] and to provide rules for action selection and scheduling [1].

The *On-Line System* is a *Knowledge-Based System* [11]; it uses the *Task Execution Model* in order to completely instanciate and schedule the actions to be performed given the current state of the workcell, as well as to control their execution in a multi-agent environment.

We will restrict ourselves, in the following, to the problem of Action Scheduling and Execution Control.

3.1 Task Modeling

The highest level of abstraction used by our system for an assembly task is modeled as follows: the workcell is supplied with several primary elements and delivers products according to a predefined process.

Example: The assembly task concerns three types of parts: *A*, *B* and *C*. Parts *A* are introduced by the conveyor 1. Parts *B* and *C* are introduced in random order on conveyor 2. The workcell produces sub-assemblies *BA* (*B* on *A*) and *BC* (*B* on *C*). The sub-assembly *BC* must be inspected before unloaded. Defective parts are put in a specific storage.

We introduce three concepts that allow to specify an assembly task at this level of abstraction: the *identity of part*, the *site* and the *posture* of a *part* in a *site*.

The identity of a part corresponds to a specific step in an assembly process (primary part, sub-assembly...)

Example:

- the identity "part A" corresponds to a part of type *A*.
- the identity "?(partA1 partA2 ... partAn)" corresponds to a part of type *A1* or *A2* or .. or *An*. This is an arbitrary notation to represent parts that belong to a class of parts but that are

4.8

not yet identified. This case may arise after a feeding operation if parts arrive in random order, after an identification operation that does not discriminate completely the identity of a part or after the analysis of a failure that makes the identity of a part doubtful.

A *site* corresponds to a place in a workcell which is intended to hold parts (such as a robot gripper, a storage place on a table or on a conveyor belt...). A symbolic name is attached to each site (*rob1, sto1, conv1*). The parts present in a cell are always in one of its sites. A site contains only one part at a time. Sites may be fixed or may move in a cell. In the case of sites representing conveyors, these sites may be AWAY.

A *part posture* provides a qualitative description of the geometric and dynamic constraints that achieve stable position of a part. On *Figure 7* different postures of part *B* are represented.

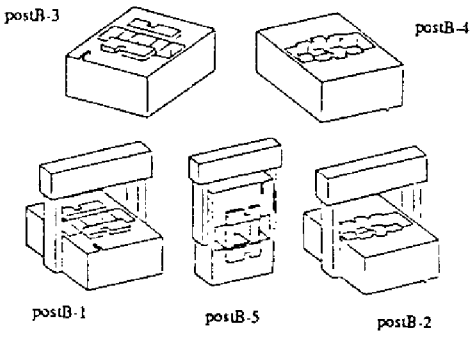


Figure 7 : Different Postures of part B

Each part has a discrete number of possible (and useful) postures. It is limited by the devices and the tools available in a cell and by the set of part postures imposed to perform operations. A number is associated to each part posture: *PostB-1, PostB-2...*

The three basic concepts of *site*, *identity* and *posture* constitute the major ingredients that allow us to specify and to represent an assembly task. The *Task State* is represented by the state of all the sites in the workcell:

$$< TaskState > ::= (< Site >< SiteState >)^*$$

$$< SiteState > ::= < part >:< posture > \mid EMPTY \mid AWAY$$

Example: *Figure 8* shows the different sites in a cell and their role in the assembly task introduced earlier.

♠ *cnv1*: conveyor belt for loading parts *B* and *C*; this site is equipped with a camera for part identification and localization;

♠ *cnv2*: conveyor for loading parts *A*;

♠ *cnv3*: conveyor for unloading sub-assemblies *BA* and *BC*;

♠ *garb*: storage where defective parts *BC* (noted *BC-*) are rejected;

♠ *rob1* and *rob2*: robot grippers;

♠ *asm1*: site for the part-mating of *A* and *B*;

♠ *asm2*: site , equipped with a camera, for the part-mating of *B* and *C* and the inspection of the result;

♠ *sto1, sto2, tab1*: storage sites.

Actions are defined as operators that change the *Task State*. An action either modifies the identity of a part present in a site, or transfers it from one site to another site, changing its posture. There are several action classes that can be represented as follows:

Feeding

$$(site\ AWAY) \mapsto (site\ partX : postX)$$

Inspection

$$(site\ partX : postX) \mapsto \begin{array}{l}(site\ partX^+ : postX^+)\\ or\\ (site\ partX^- : postX^-)\end{array}$$

Part-Mating

$$\begin{array}{l}(site1\ partX : postX_1)\\ (site2\ partY : postY_2)\end{array} \mapsto \begin{array}{l}(site1\ partXY : postXY_3)\\ (site2\ EMPTY)\end{array}$$

Pick or Place

$$\begin{array}{l}(site1\ partX : postX_1)\\ (site2\ EMPTY)\end{array} \mapsto \begin{array}{l}(site1\ EMPTY)\\ (site2\ partX : postX_2)\end{array}$$

3.1.1 Action Planning

As it has been emphasized earlier, a robotics workcell that performs repeatedly over time the same assembly task is an a priori known environment (though very difficult to model and to structure) where most problems - or at least those that need heavy decisional processes - can be addressed off-line. Yet, there remain decisions that must be taken on-line in order to provide a robust behavior and a better efficiency to the system.

For instance, action planning cannot be done completely off-line because of:

- the presence of non-deterministic actions (use of sensor information);
- the occurrence of asynchronous events (failures, random arrival of parts, interactions with the shop level..);
- the difficulty to evaluate precisely the duration of actions particularly in the case of a multi-agent system.

However, it is not useful to have a "complete" and "permanent" planning system on-line, that will, again and again, rediscover interrelations between actions, goals... Such a situation is useless and very time-consuming particularly in the case of a multi-agent system.

For all these reasons, the *Task Execution Model*, produced by the *Off-Line System*, contains what we call a *Task Plan*; it is a scheme of plans that will reduce the planning activity of the *On-Line System*. Note that the planning activity is reduced not suppressed. This is a key issue as it emphasises an interesting link between *Planning* and *Execution Control*. The *Planner* prepares the decisions using a global knowledge of the task. The *Execution Controller* has a planning activity based on this knowledge; it generates "small plans" in a close interaction with the actual environment.

There exists a first version of the software that implements these ideas. It has been used, with reasonable performance, to program and control a complex assembly task involving 4 types of parts on a two-robot assembly cell equipped with various sensors. The reader may refer to [1] [11] [9] [10].

Our intent here below is to analyse how the temporal structures and the formalisms, presented in section 2, could be used in such a system to overcome some of its limitations by introducing an explicit representation of time in the planning process off-line and on-line.

3.1.2 The Task Plan

The *Task Plan* is a part of the *Task Execution Model*; it is used on-line to determine the set of "admissible" actions that can be performed

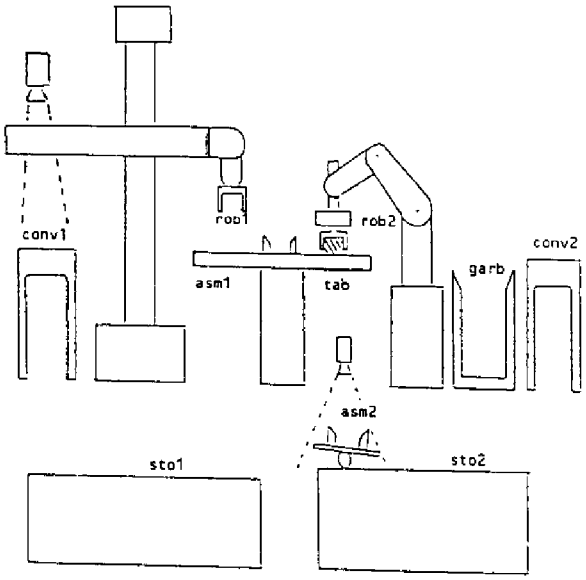


Figure 8: Spatial Organisation of a Cell

from any *Task State*. An action is "admissible" if it is feasible from the current *Task State*, relevant (the action is necessary and other actions that are "better" are not feasible) and if the action does not lead to a "deadlock" situation or a situation where the task has to "backtrack".

The *Task Plan* representation is not explicit: the number of possible task states is finite but very important, if we consider that asynchronous events may occur during the execution. The *Task Plan* consists of a set of rules. Each rule represents a condition on the Task State that makes an action admissible. All rules are terminal and made of simple propositional forms without quantified variables. They can be compiled into a decision tree according to the techniques described in [15].

Using temporal relations, it will be possible to analyse the *Task State* at various moments; this will lead to more subtle conditions like: *Action* i *is admissible if the situation* sj *is verified in a "close" future.* We give here an example of a rule corresponding to the *Task Plan* of the assembly task described above.

Example: If a part *A* is stored in *sto1* and there is no expected arrival of another part *A* on *cnv2* for a given duration (*DELTA2 time units*), and if the site *asm1* will be empty in a close future (less than *DELTA1 time units*), then the action 26 of picking part *A* from *sto1* is "admissible" (*"at"* represents the time at which the action will be executed)

```
(if ((sto1 A:3)
     (rob2 EMPTY)
     (< (span *at* (begin (asm1 EMPTY))) DELTA1)
     (> (span *at* (end (cnv2 AWAY))) DELTA2))
 then
    ((ADMISSIBLE action-26)))
```

Other conditions can be taken into account. For instance, two actions that make use of the same tool cannot overlap.

The effects of the actions are provided as follows:

```
(describe (pick ?site1 ?site2 ?part ?post1 ?post2)
      (effects
            met-by (?site1 ?part : ?post1)
            met-by (?site2 EMPTY)
            meet   (?site1 EMPTY)
            meet   (?site2 ?part : ?post2)))
```

Note that value of the properties is changed somewhere between the beginning of an action and its end. However, at the level of action planning, the states of the sites are "protected" during all the duration of an action. Lower levels manipulate a more precise description of actions. For example, a Pick action is composed of several steps: gross motion and preparation of the tools, approach motion, grasping operation, disengage motion. The change of state is situated in a small interval - called "uncertainty interval" - where we cannot define what is exactly the state. This description is used by the *Execution Controller* in order to monitor the task state.

Starting from this general description, the *Off-Line System* will instanciate specific description for each action taking into account information such as the effectors to be used, the position of the sites... in order to determine the duration of the action (for some actions, it is possible to do it off-line) or a minimum and a maximum bound of it.

We will show later how all this knowledge is used on-line to produce *Execution Plans* that describes the sequences of actions to be performed with respect to the actual workcell state.

3.1.3 The Task Plan construction or "compilation phase"

The *Assembly Task Compiler* is the last step in the *Off-Line System*. It generates the *Task Execution Model* which contains the *Task Plan*. It performs various processings, like verifying that no information is missing, that there is a way for each part entering the cell to be processed..

For this purpose, a graph representation is built for the assembly task. It describes all the actions that can be performed on parts. This graph represents the possible evolutions of each part independently *(Figure 9)*. Nodes are site states; arcs are oriented and represent actions.

This graph illustrates various difficulties encountered in action planning:

- Problems of parallel execution: several parts, even parts of the same type, may "traverse" the graph at the same time. Various actions can be executed in parallel as we assume a multi-agent system. Some actions cannot be performed simultaneously : they use the same resource (tool, sensor, space region ...)
- Special care must be taken in order to manage appropriate "critical resources" like sites linked to manipulators;
- Temporal relations and estimation of action duration (depending on the context) are clearly very useful in such applications it should be interesting to take into account the fact that conveyor cannot stay idle more than a specified time...
- It could be also useful to preselect paths for parts, that could be chosen under certain preconditions (normal functioning, periodical arrival of parts...)

4.10

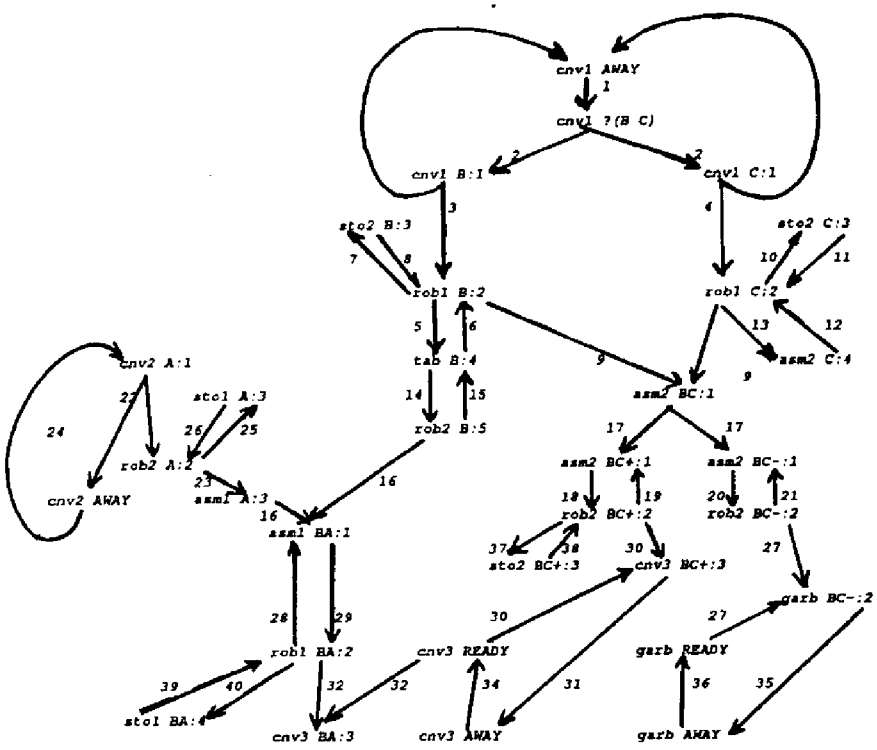


Figure 9: Assembly Task Graph

3.1.4 Execution Control

An *Execution Plan* is a particular set of action sequences - among possible sequences implicitly contained in the *Task Plan* - together with synchronization constraints between these sequences. This plan selection and instanciation will take place at the completion of non deterministic actions, after feeding and unloading actions and when a failure arises that causes a mismatch between the actual state and the state expected in the current *Execution Plan*.

In the current implementation, a new *Execution Plan* is generated as if the cell was not working; its initial state is the state that will be obtained if the the current *Execution Plan* is completely achieved. We would like to proceed more smoothly by taking into account the current plan i.e. "merging" a new *Execution Plan* plan with the current one.

An example: In the first situation, the cell activity is stopped (no action is currently performed) and we have the current *Task State*:

ES0 = (... (cnv B:1) (sto1 A:3) (asm2 C:4) (rob1 EMPTY) (rob2 EMPTY) ...)

(the two robots are empty, a part *B* is available on the conveyor belt *cnv1*, a part *A* and a part *C* are respectively stored in *sto1* and in *asm2*)

The *Execution Controller* constructs an *IxTeT* (partially) illustrated in *Figure 10a* and *Figure 10b*.

Time points for the beginning and the end of action i will be noted $si/j/k$ and $ei/j/k$ where j is the number of the *Execution Plan* et k is a complementary index (used only in the case when an action appears more than one time in a plan). The time point representing the beginning of the *Execution Plan* number j is noted $1st/j$.

The Execution Plan that has been produced contains the sequences: [3-5-14-16-29...], [26-23] and [1-2]. The temporal structure of this plan is given in *Figure 10b*.

Note that:

- In constructing the table after action 3, the *Execution Controller* has chosen action 5 while action 9 was also admissible; this is because of a task-dependent heuristic;
- the set of actions 26-23 could be placed before or after 14; as the action 23 has to be placed before action 16 (rendes-vous for part-mating), the reasonable choice has been made;
- the manipulator linked to *rob1* does nothing between *e(5/0)* and *s(29/0)*;
- a new planning step is expected after the execution of action 2 (identification of parts provided in random order by conveyor 1); this is represented in *Figure 10a* by an arrow.

The execution begins. Actions 3/0 and 26/0 can be started in parallel. Let us suppose that action 2/0 finishes with the identification of a new part of type B while actions 5/0 and 23/0 are still executing.

If we assume that the planning process is sufficiently fast and that we can estimate a time bound for its duration, then it is possible to consider a "scenario" where a complementary plan is "merged" with the plan under execution. This assumption is reasonable because of the fact that the planning activity of the *Execution Controller* is based on decisions prepared off-line.

The *Execution Controller* would then generate the *IxTeT* presented in *Figure 11a* and *Figure 11b*.

Note that:

- actions 3/1 and 9/1 have been inserted between 5/0 and 29/0;
- two actions will cause a new plan to be constructed; action 2/1 and action 17/1.

A lot of work remains to be done in order to validate the ideas and the mechanisms sketched here. However, let us summarize here below some key issues under investigation. The availability of a temporal structure for plans and of a flexible projection in the future will allow a better control of the cell.

An important issue is to be able to permanently verify the relevance of plans (or sub-plans) under execution. The *Execution Controller* has to decide when to "merge" a new plan with a plan already in execution, when to replan completely... Task dependent heuristics should be prepared off-line in order to allow the *Execution Controller* to evaluate a situation and to react to it efficiently.

References

[1] R. Alami and H. Chochon. Programming of flexible assembly cells: task modeling and system integration. In *IEEE, International Conference on Robotics and Automation, St Louis (USA)*, March 1985.

[2] J. F. Allen. An interval-based representation of temporal knowledge. In *7th IJCAI, Vancouver (Canada)*, August 1981.

[3] J. F. Allen. Planning using a temporal world model. In *8th IJCAI, Karlsruhe (FRG)*, August 1983.

[4] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[5] ARA. *Journées Annuelles du Programme ARA, Toulouse (France)*. September 1984.

[6] ARA. *Journées Bilan du Programme ARA, Paris (France)*. June 1986.

[7] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[8] R. Chatila. Mobile robot navigation: space modeling and decisional processes. In *Robotics Research 3, Faugeras and Giralt (Eds), MIT Press*, 1986.

[9] H. Chochon. *Programmation de tâches d'assemblage robotisées: modélisation et processus décisionnels.* Thèse de l'Université Paul Sabatier, Toulouse (France), Laboratoire d'Automatique de d'Analyse des Systèmes (C.N.R.S.), November 1986.

[10] H. Chochon and R. Alami. A knowledge-based system for programming and execution control of multi-robot assembly cells. In *'87 International Conference on Advanced Robotics (ICAR), Versailles (France)*, October 1987.

[21] E. D. Sacerdoti. *A Structure for Plans and Behaviour*. Elsevier North-Holland, 1977.

[22] Y. Shoham. Logics in AI: semantical and ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.

[23] A. Tate. Generating project network. In *5th IJCAI*, August 1977.

[24] J. M. Valade. Geometric reasonning and synthesis of assembly trajectory. In *'85 International Conference on Advanced Robotics (ICAR), Tokyo (Japan)*, September 1985.

[25] S. A. Vere. Planning in time: windows and durations for activities and goals. *IEEE Transactions PAMI*, 5(3), May 1983.

[11] H. Chochon and R. Alami. NNS, a knowledge-based on-line system for an assembly workcell. In *IEEE, International Conference on Robotics and Automation, San Fransisco (USA)*, April 1986.

[12] R. E. Fikes, P. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4), 1972.

[13] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 1971.

[14] M. Ghallab. Coping with complexity in inference and planning systems. In *Robotics Research 3, Faugeras and Giralt (Eds) MIT Press*, 1986.

[15] M. Ghallab. *Optimisation de processus décisionnels pour la Robotique.* Thèse d'état, Université Paul Sabatier, Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), Toulouse (France), October 1982.

[16] G. Giralt, R. P. Sobek, and R. Chatila. A multi-level planning and navigation system for a mobile robot: a first approach to HILARE. In *6th IJCAI, Tokyo (Japan)*, August 1979.

[17] L. Gousènes. Strategies for solving collision-free trajectories problems for mobile or manipulator robot. *International Journal of Robotics Research*, 3(4), Winter 1984.

[18] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6, 1982.

[19] N. J. Nilsson. *Triangle Tables: a proposal for a robot programming language.* Technical Note 347, AI Center, SRI, 1985.

[20] J. F. Picardat. *Contrôle d'exécution, compréhension et apprentissage de plans d'action: Développement de la méthode de table triangulaire.* Thèse de l'Université Paul Sabatier, Toulouse (France), Université Paul Sabatier, 1987.

4.12

Figure 10.a

rob1 EMPTY	ta(1st/0)	tb(s3/0)-fa(e3/0)	fb(s5/0)-ta(e5/0)	tb(s29/0)-fa(e29/0)
rob1 B:2	fa(1st/0)	fb(s3/0)-ta(e3/0)	tb(s5/0)-fa(e5/0)	
rob1 BA:2	fa(1st/0)	fb(s29/0)-ta(e29/0)	.....	
rob2 EMPTY	ta(1st/0)	tb(s26/0)-fa(e26/0)	fb(s23/0)-ta(e23/0)	tb(s14/0)-fa(e14/0)
rob2 A:2	fa(1st/0)	fb(s26/0)-ta(e26/0)	tb(s23/0)-fa(e23/0)	
rob2 B:5	fa(1st/0)	fb(s14/0)-ta(e14/0)	tb(s16/0)-fa(e16/0)	
tab EMPTY	ta(1st/0)	tb(s5/0)-fa(e5/0)	fb(s14/0)-ta(e14/0)	
tab B:4	fa(1st/0)	fb(s5/0)-ta(e5/0)	tb(s14/0)-fa(e14/0)	
sto1 EMPTY	fa(1st/0)	fb(s26/0)-ta(e26/0)		
sto1 A:3	ta(1st/0)	tb(s26/0)-fa(e26/0)		
asm EMPTY	ta(1st/0)	tb(s23/0)-fa(e23/0)	fb(s29/0)-ta(e29/0)	
asm A:4	fa(1st/0)	fb(s23/0)-ta(e23/0)		
asm BA:1	fa(1st/0)	fb(s16/0)-ta(e16/0)	tb(s29/0)-fa(e29/0)	
asm2 EMPTY	fa(1st/0)			
asm2 C:4	ta(1st/0)			
cnv1 AWAY	fa(1st/0)	fb(s3/0)-ta(e3/0)	tb(s1/0)-fa(e1/0)	
cnv1 ?(B C)	fa(1st/0)	fb(s1/0)-ta(e1/0)	tb(s2/0)-fa(e2/0)	
cnv1 B:1	ta(1st/0)	tb(s3/0)-fa(e3/0)	fb(s2/0) $\Longrightarrow$	
cnv1 C:1	fa(1st/0)	fb(s2/0) $\Longrightarrow$		
cnv3 AWAY	fa(1st/0)			
cnv3 READY	ta(1st/0)			
cnv3 BA:3	fa(1st/0)	.....		

Figure 10.b

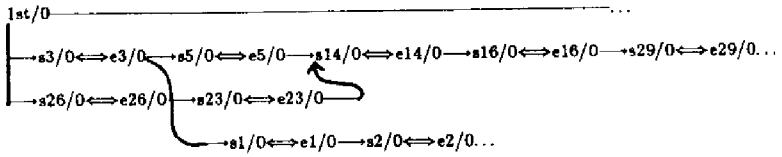


Figure 10: Indexed Time Table for Execution Plan 0

Figure 11.a

rob1 EMPTY	-ta(e5/0)	tb(s3/1)-fa(e3/1)	fb(s9/1)-ta(e9/1)	tb(s29/0)-fa(e29/0)
rob1 B:2	-fa(e5/0)	fb(s3/1)-ta(e3/1)		
rob1 BA:2	fa(1st/1)	fb(s29/0)-ta(e29/0)	.....	
rob2 EMPTY	-ta(e23/0)	tb(s14/0)-fa(e14/0)	fb(s16/0)-ta(e16/0)	
rob2 A:2	-fa(e23/0)			
rob2 B:5	fa(1st/1)	fb(s14/0)-ta(e14/0)	tb(s16/0)-fa(e16/0)	
tab EMPTY	-fa(e5/0)	fb(s14/0)-ta(e14/0)		
tab B:4	-ta(e5/0)	tb(s14/0)-fa(e14/0)		
sto1 EMPTY	fa(1st/1)	fb(s26/0)-ta(e26/0)		
sto1 A:3	ta(1st/1)	tb(s26/0)-fa(e26/0)		
asm EMPTY	-fa(e23/0)	fb(s29/0)-ta(e29/0)		
asm A:4	-ta(e23/0)			
asm BA:1	fa(1st/1)	fb(s16/0)-ta(e16/0)	tb(s29/0)-fa(e29/0)	
asm2 EMPTY	fa(1st/1)			
asm2 C:4	ta(1st/1)	tb(s9/1)-fa(e9/1)		
asm2 BC:1	fa(1st/1)	fb(s9/1)-ta(e9/1)	tb(s17/1)-fa(s17/1)	
asm2 BC+:1	fa(1st/1)	fb(s17/1) $\Longrightarrow$		
asm2 BC-:1	fa(1st/1)	fb(s17/1) $\Longrightarrow$		
cnv1 AWAY	fa(1st/1)	fb(s3/1)-ta(e3/1)	tb(s1/1)-fa(e1/1)	
cnv1 ?(B C)	fa(1st/1)	fb(s1/1)-ta(e1/1)	tb(s2/1)-fa(e2/1)	
cnv1 B:1	ta(1st/1)	tb(s3/1)-fa(e3/1)	fb(s2/1) $\Longrightarrow$	
cnv1 C:1	fa(1st/1)	fb(s2/1) $\Longrightarrow$		
cnv3 AWAY	fa(1st/1)			
cnv3 READY	ta(1st/1)			
cnv3 BA:3	fa(1st/1)	.....		

Figure 11.b

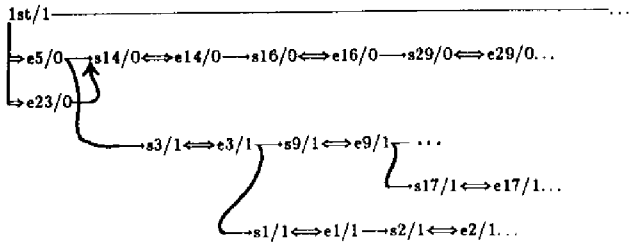


Figure 11: Indexed Time Table for Execution Plan 1

PLANNING WITH NON-DETERMINISTIC EVENTS FOR ENHANCED ROBOT AUTONOMY

Jérôme Perret - Rachid Alami*

**LAAS/CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cédex, France*

Abstract. *Beyond pure academic research, a number of application fields raise a challenge. In activities such as planetary exploration, undersea servicing, work in nuclear plants, or disaster intervention, the dynamics of the environment and the strong constraints on data transmissions call for advanced robot autonomy. In this paper we propose a cooperative robot control system architecture based on a reaction planner. We develop our representation for action and event operators and present a simple algorithm for building robust plans. Finally, we discuss plan execution and give hints of possible future developments.*

Key Words. Robotics, Reaction Planning, Autonomous Systems, Plan Execution

1. INTRODUCTION

An autonomous robot system operating in an environment in which there is uncertainty and change needs to combine reasoning with reaction. Reasoning means the ability to decide what to do; the reasoning activity we are interested in here is planning, as the action of projecting the current situation into the future in order to figure out a sequence of actions leading to a goal. Reaction means the ability to act in order to avoid the negative consequences of a situation, or on the contrary to benefit from its positive effects.

The choice of the "good" reaction is usually not trivial, and might require planning. However, in the case of an automatic planning system, the time needed to produce a plan can be very long, which contradicts the notion of "reaction" itself.

In this context, several approaches have been proposed, which give a partial solution to the problem. We will distinguish here three main strategies:

Anytime planning: (Dean and Boddy, 1988) Anytime algorithms have the ability to produce a plan within a given bounded time compatible with the necessary reaction. This approach tries to bring reasoning nearer to reactivity. The price to pay for this is the poor quality of the plan when the time allocated to planning is short. Moreover, in order to build a complete control system, there is a need for a meta-supervision level in charge of deciding when to plan and how much time to allocate to planning in a given situation.

Probabilistic planning: (Kushmeric *et al.*, 1993) The purpose of this approach is to plan for a sequence of actions which, whatever the initial state of the environment and whatever its evolution, will lead to the goal with a sufficiently high probability. In that way, uncertainty in the environment is accounted for, and the control system does not have to rely heavily on the poor perception capabilities of the robot. The problem is the existence of such plans, and the complexity of their construction.

Reaction planning: (Thiébaux and Hertzberg, 1992) Reaction planning means to foresee the changes in the environment and to produce reactions in advance. The strong hypothesis in this approach is the complete knowledge of the world and its possible evolution, and of the consequences of the robot's actions.

Finally, a number of recent works try to bring these approaches together and associate them (Drummond and Bresina, 1990). However, they hit the full complexity of the problem and are forced to make compromises.

Our claim is that reasoning and reaction are two activities which must remain distinct, and cooperate. In that way, we definitively place ourselves in the third strategy, reaction planning. The key point is the interaction between the two processes, and previous works defer largely there.

In section 2, we will discuss this interaction, and establish what conditions it imposes on produced plans. In section 3 we will describe the class of problems we are interested in here. In section 4 we

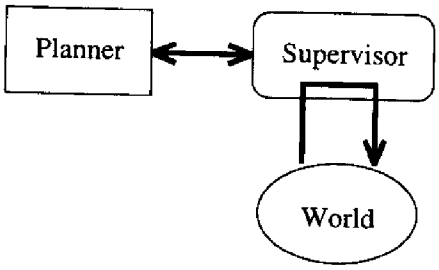


Figure 1 The Planner-Supervisor paradigm

will present our planning operators and in section 5 we will propose a planning scheme for producing plans which satisfy the conditions introduced in section 2. Section 6 will deal with plan execution. Finally, we will give an overview of our future work and a short conclusion.

2. DISCUSSION

Our system design is simple and now rather commonplace (see fig. 1). Several authors use the same description, such as (Fikes *et al.*, 1972), (Drummond and Bresina, 1990), (Kabanza, 1992), (Musliner *et al.*, 1992).

The responsibility of "closing the loop" at the level of plan execution control is entirely devoted to the supervisor. In order to achieve it, the supervisor makes use only of deliberation algorithms which are guaranteed to be time-bounded and compatible with the dynamics of the controlled system. Indeed, all deliberation algorithms which do not verify this property are actually performed by the planner upon request of the supervisor.

We are primarily interested here in the planning module, so we will develop the conditions which the plans have to fulfil, and we will come back to the supervision at the end of the article.

We say that, in order to insure the completion of the task and the safety of the robot, the plans have to be: safe, complete, and goal-achieving. We call such a plan **robust**.

Safe means it shouldn't lead the robot into a dangerous situation and leave it there without a handy reaction. Complete means all courses of events are accounted for. Goal-achieving means the plan leads to the goal.

Given our strong hypotheses and the complexity, the task of building a robust plan is way beyond our reach. Thus we adopt a **moderated** definition of robustness, based on the three following rules:

1. safety: while applying the plan to the known initial state of the world, given its foreseeable evolution, no situation shall arise where the *safety condition* is unsatisfied;
2. completeness: every foreseeable course of events shall be accounted for, i.e. followed by an action **or a** *replan* **statement**; no *replan* statement shall take care of a situation which could foreseeably become dangerous while replanning (replanning can take an unknown unbounded time). This is a key point in our approach, which helps us limit the exploration, and thus envisage to address more ambitious problems;
3. goal-achieving: the probability of reaching the goal by applying the plan to the known initial state of the world, given its foreseeable evolution, shall be non-null (and superior to a given threshold);

Several previous works give partial answers to theses rules:

(Kabanza, 1992; Godefroid and Kabanza, 1991): F. Kabanza proposes a search algorithm in a graph of states which transitions correspond to actions (operators which the system can control) or events (operators over which the system has no control); his system uses the search itself for generating *liveness rules* (leading towards the goal), *safety rules* (leading away from danger), and *heuristic rules* (which can be used at runtime to deal with an unforeseen situation). Efficient as it may be for a specific class of problems, the search in a graph of states hits the full combinatorial complexity in any closer-to-reality world. Moreover, Kabanza cannot avoid to manage the conflicts between generated rules, which can turn out to be very dangerous in the end.

(Musliner *et al.*, 1992): CIRCA is a complete integration example for a robot control system; by reasoning on a *graph model of RTS/environment interaction*, it generates reactions which are necessary to achieve the goals; from these reactions, it computes a *cyclic schedule* to be executed by the RTS. However, the graph model representation appears to be very complex and difficult to expand to a realistic world domain; this compels CIRCA to remain a low-level control system, which actually is its primary purpose.

(Thiébaux and Hertzberg, 1992): S. Thiébaux and J. Hertzberg propose a nondeterministic action model and a planning method which produces plans as *T-M graphs*; these plans can in turn be translated into finite state automata, which makes them executable and enlarges their applicability scope. However, they

avoid modelling foreseeable changes in the environment, which are indeed partly responsible of the non-determinism of the actions.

3. OUR CLASS OF PROBLEMS

Our domain is mobile robots in changing environments. We define a class of problems characterized as follows:

- the robot system has an exact knowledge of the characteristics of its environment relevant to its task, or it can find them out by executing a specific action;
- these characteristics include the state of the environment and its *reactivity*, i.e. its foreseeable evolution in the context of robot activity;
- moreover, it has an exact model of the actions it can execute; this model includes their non-deterministic consequences and their probability.

Following are two example worlds which belong to this class of problems:

First example: The robot's task is to enter a building in which a chemical incident occurred; it carries a spray which can neutralize one of the contaminating chemicals (but not all that may have been spilled); it can be itself contaminated if in presence of a corrosive chemical for too long, and the contamination is fatal in the short term; it can decontaminate itself by going out of the building and into a cleaning area.

Second example: The robot's task is to carry objects around inside a factory; the production process implies strong unscheduled constraints on its movements; moreover, it is dangerous to stay at some crossings because of heavy carriers.

We will develop the first example in the following sections.

4. REPRESENTATION

Our planning operators are variants of STRIPS operators with pre and postconditions. As in (Thiébaux and Hertzberg, 1992) and (Kushmeric *et al.*, 1993), we allow for alternative outcomes of actions applied in the same situation, and to every set of consequences of the application of an operator, we associate a probability.

We distinguish between *actions* and *events*. Actions are operators over which the robot system has full control, therefore they are akin to the STRIPS planning operators. On the other hand, events are operators over which the robot system has no control; the preconditions of an event are its trigger, and the postconditions are its consequences. The total probability of all outcomes of an action has to be 1, whereas for an event it can be lower (meaning that an event may or may not occur). Figures 2 and 3 present some sample operators, actions and events. Please note that event `robot-dead-contaminated` violates the safety condition `(robot-safe)`, which no action can re-assert.

```
(def-action check-for-room-contamination
  :args (?room)
  :precond (
    (room ?room)
    (robot-at ?room))
  :effects (
    (with-probability
      (0.8 ((:add (room-clean ?room))))
      (0.2
        ((:add (room-contaminated ?room))))))
  :duration 1)
```

Figure 2 *A sample action*

```
(def-event robot-contaminated
  :vars (?room)
  :when (
    (robot-at ?room)
    (room-contaminated ?room))
  :effects (
    (with-probability
      (0.5 ((:add (robot-contaminated))))))
  :delay 10)

(def-event robot-dead-contaminated
  :vars ()
  :when (
    (robot-contaminated))
  :effects (
    (with-probability
      (1.  ((:add (robot-dead))
            (:del (robot-safe))))))
  :delay 25)
```

Figure 3 *Two sample events*

Action operators can be used to describe robot actions, reactions of the environment to robot actions (e.g. while spraying decontaminants, a smoke alarm could be raised automatically), and actions of the lower robot control layers (e.g. when moving along a corridor, the robot might stop because of an obstacle). Event operators account

```
(defstep
   :action (check-for-room-contamination ?room)
   :precond '((room ?room)(robot-at ?room))
   :add '((room-clean ?room))
   :dele nil
   :equals nil)

(defstep
   :action (check-for-room-contamination ?room)
   :precond '((room ?room)(robot-at ?room))
   :add '((room-contaminated ?room))
   :dele nil
   :equals nil)
```

Figure 4 *Generated SNLP operators*

for the evolution of the world state only (e.g. the light being turned off by the time switch, the robot becoming contaminated).

The planning problem is defined as follows: given the initial state of the world (a set of predicates) and the goal, actions and events as described above, the *safety condition* not to be violated, the *replan threshold* and the *goal achievement threshold*, find a **robust plan** (i.e. satisfying the conditions given in section 2). It may be that the goal can be achieved in a situation which is not stable (in which events can occur); in that case, the planner should continue to plan until it has reached a stable state. The concept of *stability* is fundamental in our approach; we assume here that any stable situation is suitable for replanning. In that way, the planner can insure that it will be called again eventually, possibly with a new task to perform.

5. SIMPLE PLANNING ALGORITHM

We propose a first simple planning algorithm, much in the fashion of Warren's *WARPLAN-C* (Warren, 1976). It is based on the deterministic non-linear planner SNLP (Mc Allester and Rosenblitt, 1991; Barrett and Weld, 1992), but could be applied to other deterministic planners

The action operators are first compiled into deterministic operators, one operator being created for every possible outcome of an action. Figure 4 presents the operators for the `check-for-room-contamination` action.

Given the initial state of the world and the compiled operators, a first linear plan is build by the deterministic planner, consisting of a sequence of deterministic operators. Now, using the original actions and events, this plan is expanded into a tree structure akin to the T-M graphs of (Thiébaux and Hertzberg, 1992); this structure is composed of three types of nodes: A (i.e. start of an action), E (i.e. event) and S (i.e. world state), and has the following properties:

- the root is a S-node, as are all leaves,
- A-nodes and E-nodes have only one S-node as successor,
- non-leaf S-nodes have either one A-node or a number of E-nodes as successor.

The procedure for building the A-E-S tree is the following: the first action of the linearized deterministic plan is expanded into one A-node (the start of the action) followed by one S-node (its execution state), itself followed by a number of E-nodes and corresponding S-nodes (one pair for each possible outcome of the action); then the execution S-node is tested for events, as well as all other new S-nodes which do not fit in the deterministic plan; for each S-node, all triggered events are sorted using their *delay* parameter, and tested for exclusiveness (i.e. incompatibility of effects); the list is pruned after the first event with a probability of 1 (if present), and one E-node is generated for every remaining (set of) event(s), along with its successor S-node and its probability of occurrence. The root S-node is built from the initial world state, and the procedure is carried on with every S-node along the deterministic plan, until all actions have been translated into A-nodes and no event is applicable in any leaf S-node. The resulting structure may or may not lead to the goal, depending on the triggered events.

For example, suppose we have two actions A and B, and two events E_1 and E_2. A has two possible outcomes, and B only one, leading to three deterministic operators: A_1, A_2, and B_1. Starting from an initial state S_i, the deterministic planner produces the following linear plan: A_2 then B_1. No events are applicable in S_i, so an A-node is built with the start of action A, then a S-node S_A for its execution, leading to two E-nodes, E_{A1} and E_{A2} and their successor S-nodes S_{A1} and S_{A2} (corresponding to outcomes A_1 and A_2 respectively). Event E_1 is applicable in S_{A1}, so we add an E-node E_{E1} and a S-node S_{E1}. From S_{A2} we expand action B, leading to S_B, E_{B1} and S_{B1}. Event E_2 is applicable in state S_B, so we add two nodes: E_{E2} and S_{E2}, leading to the tree structure presented fig. 5.

Now, all new S-nodes are tested for the *safety condition* (e.g. in our example, `(robot-safe)`). Should this condition be false in state S, then we have to plan for a reaction avoiding S. If S's father is an A-node A, then we prune the plan before

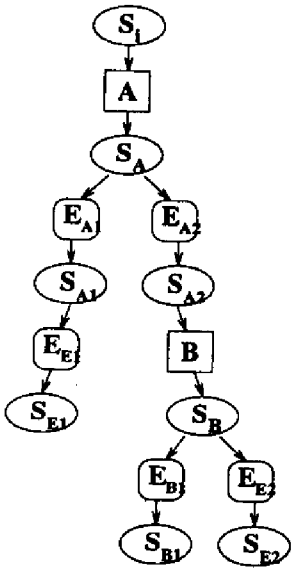


Figure 5 A-E-S tree example

A and look for an other linear plan starting from the new leave. If S's father is an E-node E, again we call the deterministic planner in order to find a new plan, which first action will inhibit E. There are several means to find a new plan: either by using the negation of one of the preconditions of E as new goal, or by inserting any applicable action compatible with the delay of E and planning again from there on. If no plan is found, then the problem is declared unsolved.

If all new S-nodes are safe, then a new S-node with a probability above the *replan threshold* is chosen, among all S-nodes which have no successor A-node. The S-node becomes the initial state, and the deterministic planner is used to build a new plan towards the goal.

In the end, all pending S-nodes which are below the *replan threshold* are flagged with the *replan* statement. The resulting plan is solution if the sum of the probabilities of all goal states is above the *goal achievement threshold*. Figure 6 gives an example of a plan after the first planning step, and at the end of the refinement phase.

Why is our algorithm sound? We can demonstrate that, provided no event loop with a 1 probability exists (which can be easily checked for), the probability keeps decreasing along all branches of the A-E-S tree, and thus will finally go under the replan threshold. Furthermore, the plans produced satisfy the three conditions given in section 2.

We have built a first prototype planner based on SNLP, which managed to produce the plan presented fig. 6. However, it is not complete, as it is not guaranteed to find a plan if one exists. Suppose we add an event stating that the doors close automatically after the robot got through them, and the only way to open them is with a remote-control. After building the first SNLP plan, the system will come up with the event (`door-closed room1`), and is not going to find a solution because it should have taken the remote-control before entering the room. In order to solve this type of problems, we need the ability to backtrack over goals, which is not yet possible with our prototype.

6. PLAN EXECUTION

The *Supervisor* interacts with the environment and with the planner. The environment (which may include one or more lower control layers) is viewed as a set of processes which exchange signals with the supervisor. These processes correspond to the actions of the agent as well as events associated with environment changes independent from robot actions (Alami *et al.*, 1993).

These processes are under the control of the supervisor which has to comply with their specific features. For example, a process representing a robot motion, cannot be cancelled instantaneously by the supervisor: indeed, such a process has an "inertia". The supervisor may request a stop at any moment during execution; however, the process will go through a series of steps before actually finishing.

The simplest way to represent such processes are finite state automata (FSA). More elaborate representations such as temporized processes should be investigated.

The plan itself can also be understood as a finite automaton: A-nodes and S-nodes correspond to the states of the automaton, A-node and E-nodes to transitions.

In the FSA class we use, at any moment:

- the set of allowed external signals correspond to all the actions that can be taken by the supervisor (either part of the plan or decided by the supervisor's own bounded-time decisional abilities);
- similarly, the set of possible internal signals correspond to all action terminations and results and all environment changes that could be perceived by the supervisor.

The activity of the supervisor consists in moni-

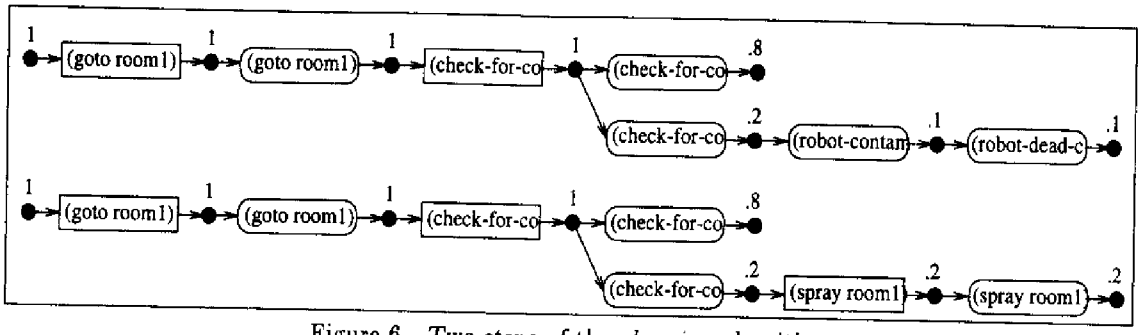


Figure 6 *Two steps of the planning algorithm*

toring the plan execution by performing situation detection and assessment and by taking appropriate decisions in real time, i.e., within time bounds compatible with the rate of the signals produced by the processes and their possible consequences. Such bounded-time decisional abilities are well in the reach of a system like KHEOPS (Ghallab, 1984).

Indeed, every single decision process which can be left to the supervisor, provided it can be done in bounded time, reduces the complexity of the planning problem. For example, suppose the supervisor is able to make the robot leave the building as soon as it becomes contaminated, by following the same path backwards. We could take advantage of this behaviour at planning time by adding a meta-planning rule which takes care of all S-nodes where `(robot-contaminated)` is true and the robot is not too far away from the entrance. That way, we would leave the world model unchanged so that the planner can still plan for these situations if it decides to.

During execution, the robot system will follow one path in the plan automaton, thus validating states which had only a probability of occurrence before. Now with each execution step, we can propagate this validation, dividing the probability of all descendants of the selected state by its own probability, and setting all other states to null. Doing this, it is possible that some states flagged with `(replan)` will raise above the probability threshold and thus become eligible for further planning. In order to save time and profit from our cooperating architecture, the execution supervisor will send right away the most probable state to the planner for further plan refinement.

7. FUTURE WORK

Our first activity will be to implement backtracking strategies in the simple planner and to investigate the possibility of enriching our representation with meta-planning rules. Then we will focus on the supervisor module.

Once the complete system is operational, we will test it extensively on different problems, trying to evaluate the robustness of produced plans and in particular the stability regarding modelling errors (action outcomes' and events' probabilities are especially difficult to estimate).

We hope to give some interesting results of these activities in the final presentation at IAS-4.

8. CONCLUSION

After having justified our approach to integrating reasoning and reaction in the context of autonomous mobile robots, we have described the planner/supervisor interaction and have derived the key properties of the plans to be produced. Then, we have proposed a model for actions and events and a tentative planning algorithm for generating such plans, which we have demonstrated on an example. Finally, we have proposed a plan execution scheme, addressing the issues of planner/supervisor interaction and real-time decision-making.

REFERENCES

Alami, R., Chatila, R. and Espiau, B. (1993). Designing an Intelligent Control Architecture for Autonomous Robots. In: *ICAR '93*. Tokyo, Japan. pp. 435–440.

Barrett, A. and Weld, D. (1992). Partial Order Planning: Evaluating Possible Efficiency Gains. Technical Report 92-05-01. Univ. of Washington, Dept. of Computer Science and Engineering.

Dean, T. and Boddy, M. (1988). An Analysis of Time-Dependent Planning. In: *AAAI-88*. pp. 49–54.

Drummond, M. and Bresina, J. (1990). Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In: *AAAI-90*. Boston. pp. 138–144.

Fikes, R. E., Hart, P. E. and Nilsson, N. J. (1972). Learning and Executing Generalized Plans. *Artificial Intelligence* 3(4), 251–288.

Ghallab, M. (1984). Task execution Monitoring by compiled production rules in an advanced multi-sensor robot. In: *Robotics Research : The Second International Symposium, Kyoto (Japan)*.

Godefroid, P. and Kabanza, F. (1991). An Efficient Reactive Planner for Synthesizing Reactive Plans. In: *AAAI-91*. pp. 640–645.

5.6

Kabanza, F. (1992). Reactive Planning of Immediate Actions. PhD thesis. Université de Liège.

Kushmeric, N., Hanks, S. and Weld, D. (1993). An Algorithm for Probabilistic Planning. Technical Report 93-06-03. Univ. of Washington, Dept. of Computer Science and Engineering. To appear in *Artificial Intelligence*.

Mc Allester, D. and Rosenblitt, D. (1991). Systematic Nonlinear Planning. In: *AAAI-91*. pp. 634–639.

Musliner, D. J., Durfee, E. H. and Shin, K. G. (1992). CIRCA: A Cooperative Intelligent Real-Time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetix*.

Thiébaux, S. and Hertzberg, J. (1992). A Semi-Reactive Planner Based on a Possible Models Action Formalization. In: *1st AIPS*. pp. 228–235.

Warren, D. H. D. (1976). Generating Conditional Plans and Programs. In: *AISB-76 Summer Conference*. Edinburgh.

J.3 Architectures de Contrôle pour Robots Autonomes

6 *Integrated planning and execution control of autonomous robot actions.*

IEEE International Conference on Robotics and Automation, Nice, mai 1992.

Auteurs: R. Chatila, R. Alami, B. Degallaix, H. Laruelle.

7 *Designing an intelligent control architecture for autonomous robots.*

International Conference on Advanced Robotics *ICAR'93*, Tokyo (Japon), Novembre 1993

Auteurs: R. Alami, R. Chatila, B. Espiau

8 *Planet exploration by robots: from mission planning to autonomous navigation.*

International Conference on Advanced Robotics *ICAR'93*, Tokyo (Japon), Novembre 1993

Auteurs: R. Chatila, R. Alami, S. Lacroix, J. Perret, C. Proust.

9 *How To Tele-Program a Remote Intelligent Robot.*

IEEE *IROS'94*, Munich (RFA), Septembre 1994.

Auteurs: J. Perret, C. Proust, R. Alami, R. Chatila

Integrated Planning and Execution Control of Autonomous Robot Actions

Raja Chatila Rachid Alami Bernard Degallaix* Hervé Laruelle†

LAAS-CNRS
7, Ave. du Colonel Roche, 31077 Toulouse cedex - France

Abstract

This paper describes an implemented integrated system allowing a mobile robot to plan its actions, taking into account temporal constraints, and to control their execution in real time. The general architecture has three levels and the approach is related to hierarchical planning: the plan produced by the temporal planner is further refined at the control level that in turn supervises its execution by a functional level. The framework of the french Mars Rover project VAP is used as an illustration of the various aspects discussed in the paper.

1 Introduction

Intervention robots are machines that have to perform non-repetitive and time-constrained tasks in ill-known environments which are often remote or of difficult or dangerous access, with specific constraints on communication (delays, bandwidth). In this context, classical teleoperation as well as telerobotics-like [21] approaches with a human operator in the control loop are not adequate [9].

Some authors argue that planning is not useful, and that reactivity is the necessary and sufficient ingredient of robot intelligence [4]. Furthermore, *collective intelligence* would emerge as a result of the simple interactions of reactive robots which have complete autonomy but limited behavior [5, 7]. Being not able of *predicting* their actions and their outcome, wich is one aspect of planning, such robots are more data than goal driven, and if ever able to accomplish a given task, they would certainly lack efficiency.

The challenge is to design an autonomous robot endowed with the capacities of planning its own actions in order to accomplish specified tasks, and having a reactive behavior with respect to is environment. Such a robot would be both goal and data driven. The work presented in this paper is a contribution to this objective.

Plans are produced by a temporal planner at the higher level. They are then refined into actions according to specified execution modalities and to the actual execution context [2, 18]. The robot control structure includes the systems necessary for task interpretation and autonomous execution. The system is aimed to be generic and applicable to several domain instances of intervention robots: planetary rovers [6], underwater vehicles [12], disaster reaction [18], etc.

The paper is organized as follows: section 2 presents the functional architecture of the overall system; section 3 focuses on planning and section 4 plan execution supervision. Finally, section 5 presents the task refinement and execution levels.

In order to illustrate the approach we will consider a Mars Rover as an example of intervention robots troughout this paper. This case study is motivated by the french national project VAP[1] [6].

2 System Architecture

The global system architecture is organized into three levels (figure 1). A higher temporal planning level - with its own supervision. The planning time is unbounded.

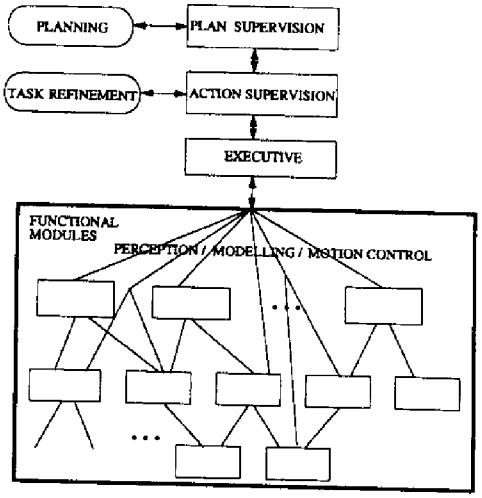


Figure 1: Global architecture

The second "refinement" level receives tasks that it transforms into sequences of actions using its own interpretation and planning capacities, and supervises

*Supported by IFREMER
†Supported by a "CIFRE" with MATRA

[1]VAP: Autonomous Planetary Vehicle, a project of the french national space agency CNES.

the execution of these actions while being reactive to asynchronous events and environment conditions. This level comprises a **task supervisor** that is in charge of receiving the plan from the higher level and interacting with the second component, the **task refinement planner** which refines tasks into actions, taking into account the execution modalities specified with the plan, and the actual situation of the robot. The response time of this level is bounded as the refinement planning is in fact very much context-driven selection of actions in a precompiled structure.

The functional "execution" level is composed of a set of modules embedding the functions for sensing and acting, including various specific processings necessary for perception and motion control [2, 20, 10]. This level is managed and controlled by a central **Executive** in order to execute the actions requested by the task supervisor. The response time of these modules that implement polynomial time algorithms is bounded.

A module embeds primitive robot functions which share common data or resources [1, 8]. An internal control process called the "module manager" is responsible for receiving requests to perform these functions from the robot controller, and for otherwise managing the module. Each function being well defined, its activation or termination must respect certain conditions that the module manager verifies.

Modules interact by message passing or by reading data exported by other modules, and by putting their own processing results into exported data structures (EDS). At a given time, a module can be executing several functions. All of the functions of a given module are pre-defined at the system design stage.

In the context of a Mars rover, the planner could be on Earth, while the plan supervisor in charge of controlling its execution and the two other levels are on-board the robot.

3 Planning

3.1 Temporal Planning

The planning level is based on general action planning techniques, including temporal reasoning, since it is necessary to take into account time constraints in robots that have to act in the real world.

The explicit representation of time allows for an operator representation that is richer than STRIPS operators or their usual extensions as summarized in [16]. It is possible, for instance, to specify information concerning the duration of operators, the relative time when the postconditions of an operator become true, the conditions which must remain true during action execution, joint effects of operators with other operators executed in parallel, and the like. This has been partly addressed in temporal planners like DEVISER [22] and in event based planners like GEMPLAN [17] or, more recently, CHICA [19]. GEMPLAN also makes a difference between temporal and causal orderings of operators. FORBIN [11], as a time map based planner, involves most of these issues.

We have developed a temporal planning system called *IxTeT* (Indexed Time Table) [13, 15] which can reason on symbolic and numeric temporal relations between time instants. It produces a set of partially ordered tasks with temporal constraints.

IxTeT's representation is based on reified logic. It relies on a 2-dimensional array (called Indexed Time Table) with rows corresponding to logical assertions and columns to time points (instants). Cells in the table are temporal qualifications of the assertions. Temporal constraints between instants are represented by a time lattice. A temporal relation manager maintains the lattice and propagates temporal constraints [14].

The descriptions of the world, the goals and the decomposition operators are given using symbolic and/or numeric temporal relations between time points (instants) or a set of temporal relations between intervals (*start, meet, finish, during, overlap, before, same* and the inverse relations) [3] which can be transformed into relations between instants.

Ixtet involves two levels of description for decomposition operators: the *Task* and the *Procedure*.

Tasks in IxTeT correspond to the lowest level of description. The 'effects' of a task are assertions which change as a direct result of performing that task. Hence, a task is described through its 'effects', temporally linked to the interval of execution of the task. Minimal and maximal duration of a task may also be specified.

For example, in the task description given in figure 2, *(Data-Available ?data ?site ?target)* becomes true at the end of the task while *(Sending-Data ?target)* becomes and remains true during task execution.

```
(ixtet:Task TASK-SEND-DATA
        :args (?data ?site ?target)
        :effects ( (:meets Data-Available ?data ?site ?target)
                   (:equal Sending-Data ?target))
        :duration-min (length ?data)
        :duration-max (length ?data))
```

Figure 2: A task description

Procedures are the decomposition operators. They are defined in terms of a context (a set of conditions needed in order to apply the procedure), a set of temporally constrained tasks required to achieve a particular goal, together with additional effects which are due to the combined execution of the tasks involved by the procedure.

In the example given in figure 3, in order to apply the procedure *SEND-PANORAMA*, the robot will have to wait until *(Sunlight)* becomes true before executing *TASK-GET-PANORAMA(?site)*, and to wait for a visibility window with the target (Earth or Orbiter) and the rover, with a sufficient duration for performing *TASK-SEND-DATA(panorama, ?site, ?target)*.

Planning in IxTeT proceeds from an initial context called 'initial situation' containing an initial state, ex-

```
(ixtet:Procedure SEND-PANORAMA
    :args (?site ?target)
    :goal (Data-Available panorama ?site ?target)
    :context (   (:during s1 (:on (Sunlight)))
                 (:during s1 (:on (VAP-Site ?site)))
                 (:during s2 (:on (Visibility ?target))))
    :steps       (s1 (TASK-GET-PANORAMA ?site)
                  s2 (TASK-SEND-DATA panorama ?site ?target))
    :such-that ((:before s1 s2))
    :effects ())
```

Figure 3: A Procedure description

pected events and goals which are linked (directly or indirectly) to an initial time-point by numerical or symbolic temporal relations. The occurence dates may be specified more or less precisely (ranging from an exact absolute or relative occurence date, to a numerical interval, to a simple symbolic ordering).

The planner performs goal decomposition by heuristically selecting procedures and inserting them in the time lattice together with constraints addition and propagation.

3.2 Example

Here we describe an example based on the VAP mars Rover mission scenario. If on-board computing capacities are limited, planning may take place on Earth, and the plan be sent to the rover for refinement and execution.

The mission of VAP consists in sending to the orbiter a panoramic view of a zone called *site-2* and in collecting geological samples from *site-3*.

The initial time-point is noted 1. At this time the rover is localized in a geographic zone called *site-1*. Besides, there are three expected events that are known to occur in the future: *(Sunlight)* will become true at instant 2, a visibility window will start with the orbiter at instant 18 and with Earth at instant 17. This initial situation is represented by the time lattice in figure 4.

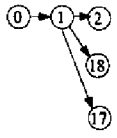


Figure 4: The initial situation

The procedure *Vap-Navigate(?site1, ?site2)* can only be applied during day-time (sunlight). Besides, *Collect-samples(?site)* can be applied only after the panoramic view of *?site* was transmitted.

Figure 5 illustrates the plan as produced by IxTeT. The upper part of the table shows the validity in time of the predicates describing the world state and the effects of the robot's actions (values before instant 1 are not relevant). The lower part shows the the tasks

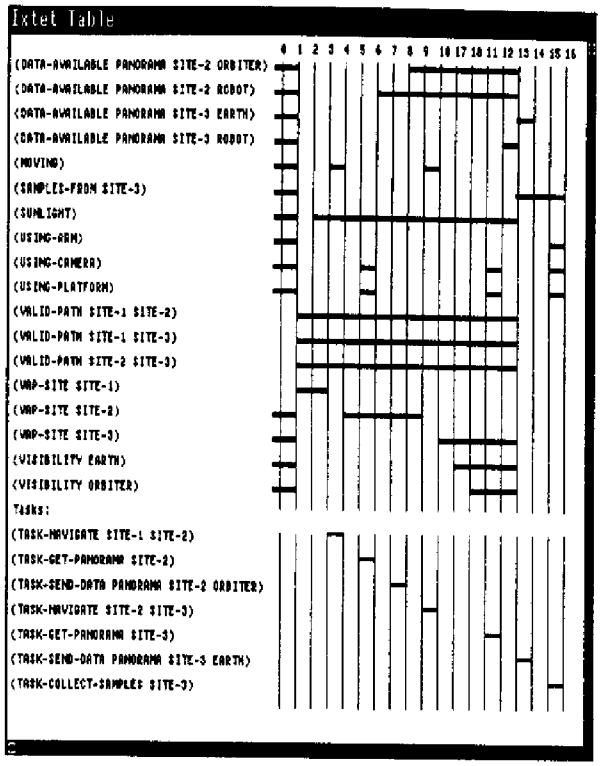


Figure 5: The indexed time table. The upper parts shows predicate validity in time and the lower part the tasks in the plan.

composing the plan and their beginning and end instants. Note that the identifiers of the time instants are purely arbitrary and are not ordered in figure 5. The temporal relations between the instants (partial order) are given in figure 6.

We see that in order to achieve the mission goal, the rover must navigate from *site-1* to *site-2*, acquire a panoramic view, and then send it to the orbiter, navigate from *site-2* to *site-3*, acquire a new panoramic view which is sent to Earth, and finally collect samples. Note also that the rover will wait until *Sunlight* (instant 2) becomes true before moving and that it will wait for visibility with the orbiter or Earth in order to communicate with them. Furthermore, the tasks *Send-data(Panorama, site-2, orbiter)* and *Navigate(site-2, site-3)* may be executed in parallel (instants 7 and 9).

4 Plan Supervision

4.1 Introduction

A Plan, as produced by IxTeT, is a set of partially ordered tasks, together with temporal constraints such as minimal and maximal expected durations, and synchronisation with expected external events or absolute dates.

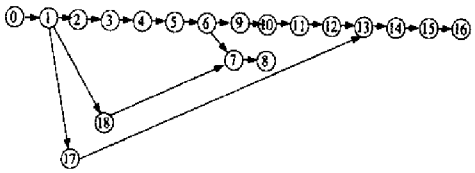


Figure 6: The lattice of time instants.

The plan supervisor is in charge of monitoring the execution of the tasks involved in the plan. It has therefore to verify that the tasks produce indeed the expected effects, and must react in case of discrepancy. Execution will be globally managed by the robot supervisor, that schedules the tasks, sending them to the action supervisor that controls their execution, possibly after refinement.

4.2 Plan Execution Monitoring

Instants in the time lattice correspond to different event types: beginning or end of task execution, intermediate events produced by tasks during their execution, expected external events (which occur independently from robot actions). Besides, numerical bounds for dates and durations may be attached to some timepoints or intervals.

The plan execution control process interacts with a clock by requiring messages to be sent at given absolute dates, and with the robot action supervisor by requiring task execution or cancellation (fig 7).

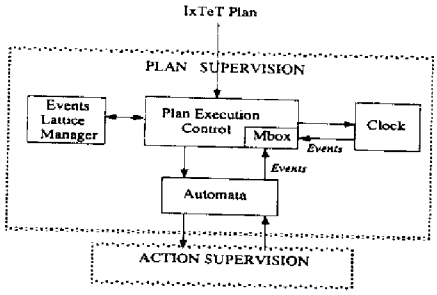


Figure 7: The Plan supervisor

Messages received from the clock authorize the system to state the occurence of dated expected events and to monitor the tasks minimal and maximal expected durations.

Messages received from the robot action supervisor are 'filtered' by the associated automata (see below §4.3) and transformed into events which correspond either to instants planned in the time lattice (in case of nominal execution) or to unexpected instants otherwise.

The plan supervisor starts from a given instant in the time lattice and 'executes' the time lattice by performing the following actions:

- At any moment, it considers only the time instants whose predecessors have been processed and whose planned occurence date is compatible with the current time.
- It requires the execution of a new task when it reaches the instant which corresponds to its beginning.
- While processing incoming events, the supervisor verifies that they correspond to planned instants and that they satisfy the planned ordering and numerical time constraints. If it is the case, the absolute date corresponding to the occurence of the events is considered, inducing a progressive 'linearisation' of the time lattice (see figure 8).
- In the current implementation, the only 'standard' reaction performed by the plan supervisor in case of non-nominal situations is to stop the current tasks, and update the world state according to events incoming from task execution, until all tasks are stopped. This is performed by inserting new time instants in the time lattice corresponding to the transitions as defined by the automata.

Figure 8 shows the plan at two different stages of interpretation. The first time lattice represents a situation wherein the rover is navigating from *site-2* to *site-3* (indeed, instant 9 is in the past while instant 10 is still in the future). It has previously acquired a panoramic view of *site-2* (between instants 5 and 6), however it has to wait for instant 18 (visibility with the orbiter) in order to send the data (between instants 7 and 8).

In the second lattice, instant 18 has occured; the rover is performing two tasks in parallel (navigating and sending data to the orbiter).

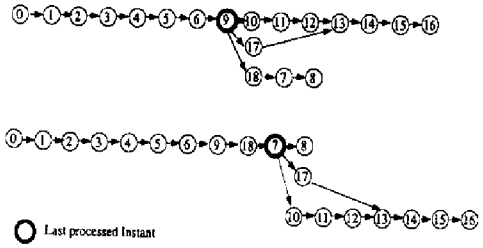


Figure 8: Plan execution

4.3 Using Automata To Monitor Task Execution

In order to allow the plan supervisor to monitor task execution and to act on them while they are executed, each task is modelled by a finite state automaton (FSA).

The FSA associated to a task not only models the nominal task execution (as defined in the task description manipulated by the planner) but also non-nominal situations (exceptions), as well as the different actions which should be taken then by the plan supervisor.

In the finite state automata we use, state transitions can be caused by 'internal events' (i.e. events generated by the task execution process and transmitted to the plan interpreter) or by 'external events' (i.e. events due to an action of the plan supervisor on the task while it is executed).

Typical internal events are: RUN (beginning of execution), RETURN (nominal end of execution), STOP_ON_FAILURE (end due to a failure), STOPPED (end of execution caused by an external STOP request). A Typical external event is STOP (corresponding to a stop request issued by the plan interpreter). Other external events are related to the task and force a state transition in the execution.

For each task, an automaton class is provided. An example is given in figure 9. Note that each automaton state change produces transitions in the world state as it is maintained by IxTeT. The example in figure 9 shows a decription of automaton associated to the task *TASK-NAVIGATE*. Whenever a task execution is started by the plan supervisor, an automaton of the associated class is instanciated and initialized. It will then represent the execution of the task as it is viewed by the plan supervisor.

```
(ixtet:Task TASK-NAVIGATE
    :args (?site1 ?site2)
    :effects (: is-met VAP-Site ?site1
             (:meets VAP-Site ?site2)
             (:equal Moving))
    :duration-min (min-length ?site1 ?site2)
    :duration-max (max-length ?site1 ?site2))

(DEFINE-AUTOMATE-CLASS TASK-NAVIGATE (?site1 ?site2)
 (:STATE begin
  :INTERNAL-EVENTS ((:run
                     :NEXT-STATE .executing
                     :TRANSITIONS ((:ON MOVING)
                                   (:OFF VAP-SITE ?site1))))
  :EXTERNAL-EVENTS ((:stop
                     :NEXT-STATE .stopping
                     :TRANSITIONS ())))
 (:STATE executing
  :INTERNAL-EVENTS ((:return
                     :NEXT-STATE :end
                     :TRANSITIONS (: :OFF MOVING)
                                  (: ON VAP-SITE ?site2))
                    (:execution_failure
                     :NEXT-STATE :failure
                     :TRANSITIONS ((:ON NAVIGATION-FAILURE))))
  :EXTERNAL-EVENTS ((:stop
                     :NEXT-STATE .stopping
                     :TRANSITIONS ())))
 (:STATE failure
  :INTERNAL-EVENTS ((:stop_on_failure
                     :NEXT-STATE :end
                     :TRANSITIONS ((:OFF MOVING)
                                   (:ON VAP-SITE CURRENT)
                                   (:ON VALID-PATH CURRENT ?site1)
                                   (:OFF VALID-PATH ?site1 ?site2))))
  :EXTERNAL-EVENTS ((:stop
                     :NEXT-STATE :stopping
                     :TRANSITIONS ())))
 ... )
```

Figure 9: Automaton for TASK-NAVIGATE

5 Refinement and Execution Control

5.1 Task Refinement

Task Refinement transforms a task into specific actions that are adapted to the actual context. As an example, a motion task may be executed in different ways:

- a displacement using only dead-reckoning systems to guide the movement.
- a closed-loop motion using a perceptual feature of the environment (landmark tracking, edge following of a large object, rim following, etc.).

These two modes correspond to the execution of different *scripts* (see [18] for details), associated with the task "move". Script selection is based on testing conditions using the acquired data. Scripts have variables as arguments, that are instanciated at execution time. The execution of a script is similar to the execution of a *program*.

5.2 Task Execution

Action execution are represented by *activities*. The execution of a script corresponds to a global activity. An activity is thus equivalent to the execution of a program, and is analogous to the notion of process in an computer operating system. A simple activity is the execution of a function by a module. An activity may cause the emission of requests to other modules, starting *children activities*. The module in the parent activity is then a client of the module in the child activity. An activity may be the parent of many children, but may be the child of only one other activity. A set of rules and mechanisms were developed to create and manage activities. Two basic mechanisms are activity creation and message transmission between two activities.

At a given moment, the set of activities represent the functions being executed in the robot system. The activity structure is a tree with a parent-child relationship. The tree evolves while the robot is executing. An activity communicates with its (single) parent activity via "up-signals" and with its (eventual) child activities via "down-signals". The activity hierarchy is not predefined and depends on the current task.

Regardless of the specific processing it performs, an activity must be able to react to signals sent by its parent or its children. In particular, it must be able to react to asynchronous signals within a pre-defined bounded time delay.

An activity is also represented by a finite automaton. Its state changes are dependent on external or internal signals. Control flow between a mother and child activities is implemented as typed messages that cause a state change. Specifc mechanisms permit the propagation of a state change along the activity tree.

Important features of the notion of activity are:

- Activities provide for the management of a hierarchy of actions without imposing a fixed number of layers.
- All activities, regardless of level, may be treated in the same way.
- No assumptions are made about the nature of the inter-connections between activities: the activity

hierarchy is *not* pre-defined. We might require that a "high level" activity starts and manages a "low level" activity at any level. This knowledge must however exist in the modules.

- All mechanisms for managing activities (starting, terminating, etc.) and the communication between them do not depend upon any programming language constructs; in this sense, they resemble part of an operating system.
- The concept of activity permits reactivity at all levels: at any moment, each activity in the tree structure is able to respond to asynchronous events.

Figure 10 represents an example of activity tree at one stage of the execution of a *go-to(location)* task. The "root" box represents the mother activity of this task, within the executive. The "monitor-11" box is the mother activity of a main child-activity (monitor-12) and of an associated monitoring activity (timer-200: a time-out for this task). When the corresponding event occurs, the main activity is stopped. "Monitor-12" is a monitoring activity also: the traveled distance should not exceed a given maximum (here 60 meters). Sequence in the "sequence-6" box means that the following children are to be executed in sequence (this defines a sub-activity block). "go-to-xy" and "exec-traj" are the two sequential activities. The first is the "go to" coordinate location, and the second is the trajectory execution phase within it. Other possible phases are "build-model" (environment modelling) and "build-traj" (trajectory planning). The last box "move-5" is the primitive action move being executed.

Figure 11 shows the execution of the *go-to(location)* task in a laboratory experiment, where the robot discovers its environment and builds a model of it while it navigates to reach the assigned location.

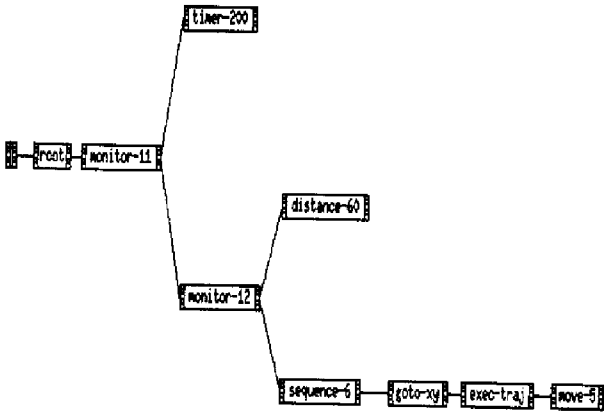


Figure 10: Activity tree during execution of a GOTO task

6 Conclusion

We presented in this paper a global approach to task planning and execution for intervention robots. Such robots are characterized by the fact that they have to be able to autonomously execute their actions in a partially and poorly known environment in order to accomplish missions and tasks specified and programmed by a human user. The robot have to interpret the tasks according to the context and its evolution, and to achieve autonomous execution. The main interest of the plan execution supervision as it is proposed in this paper is to maintain and update a complete history of the plan execution not only in nominal cases but also when a failure occurs. This is performed using automata which model tasks execution and their interaction with the plan supervisor. However other representations may be used. Indeed, we are working on an extension using on a compiled rule-based system which should allow not only to model task execution but also combined effects due to the simultaneous execution of several tasks as well as domain dependent knowledge allowing to infer new transitions starting from a set of observed events. While it is still necessary to further deepen some aspects, the experimental results show that this approach is sound and applicable.

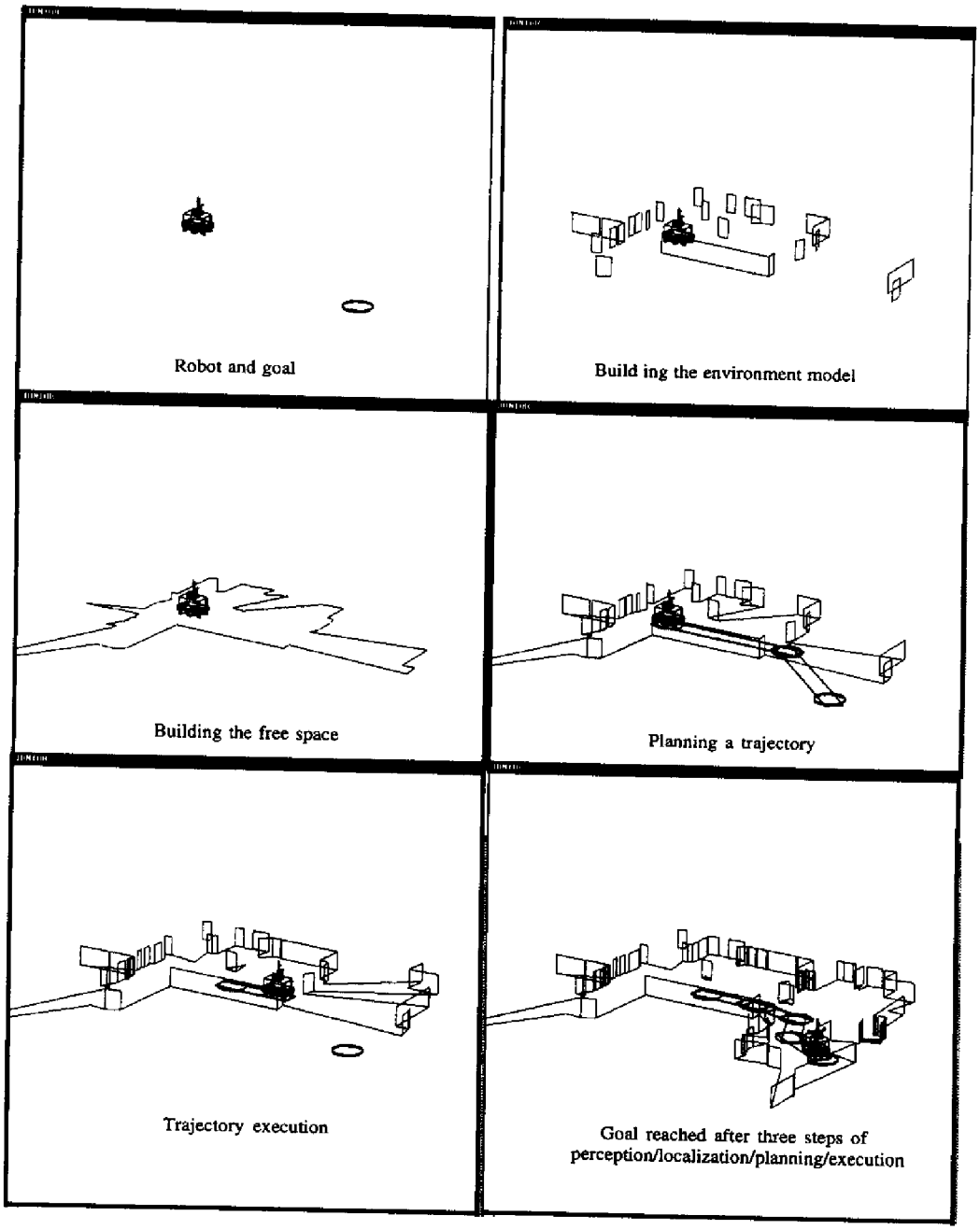


Figure 11: Experimental Execution of a *goto(location)* task.

References

[1] R. Alami, R. Chatila, M. Devy, and M. Vaisset. System architecture and processes for robot control. Technical report, Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), Toulouse (France), June 1990.

[2] R. Alami, R. Chatila, and P. Freedman. Task level programming for intervention robots. In *IARP 1st Workshop on Mobile Robots for Subsea Environments, Monterey, California (USA)*, pages 119–136, October 1990.

[3] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[4] R. A. Brooks. A robust layered control system for a mobile robot. In *IEEE International Journal of Robotics and Automation*, April 1986.

[5] C. M. Angle and R. A. Brooks. Small planetary rovers. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 383–388, July 1990.

[6] L. Boissier and G. Giralt. Autonomous Planetary Rover (V.A.P.). In *???*, ??? 1991.

[7] R. A. Brooks, P. Maes, and G. Moore. Lunar base construction robots. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 389–392, July 1990.

[8] R. Ferraz De Camargo, R. Chatila, and R. Alami. A distributed evolvable control architecture for mobile robots. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 1646–1649, 1991.

[9] R. Chatila, R. Alami, and G. Giralt. Task-level programmable intervention autonomous robots. In P. A. MacConaill, P. Drews, and K.-H. Robrock, editors, *Mechatronics and Robotics I*, pages 77–87. IOS Press, 1991.

[10] R. Chatila and R. Ferraz De Camargo. Open architecture design and inter-task/intermodule communication for an autonomous mobile robot. In *IEEE International Workshop On Intelligent Robots and Systems, Tsuchiura, Japan*, July 1990.

[11] T. Dean, R.J. Firby, and D. Miller. Hierarchical Planning Involving Deadlines, Travel Time, and Resources. *Comput. Intell.*, 1988.

[12] L. Floury and R. Gable. The Wireline Reentry in Deep Ocean DSDP/ODP Holes. Technical Report DITI/ICA-91/172-LF/DB, IFREMER - Centre de Brest, July 1991.

[13] M. Ghallab, R. Alami, and R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles, editor, *Robotics Research: The Fourth International Symposium*. MIT Press, Mass., 1988.

[14] M. Ghallab and A. Mounir Alaoui. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *11th International Joint Conference on Artificial Intelligence (IJCAI), Detroit, Michigan (USA)*, pages 1297–1303, 1989.

[15] M. Ghallab and A. Mounir Alaoui. Relations temporelles symboliques: représentations et algorithmes. *Revue d'Intelligence Artificielle*, 3(3), February 1990.

[16] J. Hendler, A. Tate, and M. Drummond. AI Planning: Systems and Techniques. *Artificial Intelligence Magazine*, 11(2):61–77, Summer 1990.

[17] A.L. Lansky. A Representation of Parallel Activity Based on Events, Structure, and Causality. In *Reasoning about Actions and Plans. Proc. of the 1986 Workshop, Timberline*, pages 123–159, 1987.

[18] R. Laurette, A. de Saint Vincent, R. Alami, R. Chatila, and V. Pérébaskine. Supervision and Control of the AMR Intervention Robot. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 1057–1062, June 1991.

[19] L. Missiaen. *Localized Abductive Planning with the Event Calculus*. PhD thesis, Dept. of Comput. Science, KU Leuven (Belgium), 1991.

[20] F. R. Noreils and R. G. Chatila. Control of mobile robot actions. In *IEEE International Conference on Robotics and Automation, Scottsdale, (USA)*, pages 701–712, 1989.

[21] T. Sheridan. Telerobotics. In *IEEE Int. Conf. on Robotics and Automation (Workshop on Integration of AI and Robotic Systems*, 1989.

[22] S.A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3), May 1983.

Designing an Intelligent Control Architecture for Autonomous Robots

Rachid Alami and Raja Chatila
LAAS-CNRS
7, Ave. du Colonel Roche, 31077 Toulouse cedex - France

Bernard Espiau
INRIA
2004, Avenue des Lucioles, 06902 Sophia-Antipolis - France

Abstract: Designing a control architecture for autonomous robots able to reason and to act in their environment to accomplish tasks, and to adapt in real time to the actual execution context, raises major questions in the field of control theory, knowledge-based and reasoning systems, as well as software architecture. We propose in this paper an approach to build such systems, based on the experiences and contributions of two laboratories, INRIA and LAAS-CNRS.

1 Introduction

It is known that the necessity of designing truly autonomous robots comes from highly demanding applications (undersea intervention, planetary exploration, working in nuclear plants, etc.). Since, in such application fields, any repairing intervention is excluded, the need for a total *technological* autonomy is obvious: energy, computing facilities, sensors... Furthermore, the dialogue with an end-user is reduced to mission specification and, when possible, to very few high-level communications (for example teleprogramming). This means that an almost total *operational* autonomy is also required, especially in the domains of perception, decision and control.

This search for operational autonomy raises major questions in the fields of control theory, knowledge-based and reasoning systems, as well as architecture. However, the specificity of the application domain generates particular constraints which may sometimes be felt as antagonistic according to the concerned scientific domain:

1) The need for *adaptivity* in time and space. This leads to the necessity of on-line planning:

- because of the incompleteness of the knowledge, or of existing uncertainties on measurements. This is for example the case when the environment is gradually discovered by the robot;
- because of technical limitations. In most cases, the on-board storage capabilities make impossible to keep in memory all possible execution courses.

2) The need for *reliability*. Since no intervention is possible in general, it is necessary to be sure, as far as possible, that the robot will work satisfactorily *before its launching*. This implies:

- the necessity of ensuring that every step down to the real-time implementation of the tasks handled by a planner is correct, takes the expected time and that possible conflicts may be avoided;
- to be able to verify the largest possible set of assertions about the mission: nominal behavior, emergency procedures, etc.

Up to now, and in most of the architectures for autonomous robots, these requirements were not considered as compatible: item 2 needs an exhaustive knowledge of all the handled entities and of the way they are organized. Determinism and synchrony assumption are often used. Analysis of continuous-time aspects is also required. This may be clearly not compatible with issues of item 1. Conversely, item 1 requirements make difficult non-trivial verifications and are inappropriate to take into account automatic control aspects.

The aim of this paper is to present and illustrate some ideas which would constitute the generic basis of an architecture allowing to take into account as far as possible the previous requirements. The challenge may here be understood as the feasibility of designing an autonomous robot endowed with capacities of planning its own actions in order to accomplish specified tasks while having a so-called reactive or reflex behavior with respect to its environment. For that purpose, we emphasize design aspects at two levels in the robot architecture:

- the *functional* level: it concerns the design, validation and implementation of control loops associated with a local behavior, called robot-tasks;
- the *decisional level*: it relies on a layered plan-based architecture and provides with a suitable framework for the interaction between *deliberation* and *action*, as defined later.

The interest of this structure is that on-line planning will be allowed at the last level, without excluding reactivity or sensor-based reflex actions at the first one. Furthermore, verification aspects will be made possible almost from implementation up to high

level mission specification when models based on state-transitions systems are used.

The paper is organized as follows: in the next section, we present the concepts underlying our approach to the design of a intelligent robot. Then, in section 3, we detail the functional level, and in section 4 the decisional level. An example of the proposed architecture is finally given in section 5.

2 Basic Principles

The proposed approach is based on a few simple ideas, an overview of which we give in this section.

2.1 At the Functional Level

A key question is "What is an action?" In other words, what are the characteristics of the smallest entity of this type handled by the decisional level? A partial response lies in the concept of *reactivity*, widely used, sometimes wrongly. The theory of discrete-event systems provide us with a rather precise definition: a synchronous reactive system produces a set of output events deterministically and instantaneously upon the occurrence of a set of input events. Such an emission of signals may thus be considered as an *internal* action, which will perhaps lead to a *robot* action, i.e. a dedicated motion. This last may therefore be started by an event occurrence, but it still remains to describe what this motion will be. A second point is that, clearly, other types of actions may also be triggered on event reception at a higher level: sensor activation, planning requirement, asking for operator intervention, etc. We thus already conclude: 1- that a too general concept of action is not the adequate one at the functional level; 2- that the specification in terms of reactivity only is necessary but not sufficient.

Let us now come to the problem of specifying and controlling a robot action. This may be stated as a control problem which may be efficiently solved in real-time using adequate feedback control loops. Since it is a powerful and mathematically rigorous tool, we believe that control theory should be used as far as possible. In particular, sensor-based control loops are a nice way of performing motions in continuous interaction with the environment, called *reflex actions*. Furthermore, it will be seen that implementation issues are easily handled within this framework. This is why we define the key entity at the functional level as a kind of *event-driven reflex action*, called *robot-task*.

2.2 At the Decisional Level

The investigation on the interaction between deliberation and action is certainly a key aspect in the development of intelligent agents and particularly autonomous robots. Deliberation here is both a goal-oriented planning process wherein the robot anticipates its actions and the evolution of the world, and also a time-bounded context dependent decision for a timely response to events.

While there are high emergency situations where a first and immediate (reflex) reaction should be performed, such situations often require last resort safety actions. They can often be avoided if the agent is able to detect events which allow it to predict such situations in advance. Note that this is first a requirement for sensors and sensor data interpretation. Being informed in advance and consequently having more time to deliberate, the agent should be able to produce a better decision. Note also that such a capacity is generally ignored in "purely" reactive approaches where the robot makes uses only of "short" range sensors giving it data on its immediate environment.

Acting is permanent, and planning should be done in parallel: an intelligent agent should not neglect any opportunity to anticipate (i.e, to plan). However, since planning requires an amount of time usually longer than the dynamics imposed by the occurrence of an event, the paradigm that we shall develop consists in controlling the functional level by a deliberative system that has a bounded reaction time for a first response. This level is composed of a *planner* which produces the sequence of actions necessary to achieve a given task or to reach a given goal, and a *supervisor* which actually interacts with the functional level, controls the execution of the plan and reacts to incoming events.

The decisional level may not be unique: a planner usually requires abstract models, and its representations of actions do not embed the *actual* interactions with the environment. For example, a planner would use a model in which situations are described in terms of predicates and general topology (e.g. "connects (D1, R1, R2)") without taking into account the geometry of the environment. Furthermore, there may be several ways to execute a given action (as defined at the high-level planning system) depending on the actual execution situation. Hence, there may be several decisional layers the lower ones manipulating representations of actions which are more procedural and closer to the execution conditions.

Let us now present functional and decisional levels more deeply.

3 The Functional Level

As previously evoked, this level mainly relies on the concept of *robot task*. This keystone concept is the minimal granule to be handled by the *decisional* level, while it is the object of maximum complexity to be concerned by the *control* aspects. It characterizes in a structured way a closed loop control scheme, the temporal features related to its implementation and the management of associated events. Fully described in [19], it is defined in a formal way as follows:

A Robot-Task is the entire parametrized specification:
- of *an elementary servo-control task*, i.e. the activation of a control scheme structurally invariant along the task duration;
- and of *a logical behavior* associated with a set of signals liable to occur previously to and during the task execution.

Let us give some details on these two aspects.

3.1 Design and Implementation of a Servoing Task

A first step consists in specifying the continuous-time control law associated with a dedicated robot-task. In the case of a two-wheeled mobile robot, it is for example the expression of the control which, given desired and actual configuration at time t, computes the wheel velocities or the driving torques to be applied. Usually, such a scheme may be split in several functional modules (position estimation, trajectory generation, feedback control...) which exchange data.

Now this description should take into account implementation issues: discretization, variable quantization, delays, computation times, periods, communication and synchronization between the involved processes. This is done by defining the basic entity called *Module-Task*, which is a real-time task used to implement an elementary functional module of the control law.

Since the *Module-Tasks* may, possibly, be distributed over a multiprocessor target architecture, they communicate using message passing and typed ports. A set of 8 communication and synchronization mechanisms is provided (see [19]). Dedicated simulation tools allow to finely validate this design step.

3.2 The Event-based Behavior

In a way, a Robot Task is atomic for the decisional level. However it follows an internal sequencing which has not to be seen in normal (failure-free) circumstances. Nevertheless the Robot Task has also to exchange information with the supervisor, described later, which synchronizes and/or conditions their activation. In the present approach, these two aspects are considered in a single way. Thus the Robot Tasks can be considered as reactive systems and can be programmed using the synchrony assumption ([5]): signals are emitted from and to a finite state automaton which specifies the Robot-Task behavior. This automaton, called RTA, is encoded using the synchronous language ESTEREL ([5]).

The signals are emitted by specific module tasks, called "observers". They are strongly typed:

- *the pre-conditions.* Their occurrence is required for starting the servoing task. They may be pure synchronization signals or signals related to the environment, usually obtained through a sensor
- *the exceptions.* They are exclusively emitted by observers in case of failure detection.
- *the post-conditions.* They are either logical synchronization signals emitted by the RTA itself in case of correct termination, or signals related to the environment.

The treatments associated with pre- and post-conditions are quite simple and not described here. The exception processing is more specific:

- *type 1 processing*: the reaction to the received exception signal is limited to the modification of the value of at least one parameter in the module-tasks (gain tuning, for example).

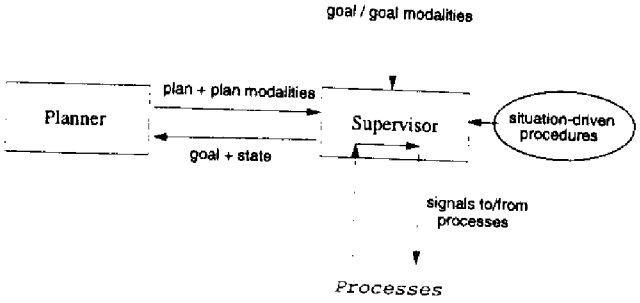


Figure 2: Paradigm for the Integration of Planning and Execution

- *type 2 processing*: the exception requires the activation of a new Robot-Task. The current one is therefore killed. When the ending is correct, the nominal post-conditions are fulfilled. Otherwise, a specific signal is emitted towards the supervisor, which *knows* the recovering process to activate (see later).
- *type 3 processing*: the exception is considered as fatal. Then, everything is stopped.

Design of a Robot Task: An object-oriented approach is used for design and modelling of robot-tasks. Based on this approach, a dedicated human-machine interface has been realized. It allows to easily instantiate all the objects required in a given robot task and to specify the values of temporal attributes. Graphic facilities are also provided. Figure 1 gives an example of a robot task which consists in making park a mobile robot. To conclude on this aspect, let us underline that the ESTEREL synchronous code is automatically generated from the object specification. All existing verification tools may then be used on this code without any need for the user to learn the language.

4 The Decisional Level

4.1 A Paradigm for Integrating Deliberation and Action

We summarize here the paradigm we have adopted in order to take into account the different attributes discussed in section 2.2.

A decisional level is composed of two independent entities: a planner and a supervisor (figure 2).

The *Planner* is given a description of the state of the world and a goal; it produces a plan. One criterion that should be considered when speaking about planning is the "quality" of the produced plan which is related to the cost of achievement of a given task or objective (time, energy, ...), and to the robustness of the plan, i.e., its ability to cope with non nominal situations. This last aspect is one of the motivations of our approach: besides providing a plan, the planner should also provide a set of execution "modalities". These execution modalities are expressed in terms of:
- constraints or directions to be used by a lower plan-

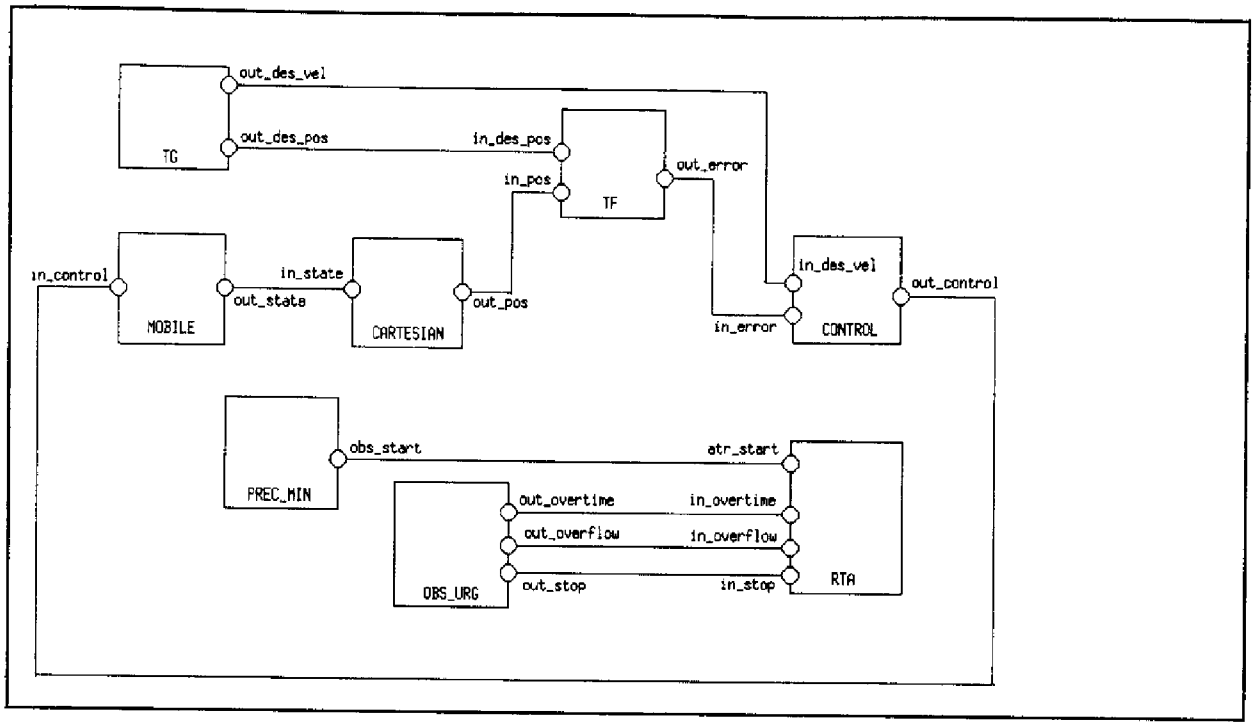


Figure 1: A robot-task example for parking a mobile robot

ning level;
• description of situations to monitor and the appropriate reactions to their occurrence; such reactions are immediate reflexes, "local" correcting actions (without questioning the plan), or requests for re-planning.

These "modalities" provide a convenient (and compact) representation for a class of conditional plans. However, the generation of "modalities" still remains to be investigated: we have no generic method yet for integrating modalities production in a planning algorithm. However, it is possible to produce useful modalities in a second step by performing an analysis of the plan as produced by the a first "classical" planning step. Such an analysis can be based on a knowledge of the limitations of the planner itself and of the world description it uses, as well as on domain or application specific knowledge (see section 5 for example).

The *Supervisor* interacts with the other layers and with the planner. The other layers are viewed as a set of processes which exchange signals with the supervisor. These processes correspond to the actions of the agent as well as events associated with environment changes independent from robot actions.

These processes are under the control of the supervisor which has to comply with their specific features. For example, a process representing a robot motion, cannot be cancelled instantaneously by the supervisor: indeed, such a process has an "inertia". The supervisor may request a stop at any moment during execution; however, the process will go through a series of steps before actually finishing.

The simplest way to represent such processes are finite state automata (FSA). More elaborate representations such as temporized processes should be investigated.

In the FSA we use, at any moment:
• the set of allowed external signals correspond to all the actions that can be taken by the supervisor;
• similarly, the set of possible internal signals correspond to all environment changes that could be perceived by the supervisor.

The activity of the supervisor consists in monitoring the plan execution by performing situation detection and assessment and by taking appropriate decisions in real time, i.e., within time bounds compatible with the rate of the signals produced by the processes and their possible consequences (Figure 3).

The responsibility of "closing the loop" at the level of plan execution control is entirely devoted to the supervisor. In order to achieve it, the supervisor makes use only of deliberation algorithms which are guaranteed to be time-bounded and compatible with the dynamics of the controlled system. Indeed, all deliberation algorithms which do not verify this property are actually performed by the planner upon request of the supervisor.

This execution control is done through the use of the

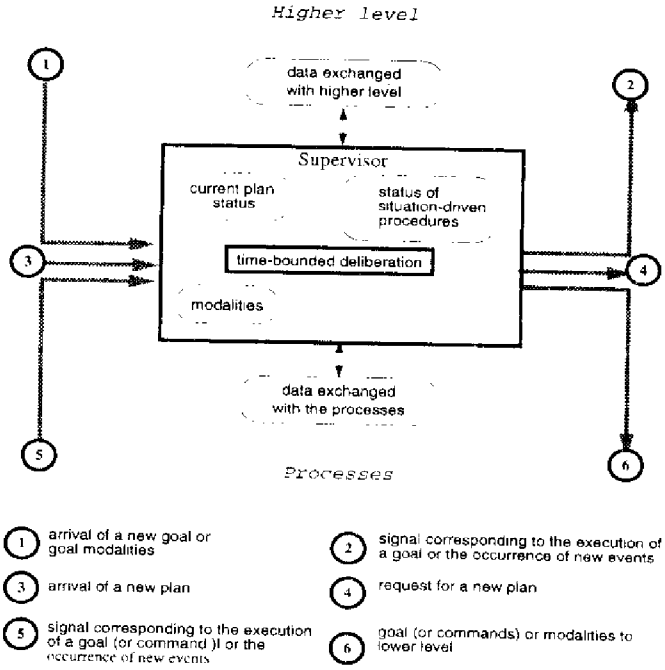


Figure 3: The Supervisor

plan and its execution modalities, as well a set of *situation-driven procedures* embedded in the supervisor and independent of the plan. These procedures are predefined (at design phase), but can take into account the current goal and plan when they are executed, by recognizing specific goal or plan patterns.

4.2 A Generic Architecture

We propose here below an architecture which is adapted to a class of autonomous robot applications where it is possible (and suitable) to describe the world model in an abstract symbolic level, and where the robot is given goals expressed in terms of a state. However, even though there is sufficient information to build and maintain such a description, this does not mean that detailed information is also available. Typical cases are "intervention robots" that have to operate in an ill-known environment discovered by the robot's sensors and on which only incomplete and uncertain previous knowledge may exist. Examples are disaster intervention (e.g., the AMR-EUREKA [16] project) or planetary rovers (e.g., VAP[1] [12].

The global system architecture we propose is organized into three levels representing two decisional layers and a functional level (figure 4). The two upper levels are built with the supervisor-planner paradigm. The higher level uses a temporal planner. The second level receives tasks that it transforms into scripts (procedures) composed of elementary robot actions, and supervises script execution while being reactive to asynchronous events and environment conditions. Planning at this level is a "refinement" using domain or task specific knowledge: it does not use a general planner.

[1]VAP: Autonomous Planetary Vehicle, a project of the French national space agency CNES.

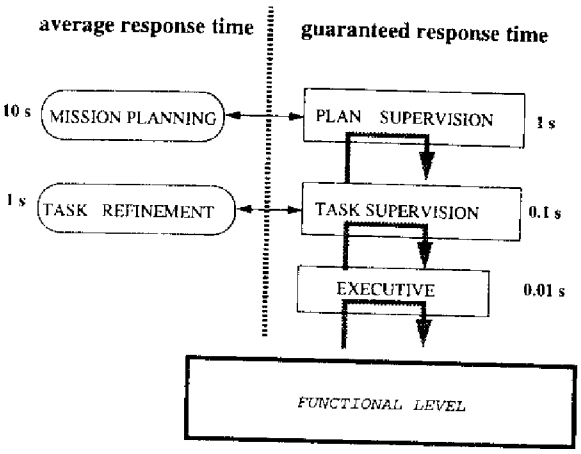


Figure 4: Global Architecture

The functional "execution" embeds a set of elementary robot tasks implementing task-oriented servo-loops as well as a set of robot primitive functions (motion planner, perception, etc...).

5 An Instance of the Architecture

We present next an example of the architecture which complies with the different properties discussed in the previous sections.

At The Mission Planning Level: For the planner, we have developed a temporal planning system called *IxTeT* (Indexed Time Table) [11, 9] which can reason on symbolic and numeric temporal relations between time instants.

A Plan, as produced by IxTeT, is a set of partially ordered tasks, together with temporal constraints such as minimal and maximal expected durations, and synchronization with expected external events or absolute dates.

In the current implementation, the temporal plan supervisor is provided with *automaton classes* corresponding to the execution of the different actions which may be included in the plan. Whenever an action is started an automaton of the associated class is instantiated. The "internal" events, as well as the "external" events, are represented as time-stamped transitions in the world model as it is used by the temporal planner.

At the task-planning level: At this level, we use C-PRS [15] which provides a suitable framework for goal-driven as well as situation-driven deliberation processes. Indeed, PRS implements script (called KA in PRS) se-

lection and goal posting mechanisms. Planning can be performed through context dependent goal decomposition; situation-driven reaction can be performed by triggering KAs according to the world model content.

At the executive level: This level is purely reactive with no planning capacities. It controls the execution of robot-tasks and other robot primitive functions using pre-defined context-dependent actions. It is implemented using a rule-based system KHEOPS [10] which allows to compile (off-line) a set of rules, producing a program which performs a time-bounded search through a decision-tree.

At the functional level: Concerning the functional level, an example of architecture is described is [19] and an implementation is given in [18]. It allows the user to specify complex robot-tasks with temporal properties, to check their performances in a discrete-time multi-rate framework and on a target hardware architecture. It also provides with automatic code generation for automaton encoding as well as for execution within a real-time operating system.

6 Conclusion

We have presented some key design issues of intelligent control architectures for autonomous robots. We have discussed the relevant attributes and showed how they can be integrated in a coherent architecture which provides a framework for implementing time-bounded decision and reaction at different levels, from task-oriented closed loops to situation-driven decision and goal-driven plan generation. We have also briefly presented several tools which can be used to implement instances of this generic architecture.

References

[1] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] J.G. Bellingham, T.R. Consi, "State Configured Layered Control, *Proc. 1st Workshop on Mobile Robots for Sub-sea Environments*, pp 75-80, Monterey, October 1990.

[3] A. Benveniste and G. Berry, "The Synchronous Approach to Reactive and Real-Time Systems", *Proceedings of the IEEE*, vol. 79, no 9, September 1991.

[4] R. F. Camargo, R. Chatila, R. Alami. Hardware and Software Architecture for Execution Control of an Autonomous Mobile Robot. IECON '92, San Diego, USA, Nov. 1992.

[5] G. Berry and G. Gonthier, "The Synchronous Esterel Programming Language : Design, Semantics, Implementation",*Science of Computer Programming*, vol 19, no 2, pp 87-152, Nov. 1992.

[6] R.B. Byrnes, "The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles", PhD Dissertation, Naval Postgraduate School, Monterey, USA, March 1993.

[7] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. " Integrated planning and execution control of autonomous robot actions". In *Proc. IEEE Int. Conf. on Robotics and Automation, Nice (France)*, 1992.

[8] E. Coste-Manière, B. Espiau and D. Simon, "Reactive Objects in a Task-Level Open Controller, *Proc. IEEE Conf. on Robotics and Automation*, pp 2732-2737, Nice, France, May 1992.

[9] C. Dousson, P. Gaborit, and M. Ghallab. Situation Recognition: Representation and Algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Chambéry, France, 1993.

[10] M. Ghallab and H. Philippe. A compiler for real-time Knowledge-based Systems. In *International Workshop on Artificial Intelligence for Industrial Applications*, Hitachi City, Japan, May 1988. IEEE.

[11] M. Ghallab, R. Alami, and R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles, editor, *Robotics Research: The Fourth International Symposium*. MIT Press, Mass., 1988.

[12] G. Giralt and L. Boissier. The French Planetary Rover VAP: Concept and Current Developments. IROS'92, pages 1391–1398, July 1992.

[13] G. Giralt, R. Chatila, and R. Alami. "Remote Intervention, Robot Autonomy, And Teleprogramming: Generic Concepts And Real-World Application Cases". IROS'93, Yokohama, japan, July 1993.

[14] A.J. Healey, R.B. McGhee e.a., "Mission Planning, Execution and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle", *Proc. 1st Workshop on Mobile Robots for Sub-sea Environments*, pp 177-186, Monterey, October 1990

[15] Ingrand, F. F., Georgeff, M. P., and Rao, A. S. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992. Also available as LAAS Technical Report 92-521.

[16] R. Laurette, A. de Saint Vincent, R. Alami, R. Chatila, and V. Pérébaskine. Supervision and Control of the AMR Intervention Robot. pages 1057–1062, June 1991.

[17] F. Noreils and R. Chatila. Control of mobile robot actions. In *IEEE, International Conference on Robotics and Automation, Scottsdale - Arizona*, pages 701 - 712, June 1989.

[18] P. Rives, R. Pissard-Gibollet, K. Kapellos, "Development of a Reactive Mobile Robot using Real-Time Vision" *Third Int. Symp. on Experimental Robotics*, October 28-30 1993, Kyoto, Japan

[19] D. Simon, B. Espiau, E. Castillo, K. Kapellos, "Computer-aided Design of a Generic Robot Controller Handling Reactivity and Real-time Control Issues", *IEEE Trans. on Control Systems and Technology*, to appear, fall 1993

Planet Exploration by Robots: From Mission Planning to Autonomous Navigation

Raja Chatila Rachid Alami Simon Lacroix Jérôme Perret Christophe Proust
LAAS-CNRS
7, avenue du Colonel-Roche
31077 Toulouse Cedex - France
{raja,rachid,simon,jerome,proust}@laas.fr

Abstract

We present in this paper an approach to planet exploration by mobile robots. We discuss the capacities that the robot should be endowed with according to the objectives of the mission, and we present a complete system that provides for mission planning and supervision from a ground station, and for on-board mission interpretation and execution, including autonomous navigation. The concepts are illustrated by results from the experimental testbed EDEN developed at LAAS.

I. Introduction

Several aspects make planet exploration a demanding and difficult problem for robotics:

- The robot has to operate in a natural, unstructured and a priori unknown environment.
- There is no possibility of continuous interaction with the robot because of important delays in communications and low bandwidth.
- The information on the robot and the environment is mostly acquired through the robot's own sensors.

These constraints rule out direct teleoperation as well as telerobotics approaches [18, 13, 12], and point towards robots with important autonomous capacities.

One approach that has been proposed [1, 15] is to send one or more "simple" and *completely* autonomous robots, without any control from a ground station. Such robots, using a behavior-based control scheme [3], would achieve an imaging, measurement or sample collection mission but not in a designated site, since they cannot be given their objective from the ground.

The perception and navigation systems of this type of robot would be simple (not considering the instruments necessary for the scientific measurements) since they don't plan their motion except locally. A local or contact-based obstacle avoidance algorithm and low-level security reflexes would be sufficient to carry out the mission. If it is necessary to get back to the landing module, these robots would be equipped with some radio device for homing back to the lander (without a real guarantee to reach it: they have no motion planning).

For the mission to be really useful if based on this concept, a number of robots should be sent since some could be lost and other could acquire a too poor data. Furthermore, the area explored by such robots would be limited to the surroundings of the landing module because of the simplicity of their navigation and the absence of path planning.

We think this approach would certainly be easier to implement and probably produce robust robots (because they are simple) but would be far from guaranteeing the success of the mission, neither its scientific harvest.

We have a different approach to the challenge of planet exploration by mobile robots [11]. Our approach stems from the following considerations:

- The landing site may be remote from areas of interest to the scientists, mainly because it will be selected for its safety whereas the interesting areas are in general rather inadequate for landing [5]. Hence the robot has to travel some distance (tens of kilometers or more) from the lander to reach a *specific region* (not at random nor merely in a given direction).
- The mission is not defined once and for all. According to returned data, the scientists on Earth should be able to decide for the exploration of such or such site, the analysis of such sample, etc. It is necessary then to be able to control the robot, i.e., send it new missions. It is therefore important to know what it is doing. In order to provide it with new objectives, it is also important to know where it is.
- Because the environment is poorly known, the mission can only be defined at a *task-level* in general, and not in its every detail (except in very special cases such as picking up a rock at reach). Hence the robot must be able to interpret the mission according to its actual context during its autonomous execution.
- The robot could fall into difficult situations wherein its perception, interpretation or decision making capacities are unsufficient. Human intervention would then be necessary for troubleshooting (which could be at a very low-level of command).

According to the preceeding arguments, we propose a global architecture for a robotic exploration system in two main parts: a ground station for mission programming and supervision, and a remote robot[1] able to interpret the mission and execute autonomously.

[1] not necessarily a single one.

II. The Ground Station

The Ground Station includes the necessary functions to allow a human to a) build a mission that can then be interpreted and executed by the robot. Such a mission is called an *executable mission*, as opposed to a higher level description of objectives as they may be expressed by planet scientists, and b) supervise its execution, taking into account the delays and communication constraints.

It consists of:

- A user interface that includes representations of the environment, the mission, and the robot state.
- A computer-aided environment to support the planning, programming and supervision of the mission by the human user.

The process of building an executable mission is decomposed into two phases which correspond to two different levels of abstractions and to different planning techniques:

1. a phase called "mission planning" which produces a "mission plan", i.e. a set of (partially) ordered steps that will allow the robot to achieve a given goal.
2. a phase called "teleprogramming" that consists in refining a step in the mission in terms of tasks that can be interpreted and then executed by the robot. Depending on the nature of the mission and its difficulty, and on the amount of information available at planning time, an executable mission can be composed of a variable number of steps.

A. *Mission Planning*

Mission planning can be carried out with the help of a planning system able to take into account temporal and resource constraints as they can be foreseen at this stage. We have developed a temporal planning system called *IxTeT* [9, 10] which can reason on symbolic and numeric temporal relations between time instants. It produces a set of partially ordered tasks with temporal constraints.

The explicit representation of time allows for a representation of planning operators that specify information concerning the duration of actions, the relative time when the consequences of an operator become true, the conditions which must remain true during action execution, joint effects with other operators executed in parallel, and the like. This has been partly addressed in planners like DEVISER [20] or FORBIN [6] that introduce temporal operators.

The descriptions of the world, the goals and the planning operators are given using symbolic and/or numeric temporal relations between time instants or elementary temporal relations between intervals which can be transformed into relations between instants.

IxTeT architecture is closer to FORBIN's, with an explicit time-map manager. Its representation is based on a reified logic formalism. It relies on a 2-dimensional array (called Indexed Time Table) with rows corresponding to domain relations and columns to temporal qualifications with time points at the primitive level. Temporal constraints between instants are represented by a time lattice. A temporal relation manager maintains the lattice and propagates temporal constraints [10].

Ixtet involves two levels of description for operators: the *Action* and the *Task*. *Actions* correspond to the lowest level of description. The 'effects' of an action are assertions which change as a direct result of performing that action. Hence, an action is described through its 'effects', temporally linked to the interval of execution of the action. Minimal and maximal duration of an action may also be specified.

For example, in the action description given in figure 1, *(Data-Available ?data ?site ?target)* becomes true at the end while *(Sending-Data ?target)* becomes and remains true during action execution.

```
(ixtet:Action ACTION-SEND-DATA
        :args (?data ?site ?target)
        :effects ( (:meets Data-Available ?data ?site ?target)
                   (:equal Sending-Data ?target))
        :duration-min (length ?data)
        :duration-max (length ?data))
```

Figure 1: An Action description

Tasks are the planning operators. They are defined in terms of a context (a set of conditions needed in order to apply the procedure), a set of temporally constrained actions required to achieve a particular goal, together with additional effects which are due to the combined execution of the actions involved by the task. In the example given in figure 2, in order to apply the task *SEND-PANORAMA*, the robot will have to wait until *(Sunlight)* becomes true before executing *ACTION-GET-PANORAMA(?site)*, and to wait for a visibility window with the target (Earth or Orbiter) and the rover, with a sufficient duration for performing *ACTION-SEND-DATA(panorama, ?site, ?target)*.

```
(ixtet: Task SEND-PANORAMA
    :args (?site ?target)
    :goal (Data-Available panorama ?site ?target)
    :context (   (:during s1 (:on (Sunlight)))
                 (:during s1 (:on (VAP-Site ?site)))
                 (:during s2 (:on (Visibility ?target))))
    :steps       (s1 (ACTION-GET-PANORAMA ?site)
                  s2 (ACTION-SEND-DATA panorama ?site ?target))
    :such-that ((:before s1 s2))
    :effects ())
```

Figure 2: A Task description

B. *Task-level Teleprogramming*

Depending on the nature of the task and on its difficulty with respect to environment conditions, and depending on the robot decisional and operational capacities, a task selected by the planner can be sent as it is to the robot or must be further refined at the ground station.

We call this process the "tele-programming phase". It uses

all the information and expertise available at the ground station which may help the robot in performing its task.

The result of this phase can be a more or less detailed program together with a set of execution "modalities" which provide a convenient representation for a class of conditional plans.

These execution modalities are expressed in terms of:

• constraints or directions to be used by the robot control system for executing the mission and each of its tasks ;

• a description of situations to monitor and the appropriate reactions to their occurrence; such reactions are immediate reflexes, "local" correcting actions (without questioning the mission), or requests for re-planning a task.

We can consider four types of tasks: global motion, perception tasks, local motion, and scientific tasks.

Global motion teleprogramming consists in replacing a "GO-TO(site)" task by a sequence of more robust global motions that guide the robot along a "good" route as it may be selected from data taken from orbit, and also relying on environment features - if any. The robot will be still executing autonomously its navigation, but instead of using only its own data, it also takes advantage of other obsevatons. In addition, execution modalities can be added to the task, to be used by the robot to take its own decisions. Such modalities include constraints and indications for selecting the adequate actions (e.g., decision to cross an uneven terrain, with respect to try to avoid it, at the price of a longer but easier trajectory). This teleprogramming phase ends up for the case of global navigation in constructing navigation routes as shown in figure 3.

Perception tasks are used to acquire specific data for local motion or other actions. Local motions or scientific tasks are those which concern positioning the robot either for sample picking or equipment deployment. Local motions can also be programmed for helping the robot to overcome some difficult situation. Such tasks need data sent back by the robot and cannot be programmed beforehand.

Telesupervision in this context has both a mission monitoring role and a troubleshooting role. Because of communication constraints (communication requires pointing the antenna and hence cannot be done continously) specific supervision commands such as status reports and data on mission execution must be included in the mission itself. In case of a problem encountered during execution, the robot must take the decision to call for help, or to continue the mission according to the modalities given with it.

III. The Robot System

Because the robot is in a remote ill-known environment, and communications constraints prevent from a continuous exchange of data with it, it is not possible in general to plan its actions with all the details. Therefore, the robot control system should be able to interpret the tasks in terms of actions to be executed, taking into account the actual state of the system and of the environment [2]. Nominal mission execution is completely autonomous and controlled on-board, without any direct interaction with the station (except if planned). Exchange of data with

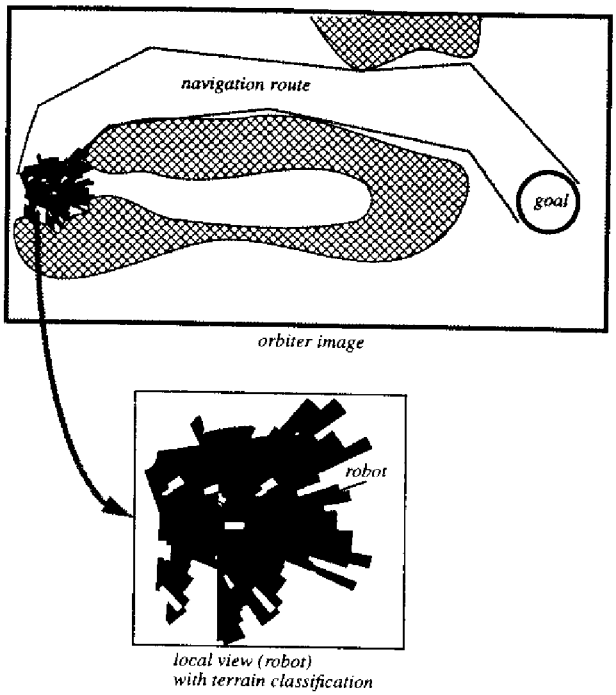


Figure 3: *Construction of a navigation route*

the Ground Station takes place as planned in the mission or when necessary because of execution status.

The robot system is composed of two levels: a decisonal level and a functional execution level.

A. *The Decisional Level*

The first level receives the tasks composing the plan, transforms them into sequences of actions and supervises their execution. It is composed of a **task refinement** system which selects the adequate actions - coded as procedures to execute - according to a current context, i.e., to mission, environment and robot state. This is a time-bounded process. The other component at this level is a **task supervisor** that is in charge of receiving the mission from the Ground Station and transmit it for refinement, and then controlling the execution of the selected procedures. This system is reactive to asynchronous events coming from the execution level. Its time response is also bounded.

B. *The Functional Level*

The second level is an "execution" level. It is composed of a set of modules embedding the functions necessary for perception, motion and other actions [17, 4]. The response time of these modules that implement polynomial time algorithms is bounded. This level is managed and controlled by a central **Executive** in order to execute the actions requested by the task supervisor. The executive is a time-bounded system: its reactions to events are predetermined precompiled structure.

A module embeds primitive robot functions which share

common data or resources [7]. An internal control process called the "module manager" is responsible for receiving requests to perform these functions from the robot controller, and for otherwise managing the module. Each function being well defined, its activation or termination must respect certain conditions that the module manager verifies. Modules interact by message passing or by reading data exported by other modules, and by putting their own processing results into exported data structures (EDS). At a given time, a module can be executing several functions.

IV. Autonomous Navigation

The global system as described in section III has been partially instanciated in the "EDEN" experiment carried out at LAAS with the mobile robot ADAM[2]. This robot is not to be sent to Mars; it is only used for development and testing.

We focus here on a fully autonomous navigation in a natural environment, including perception, environment modeling, localization, path and trajectory planning and execution on flat or uneven terrain. All the functions are encapsulated in modules integrated together in a real-time environment for autonomous operation.

The canonical task is "GO-TO (Landmark)" in an initially unknown environment that is gradually discovered by the robot. The landmark is an object known by its model, which can be recognized and localized on a video or laser image.

The robot has six motorized non directional wheels with passive suspensions[3]. It is equipped with a 3D scanning laser range finder (LRF), two orientable color cameras, and a 3-axis inertial platform (figure 4). The motion controller also provides for odometry. On-board computing equipment is composed of two VME racks, one for locomotion and attitude control, and the other for the perceptual and decisional functions.

A. *General approach*

In a natural terrain, there may be areas that are rather flat - cluttered or not by obstacles - and others which are uneven with respect to the locomotion capacities of the robot. Uneven areas require detailed modeling and complex trajectory planning, whereas flat areas can be more easily crossed, and motion planning is sometimes not necessary if the area is essentially obstacle-free.

According to a general economy of means principle, we chose to adapt the robot's behavior to the nature of the terrain, and hence consider three navigation modes :

- A **2D planned** navigation mode : Applied when the terrain is mostly flat - or with an admissible slope for the robot -, it relies on the execution of a planned 2D trajectory . The motion planner requires a binary description of the environment in terms of *Crossable/Non-Crossable* areas.

[2]ADAM: Advanced Demonstrator for Autonomy and Mobility is property of FRAMATOME and MATRA MARCONI Space, currently lent to LAAS.

[3]Robot chassis was built by VNII Transmash (St Petersbourg, Russia).

Figure 4: *The mobile robot Adam.*

- A **3D planned** navigation mode : Applied when an uneven area has to be crossed, it requires a precise model of the terrain, on which a 3D trajectory is planned and executed [19].

- And a **reflex** navigation mode : The robot locomotion commands are determined on the basis of *(i)* a goal (heading or position) and *(ii)* the information provided by "obstacle detector" sensors. This navigation mode does not require any modeling of the terrain, that has simply been labeled as "essentially obstacle-free".

Our approach to determine which navigation mode can be applied is based on a fast analysis of the raw 3D data produced either by the LRF or by a stereovision algorithm. This analysis provides a description of the terrain in terms of five classes : {*horizontal, Flat with slope, Uneven, Obstacle, Unknown*} (figure 5). This model is maintained (old and new data are fused) as the robot moves by to build a global description of the environment. All the "strategic" decisions are taken on the basis of this global representation. These decisions concern the determination of the intermediate positions, the choice of the navigation mode to apply to reach them, as well as the definition of the next perception task to execute: sensor orientation (and selection, since Adam is equipped with stereo and LRF), and operating modalities.

B. *Terrain representations*

As mentioned above, the different navigation modes require different types of terrain description, and the strategic decisions are taken on the basis of a specific representation of the environment. The localization processes also requires specific representations: a localization process based on landmark recognition for instance, needs geometric features to model and match the perceived landmarks, and it also uses a global map of landmarks or areas of interest.

Several data structures that represent the environment must coexist in the system. We developed a multi-layered heterogeneous model of the environment: in such a model, the different representations are easily managed and a global coherence can be maintained.

The top-level of this heterogeneous model is a "bitmap" description of the environment (figure 5), built upon the results of the quick terrain analysis algorithm. A large amount of information is available in each pixel of this bitmap, such as the terrain label and its confidence level, the estimated height, the identification of the region it belongs to (section IV.C) etc. We have chosen such a structure for the following reasons : it is simple, rich, adapted to a unstructured terrain, and open, in the sense that any complementary information can easily be encoded in a pixel whithout reconfiguring the entire description and the algorithms that runs on it. Moreover, the techniques that allow to extract structured informations (regions, connexity...) from a bitmap are well known and easily implemented.

C. *Navigation strategies*

Navigation strategies include *(a)* perception strategies, *ie.* the choice and definition of the different perception tasks to perform, and *(b)* the motion strategies, that imply the definition of intermediate goals and the choice of navigation modes; The two problems are obviously linked, but to avoid a great complexity, we developed two specific independant techniques, that are coupled afterwards to obtain an adequate behavior of the robot.

• **Perception strategies**: they mainly concern the decisions to acquire more detailed information on some regions (e.g., building an elevation map to cross an uneven area), or to perceive a new zone, or robot localization. They are always performed according the following procedure: *(a)* perceptual constraint check, *(b)* prediction of the result of the perceptual task, and *(c)* evaluation of the contribution of the task. The first two steps need an explicit model of the environment (in our case the bitmap description), and but also of the sensor to be used (in terms of logical sensors such as " landmark extractor", or "a terrain analyzer" for instance), and the third step is computed heuristically, considering the current mission constraints.

Fast terrain analysis [14]: This procedure, performed each time 3D data are acquired, relies on a discretization in "cells" of the perceived zone, and on the determination of global attributes for these cells that allow to label them (figure 5). The main point is that labels are associated to a confidence value, which allow to fuse different perceptions in a global model (the bitmap structure mentioned above).

Elevation map building [16]: When a uneven area has to be crossed, it is more precisely modeled using and interpolation algorithm to produce a digital elevation map on a cartesian grid.

Landmark based localization [8]: Localization using environment features has been developed. Currently, it mainly relies on the recognition of terrain "peaks" (points of important elevation).

Perception processes are autonomously activated and controlled, according to mission constraints and knowledge on the environment.

• **Motion strategies**: The model structure used for path planning and subgoal selection is an adjacency graph of

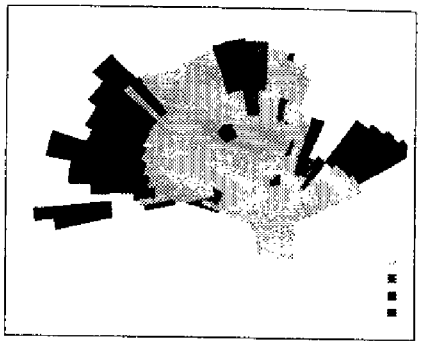


Figure 5: Bitmap model after 5 perceptions...

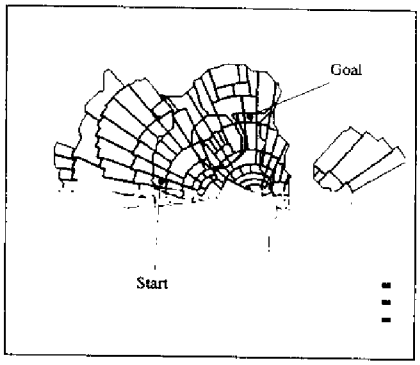


Figure 6: ... and the corresponding region structure

regions on which a heuristic search can be performed (figure 6. Regions are areas of uniform label, confidence and height: their crossing cost is defined taking into account these characteristics, and also the quality for localization, depending on the mission context. The heuristic search provides intermediate goals; navigation modes are deduced from the labels of the crossed regions.

In the current implementation, an intermediate goal search is first performed, and the perception tasks to execute are determined according the results of this search: elevation map building is performed on the uneven regions to cross.

V. Conclusion

We have presented a complete system for operating an autonomous rover for planet exploration missions, together with all the ingredients which implement the robot decisional and operational autonomy. This system is based on a generic architecture for intervention robots we are developing for several highly demanding applications. It has been implemented and demonstrated using an all terrain mobile robot performing autonomous navigation tasks in an unknown natural environment. The experimentation of the full system including the mission planning and teleprogramming phases is under way.

Acknowledgments

The authors wish to thank all the members of the EDEN team, and especially Matthieu Herrb.

REFERENCES

[1] C. M. Angle, R. A. Brooks. Small Planetary Rovers. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '90), Tsuchiura (Japan)*, pages 383-388, July 1990.

[2] R. Chatila R. Alami, B. Degallaix, and H. Laruelle. Integrated Planning and Execution Control of Autonomous Robot Actions. In *Proc. IEEE Int. Conf. on Robotics and Automation, Nice (France)*, 1992.

[3] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, April 1986.

[4] R. Chatila and R. Ferraz De Camargo. Open architecture design and inter-task/intermodule communication for an autonomous mobile robot. In *IEEE International Workshop On Intelligent Robots and Systems, Tsuchiura, Japan*, July 1990.

[5] F. Costard *et al.*. A Reference Martian Mission for a Long Range Rover. In *Misions, Technologies and Design of Planetary Mobile Vehicles*, Toulouse, France, Sept. 1992.

[6] T. Dean, R.J. Firby, and D. Miller. Hierarchical Planning Involving Deadlines, Travel Time, and Resources. *Comput. Intell.*, 1988.

[7] R. Ferraz De Camargo, R. Chatila, and R. Alami. A distributed evolvable control architecture for mobile robots. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, pages 1646-1649, 1991.

[8] P. Fillatreau and M. Devy. Localisation of an Autonomous Mobile Robot from 3D Depth Images using Heterogeneous Features. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '93) Japan*, July 1993.

[9] M. Ghallab, R. Alami, and R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles, editor, *Robotics Research: The Fourth International Symposium*. MIT Press, Mass., 1988.

[10] M. Ghallab and A. Mounir Alaoui. Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In *11th International Joint Conference on Artificial Intelligence (IJCAI), Detroit, Michigan (USA)*, pages 1297-1303, 1989.

[11] G. Giralt and L. Boissier. The french planetary rover vap: Concepts and current developments. In *IROS-92, Raleigh (USA)*, July 1992.

[12] S. Hirai and T. Sato. Motion Understanding for World Model Management of Telerobot. In *Robotics Research : The Fifth International Symposium, Tokyo (Japan)*, pages 5-12, 1989.

[13] G. Hirzinger, J. Heindl, and K. Landzettel. Predictive and Knowledge-based Telerobotic Control Concepts. In *IEEE International Conference on Robotics and Automation, Scottsdale, (USA)*, pages 1768-1777, 1989.

[14] S.Lacroix, P. Fillatreau, F. Nashashibi, R. Chatila, and M. Devy. Perception for autonomous navigation in a natural environment. In *Workshop on Computer Vision for Space Applications, Antibes, France*, Sept. 1993.

[15] D. P. Miller, R. S. Desai, E. Gat, R. Ivlev and J. Loch. Experiments With a Small Behaviour Controlled Planetary Rover. In *Misions, Technologies and Design of Planetary Mobile Vehicles*, Toulouse, France, Sept. 1992.

[16] F. Nashashibi, M. Devy, and P. Fillatreau. Indoor Scene Terrain Modeling using Multiple Range Images for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, pages 40-46, May 1992.

[17] F. R. Noreils and R. G. Chatila. Control of mobile robot actions. In *IEEE International Conference on Robotics and Automation, Scottsdale, (USA)*, pages 701-712, 1989.

[18] T. Sheridan. Telerobotics. In *IEEE Int. Conf. on Robotics and Automation (Workshop on Integration of AI and Robotic Systems*, 1989.

[19] T. Siméon and B. Dacre Wright. A Practical Motion Planner for All-terrain Mobile Robots. In *IEEE International Workshop on Intelligent Robots and Systems (IROS '93) Japan*, July 1993.

[20] S.A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3), May 1983.

How To Tele-Program a Remote Intelligent Robot

Jérôme Perret Christophe Proust Rachid Alami
Raja Chatila
LAAS-CNRS
7, Ave. du Colonel Roche, 31077 Toulouse Cedex - France

Abstract

How to program a remote intervention robot with a high degree of autonomy and reasoning abilities? We have proposed a control strategy, called Task-Level Teleprogramming, based on a specific architecture. After a brief presentation of our approach, we discuss in this paper the recent developments, mainly a Teleprogramming language, and illustrate them with an example.

1 Introduction

An ever increasing number of application fields (undersea intervention [13], planetary exploration [4], work in nuclear plants, disaster intervention [11], etc) show the need for very robust robots with a high degree of autonomy. Indeed, in those areas, the Telerobotics approach is of little help, the main reasons being:

- the environment is not known well enough beforehand in order to simulate or model it,
- the communication link is too constrained in terms of capacity and/or time lag.

All these constraints require Remote Intervention Robots to have all the attributes of operational and decisional autonomy: the environment being ill-known, the task cannot be completely specified in advance.

There is a need for Tele-Programming.

We will first discuss in section 2 what this programming means and how it can take profit of on-board decisional abilities. We will then briefly describe a specific control architecture which complies with this needs (section 3). Then we will present in section 4 how such programming can be performed and what it produces. And finally we will give an illustrative example in section 5.

2 What does Teleprogramming mean?

The Teleprogramming strategy has already been presented on several occasions: [5, 4]. We give here an overview of the subject.

Teleprogramming opposes Teleoperation, because the operator is not in the perception-control loop, and Telerobotics [9, 14, 12] because the operator does not perform the action, even remotely.

Teleprogramming consists in transmitting the task to be achieved in the form of a *program*, along with any information which the operator thinks could be useful to the robot. The abstraction level of that program depends on the refinement capabilities on board. Thus, we distinguish *Task Level Teleprogramming*, which makes use of the decisional capacities of the robot.

The Teleprogramming scheme divides into two aspects: first, to describe in advance the tasks to be performed and the situations to be recognized; second, to assist the robot in case of failure or unforeseen situations. In order to insure robustness, the first aspect must prepare the second. Indeed, the operator can assist the robot only in a stable situation, that is in a situation which is not evolutive (in order to give time to the operator to react, considering the communication constraints), not dangerous for the robot, and of course in which the robot can communicate with the operator.

Consequently, the robot must be able to identify and to reach a stable situation (or to avoid any unstable one) in all cases. To achieve this primary condition, the operator will use every resource available to supply the robot with useful and usable knowledge. Typically, the *Teleprogram* will carry along:

- processed information coming from other sensors, unavailable to the robot (e.g. a planetary orbiter),
- indications about the context of the task,
- specification of (unstable) situations beyond the interpretation capacities of the robot (e.g. quicksand),

- actions to be taken in the above situations (e.g. stop and move back!).

To illustrate the Task-Level Teleprogramming approach, let us consider an exploration robot on the surface of the planet Mars. The time delay between Mars and the Earth can be up to 20 minutes (40 minutes round trip), and the available data flow very small (a few kbits per second) within narrow communication windows. Therefore, neither Teleoperation (needing no time delay and a large data flow), nor Telerobotics (needing precise and frequently updated knowledge of the environment) is affordable.

A typical task could be: the robot has to move from a place called `site1` to `site2`, at a few hundred meters' distance. We will suppose that it can perceive its environment and plan a trajectory around obstacles. Is this sufficient to assure robustness? Surely not, for the surface of Mars is far from robot-friendly. In order to insure robustness, the operator will supply the robot with a program including:

1. the description of the task:
   `goto (site1, site2)`,
2. the location of possible dangers (pits, possibly unstable terrain, quicksand, etc) detected in the orbiter's images for example,
3. the location of known dead ends, or if possible a valid path to the goal (at orbiter's resolution),
4. how to detect quicksand (motor overload or odometry errors) and what to do in that case (backtrack!),
5. what to do in case of trouble (go to a safe place and call back!).

3 The Architecture

We contend that this class of problems demands a specific architecture which couples "intelligence" at the programming level with "intelligence" at the level of the physical machine.

We have developed an architecture composed of two systems (Figure 1):

1. The *Operator Station*, which includes modules and facilities for mission planning, task-level programming and supervision.

   Given a set of goals (mission) to be achieved, the operator station plans and then refines the mission, generating a set of tasks, to be interpreted and performed by the robot, along with their "execution modalities".

2. The *Robot Control System* which possesses all the functional capabilities of an autonomous robot.

   The *Robot Control System* is derived from the architecture for complete autonomy presented in [1], in which the task planning component is deported on the operator station, having more powerful computers as well as computer-aided facilities and human expertise at its disposal [8, 2]. It is organized into three levels.

   The higher level is composed of a *Supervisor* which interacts with the operator station and the next level (viewed as a set of processes which exchange signals with it).

   The second level is composed of a supervisor and a task refinement planner.

   The activity of the supervisors consists in monitoring plan execution at their level by performing situation detection and assessment and by taking appropriate decisions in real time. In order to achieve this, the supervisor makes use of deliberation algorithms which are *guaranteed to be time-bounded* and compatible with the dynamics of the controlled system. Indeed, all deliberation algorithms which do not verify this property are actually performed by the planner (on-board or at the Operator's Station) upon request of the supervisor.

   Note that in this architecture, on-board planning is necessary only at the second level. It is essentially a "refinement" using domain- or task-specific knowledge. For this, we use C-PRS [10] which provides a suitable framework for goal-driven as well as situation-driven deliberation processes. Indeed, PRS implements script (called KA in PRS) selection and goal posting mechanisms. Planning can be performed through context-dependent goal decomposition; situation-driven reaction can be performed by triggering KAs according to the environment model.

   The lowest level includes the robot modules that perform perception and action execution.

Such an architecture allows a level of robot autonomy which is essentially dependent upon the difficulty of the task and the state of the environment.

4 Task-Level Teleprogramming

4.1 A Language for Teleprogramming

The first point is, in order to transmit a program, we need a *programming language* to express it. What elements of the language can we deduce from the application characteristics?

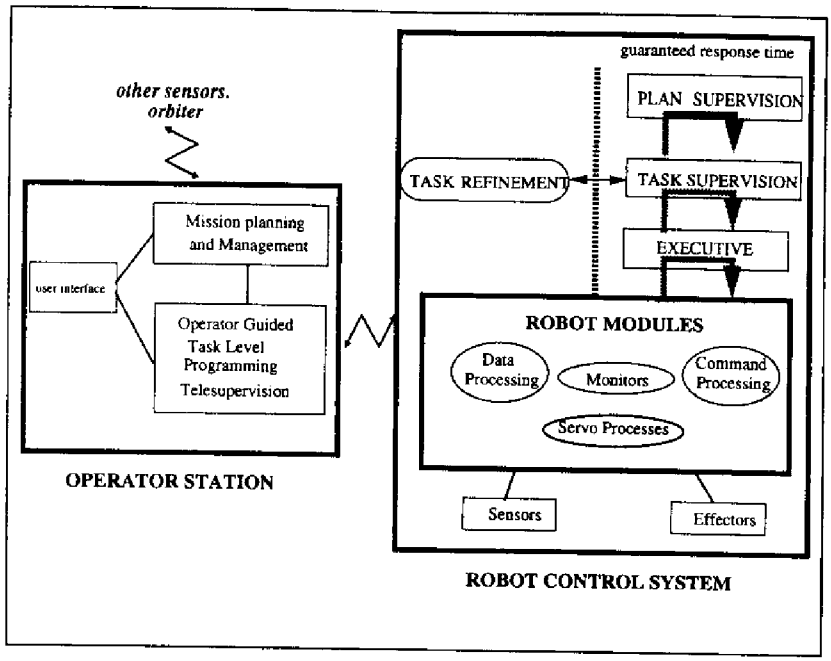


Figure 1: An architecture for Remote Intervention Robots

The program is to be merged into the robot's control system, so it has to share the same opening towards real-time processing. We advise that it include:

1. event-driven control structures
2. multi-threading
3. on-board procedure calls

The introduction of event-driven control structures will make program verification possible, as in ESTEREL [3].

To take advantage of the decisional abilities of the robot, we further need:

4. goal-manipulation primitives, such as posting, suspending, cancelling, testing
5. conditional and relational structures necessary to express the organization of a mission in terms of goal/task sequencing, etc.

Finally, we introduce a new concept, called:

6. modalities

Modalities are added pieces of information, intended for the on-board reasoning system. They are of three different sorts: advice, constraints, and additional data. Modalities can be attached to the entire program, or to individual tasks, in which case their scope covers all sub-tasks and posted goals.

4.2 The Task-Level Teleprogramming Environment

Once we have designed a programming language, we need a *Task-Level Teleprogramming Environment*. Typically, it will include:

- mission planning,
- mission refinement,
- mission supervision.

At the mission planning level, the operator describes the mission in terms of results to be achieved, goals to be reached, temporal relations and numerical constraints, and so forth. The planning module produces a set of tasks according to that description, be it the result of an actual planner or simply of a scheduling algorithm. The degree of completeness of the plan depends on the reasoning abilities of the robot: at the lower level, the plan will include all situations and reactions; at the upper level, the handling of abnormal events will be left to the robot, as well

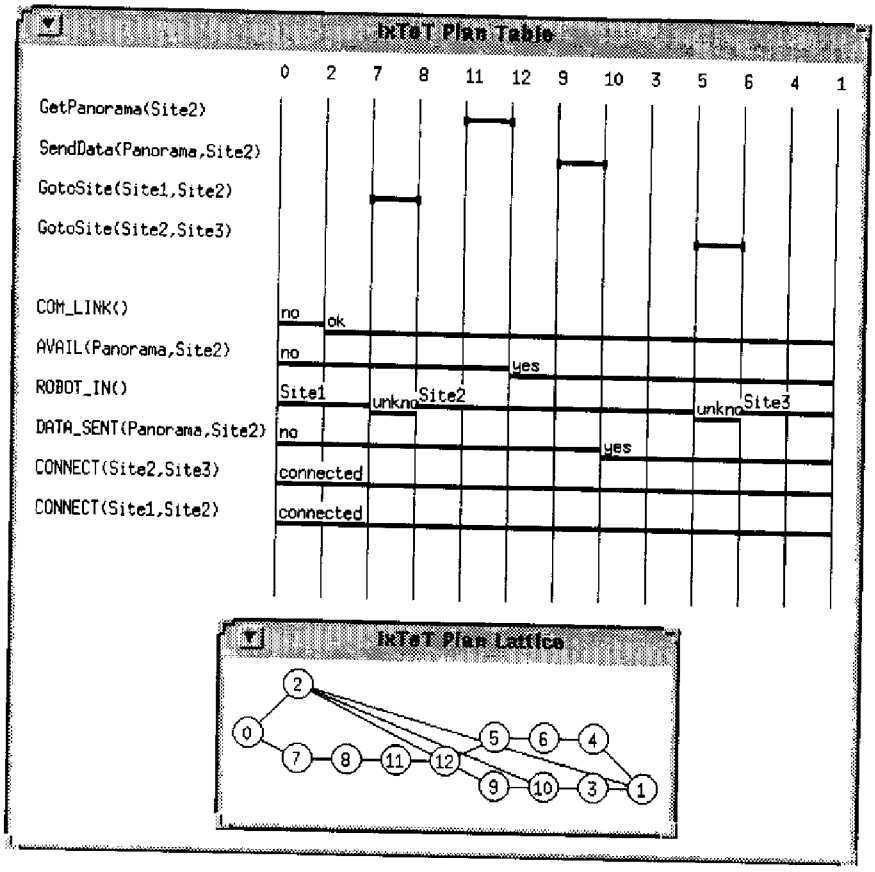


Figure 2: The IxTeT plan

as a substantial part of the task refinement. Actually, task refinement generally requires a large amount of context-dependent information available only to the robot. It entails that using a robot with high-level on-board reasoning abilities can lead to a great improvement of the mission scope, i.e. the extent of the mission which can be achieved without intervention of the operator.

At the mission refinement level, information will be added to the mission plan, such as modalities or the description of tasks not yet defined. Various types of verification algorithms will be used in order to increase confidence of the mission's robustness. The mission supervision will be anticipated, and its representativity will be checked.

At the mission supervision level, the robot's reports will be used for synchronization of the supervision module, and the mission progress will be evaluated.

The special case of recovery from failure is addressed much in the same fashion, except that the produced program does not have the same meaning. First, the robot's situation is evaluated, and the operator defines the recovery strategy, which is either the replacement of the former mission program by a new one, or its suspension for the duration of the recovery, which is, in that case, handled by a sub-program.

The main reasons why recovery from failure is left to the operator is that the robot is not aware of the general purpose of the mission, and that its interpretation abilities are limited.

5 An Implementation Example

In this section, we will present how the Task-Level Teleprogramming concept can become an actual robot control procedure. As a matter of fact, such a system has already been partially instantiated in the "EDEN" experiment carried out at the LAAS [4]. In an attempt to keep it short, we have chosen a very simple example, which does not pay credit to the actual development effort.

3-4

5.1 Mission Planning

For the planner, we have developed a temporal planning system based on *IxTeT* (Indexed Time Table) [7, 6] which can reason on symbolic and numerical temporal relations between time instants.

A Plan, as produced by IxTeT, is a set of partially ordered tasks, together with temporal constraints such as minimal and maximal expected durations, and synchronization with expected external events or absolute dates.

This set of tasks is the spinal axis of the mission plan, actually forming the *nominal plan*.

As an example, let us consider the planetary rover again, and the following mission: proceed from area called `site1` to area `site2`, there take a video panorama, send it to Earth, and finally proceed to `site3`, in order to be there before nightfall. Given the appropriate task models, the IxTeT planner will produce the plan shown in figure 2, provided the overall time constraint ("before nightfall") is not too tight. The numerical constraints between time instants are not represented here.

In this example, we have chosen to use very simple task models (consisting in two instants: begin and end) for the sake of demonstrativity[1]. According to these models, the two tasks `GotoSite` and `SendData` do not share the same resources, so the planner leaves them unordered. What this means is, they both occur after instant 12 (when `GetPanorama` is finished), and after instant 2 in the case of `SendData(Panorama,Site2)` (when the communication with the Earth becomes possible). In the case of the planetary rover, the physical machine cannot execute both tasks at the same time, because talking to the Earth means unfolding the antenna, pointing it, etc. But ordering these two tasks would be very constraining, considering that we do not know in advance which instant occurs first, 2 or 12.

5.2 Plan Refinement

The IxTeT data structures are entirely planning-oriented and do not provide enough flexibility for our purposes. Thus, we operate a change of representation. The new representation has to provide conditional control structures as well as parallel execution, while remaining expressive and simple enough to use a graphic display.

The transposition uses a new, non-deterministic model of the tasks. Hereafter is an example:

```
Task GotoSite(?Site1,?Site2)
{
 Prerequisite robot_at_site():?Site1;
 Resource locomotion(1);
 Resource energy(150);
 Nominal success(travel_time(?Site1,?Site2))
 {
  robot_at_site() = ?Site2;
  Use[locomotion] = 0;
  Use[energy] = travel_expense(?Site1,?Site2);
 };
 Other failure([00:30:00,06:50:00])
 {
  robot_at_site() = current;
  Use[locomotion] = 0;
  Use[energy] = 240;
 };
 Other hardware_breakdown([00:00:00,06:50:00])
 {
  robot_at_site() = current;
  Use[locomotion] = 1;
  Use[energy] = 240;
 };
};
```

This description includes preconditions and effects, as well as resource consumption and duration of the task along each possible path. The plan produced by IxTeT corresponds to the *nominal* path of each task. The procedure is as follows: the new operators are instantiated once for each task in the IxTeT plan, and ordered according to the instant lattice; then, all pending leaves of the resulting graph are tested for stability (does the robot know how to get out of that situation?) and unstable ones are signalled to the operator. The operator can modify and/or append the graph using a graphic display and tools for the verification of preconditions and resource consumption.

Figure 3 shows an example. Here, the operator decided that the robot proceed to `Site3` should it fail to reach `Site2`; as well if it is short of time after completing the task `GetPanorama`. Of course, all events not made explicit should be handled by the robot itself.

Finally, execution modalities can be attached to the plan, in order to specify global constraints or default actions such as: *in case of trouble, call home.*

5.3 Task Programming

At this point, the plan skeleton is complete, but the tasks need yet to be refined.

If a task is resident on the robot, the only programming required is the definition of the modalities. Otherwise, a task program must be supplied.

Again, in our example, we suppose that the task `GotoSite()` is already known to the robot (see §6). As we do not want to develop in detail in this paper, please refer to [4] for an illustration of how modalities can be produced, such as navigation routes and landmarks in the

[1] The present plan is by no means representative of the capacities of the IxTeT planner.

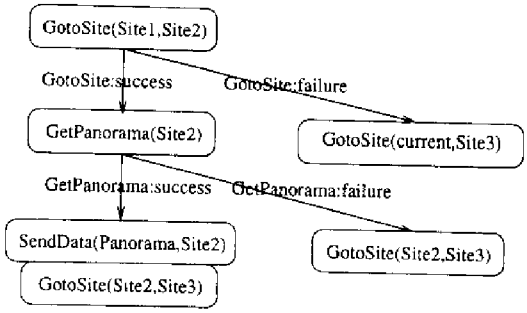


Figure 3: The final plan

case of a GotoSite task. We will simply note that this process is mostly task-dependent and can require specialized tools (e.g. a path planner).

On the contrary, let us suppose that the task GetPanorama is not yet known by the robot control system. The operator has to supply a program for that task. Here is a simplified example:

```
Task GetPanorama ( place ) {
  do {
    exec check_robot_at_place(place);
      report RESULT;
    if ( is_false(?RESULT) )
      raise END("context_failure");
    post point_camera(0.00)
      report STATUS;
    if ( is_failed(?STATUS) )
      raise END("hardware_failure");
    loop {
      post get_picture();
      post move_camera(30.00);
      exec check_camera_position(0.00)
        report RESULT;
      if ( is_true(?RESULT) )
        raise BREAK;
     } watching BREAK;
    raise END("success");
   } watching END;
} export END;
```

Using the model of the procedures called in GetPanorama, the system can certify that this program corresponds to the non-deterministic task-model used in the "plan refinement" phase.

In our current implementation, the task program is written as a PRS-KA, in order to be executed on board.

5.4 Telesupervision

The same reasons that led us to consider the Task-Level Teleprogramming approach make mission supervision a difficult problem. Indeed, the intervention robotics paradigm involves that any communication with the operator be the result of an action undertaken by the robot, possibly involving complex decisions (e.g. moving to a clear place).

We distinguish two cases. In the first case, the operator can obtain information from other sensors (e.g. the orbiter [4]); he can verify whether the mission is proceeding smoothly or not, but cannot influence the robot, short of an emergency call (not always possible). In the second case, no such sensor is available, and the operator depends entirely on the robot for the mission supervision; in that case, regular check-points must have been planned in advance. We introduce *Telesupervision Directives* as special tasks inserted into the mission plan. These tasks are responsible for transmitting the mission progress reports, along with any information specified by the operator. Two modes of operation are possible: blocking or non-blocking.

We will not demonstrate the Telesupervision Directives here, our example being inadequate. We will suppose that the operator is content with having only that much visibility provided by the *mission completion* report.

5.5 Producing the Program

After the Telesupervision Directives have been inserted, two mission programs are produced, one for the robot and one for the Telesupervision module, which will be evaluated during the mission and synchronized using the data received from the robot according to the Telesupervision Directives.

The robot's version of the mission program includes the final mission plan, the modalities, the description of new tasks, and all necessary pieces of data. The plan is expressed as a data structure, consisting in a set of tasks, defined with their arguments, temporal constraints (relative to the start and end point, as well as duration) and modalities, connected by transitions labelled with internal and external events.

6 Plan Execution

6.1 Overall Strategy

The plan supervision consists in sequencing the tasks according to expected events specified in the plan (begin and end events of the tasks, and time-synchronization events) as well as unspecified (for instance, task failure not addressed in the plan). In case of conflict between two tasks, the plan supervisor is responsible for deciding which task should be executed or interrupted and for enforcing that decision.

Each task in the plan corresponds to the execution of one or several procedures. According to the tasks and to the execution context, the procedures are either selected because they are explicitly designated in the task plan, and are then intanciated for execution, or are selected as a result of goal posting. In this case the selection of a procedure follows the general scheme of PRS and is based on some invocation conditions and on the context of execution as expressed in the data base. The choice of the best procedure, when several are possible candidates, is made by a meta-procedure that reasons on applicability criteria. Procedure selection is an iterative process.

The execution of a procedure may produce several outcomes. The plan explicitly provides the desired chaining between the tasks according to *some* of this outcomes (§5.2). If this chaining is not explicit in the plan, default procedures are selected (or goals) and executed by the supervision system. Usually, such procedures will put the robot in a safe and stable situation, and try to communicate with the ground station.

As an example, we describe the procedure `GotoSite` (see figure 4), which loops until the robot has reached the target site. Executing this procedure makes the system post new goals, and select new procedures that will eventually result in executing some actions (e.g., perception, trajectory planning, etc.) and so on. In the EDEN experiment, this task is implemented as a set of PRS-KAs.

Here the robot starts by acquiring new data on the environment, and decides, on the basis of a first modelling of the terrain, which navigation mode should be selected. Two modes are possible: reactive and planned. The reactive mode is selected in case of a flat terrain almost free of obstacles. It makes the robot move toward the goal while trying to detect obstacles - without a full analysis of the terrain. In the planned mode, a navigation map is built, and a trajectory planner is selected to compute a collision free trajectory (either on flat or uneven terrain).

The selection of subgoals for navigation depends on the modalities associated with the plan, such as the navigation routes and landmarks.

6.2 Sample Execution

When the mission plan is received by the robot, the supervision system on board first loads the new procedures, then the new plan is initiated.

Let us go back to the plan showed figure 3. The first task to be executed is the navigation from `Site1` to `Site2`. Accordingly, the plan supervisor asks the task supervision level to launch the task `GotoSite(Site1,Site2)`, and then waits for the termination event (either `GotoSite:success`, `GotoSite:failure`, or `GotoSite:hardware_breakdown`)

```
task GotoSite (site) {
  loop {
    exec check_robot_at_site (site)
      report RESULT;
    if ( is_true (?RESULT) )
      raise END("success");
    post get_environment_model ()
      report MODEL;
    post choose_navigation_mode (?MODEL)
      report MODE;
    if ( equals(?MODE,#reactive) )
     {
      fork watch_site_entry (site)
        report SITE_REACHED;
      exec move_until_obstacle ()
        watching SITE_REACHED
        report STATUS;
      if ( is_true(?SITE_REACHED) )
        raise END("success");
     }
    else
     {
      post find_sub_goal (?MODEL)
        report SUB_GOAL;
      post find_trajectory (?MODEL,?SUB_GOAL)
        report TRAJECTORY;
      if ( is_void(?TRAJECTORY) )
        raise END("failure");
      exec follow_trajectory (?TRAJECTORY);
     }
   } watching END;
}
```

Figure 4: The GotoSite task program

while monitoring the temporal constraints.

The task supervisor then updates the execution modalities and posts the PRS-goal corresponding to the task. The task refinement level (see figure 1) selects a suitable KA for achieving the goal with respect to the execution context and the modalities. When the goal is fulfilled or recognized as unreachable, the task supervisor generates the task-termination event.

A number of outcomes are possible: if the robot reaches a deadlock, the plan supervisor proceeds with `GotoSite(unknown,Site3)`; if a breakdown occurs, it calls for help; if the task takes too much time, it requests it to stop. Let us suppose that the robot managed to reach `site2`. At this point, the picture acquisition task is launched, which results in the KA `GetPanorama` being selected at the task supervision level. In the special case

of the two concurrent tasks `SendData` and `GotoSite`, the plan supervisor has to suspend the navigation task as the communication link becomes available, in order to execute the data sending task.

Any event not specified in the mission plan, such as a hardware breakdown, would cause the abortion of the plan execution and the activation of a default recovery procedure. This procedure would place the robot in a safe and stable state and contact the ground station for further instructions.

7 Conclusion

We have presented a paradigm for the planning, refinement and execution of tasks by a remote robot, called Task-Level Teleprogramming. This paradigm provides a high degree of operational and decisional autonomy. We have described a specific architecture, which combines "intelligence" at the programming level with "intelligence" at the level of the physical machine. This architecture has already been partially implemented on an all-terrain robot (EDEN project), and has been chosen for the Eureka-project IARES, which aims at developing a demonstrator for a planetary exploration robot.

References

[1] R. Alami, R. Chatila, and B. Espiau. Designing an intellingent control architecture for autonomous robots. In *International Conference on Advance Robotics*. ICAR'93, Tokyo (Japon), Novembre 1993.

[2] R. Alami, R. Chatila, and P. Freedman. Task level programming for intervention robots. In *IARP 1st Workshop on Mobile Robots for Subsea Environments, Monterey, California (USA)*, pages 119–136, Octobre 1990.

[3] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, vol. 19(no 2):pp 87–152, Novembre 92.

[4] R. Chatila, R. Alami, S. Lacroix, J. Perret, and C. Proust. Planet exploration by robots : from mission planning to autonomous navigation. In *International Conference on Advance Robotics*. ICAR'93, Tokyo (Japon), Novembre 1993.

[5] B. Degallaix. Une architecture pour la téléprogrammation au niveau tache d'un robot mobile autonome. Thèse de l'Université Paul Sabatier, Toulouse (France) 1372, Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), Toulouse (France), 1993.

[6] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: Representation and algorithms. In *13th International Joint Conference on Artificial Intelligence (IJCAI), Chambery (France)*, 1993.

[7] M. Ghallab, R. Alami, and R. Chatila. Dealing with Time in Planning and Execution Monitoring. In R. Bolles, editor, *Robotics Research: The Fourth International Symposium*. MIT Press, Mass., 1988.

[8] G. Giralt, R. Alami, and R. Chatila. Autonomy versus teleoperation for intervention robots? a case for task level teleprogramming. In *Intelligent Autonomous Systems-2, Amsterdam, Netherlands*, Décembre 1989.

[9] G. Hirzinger. Rotex - the first robot in space. In *'93 International Conference on Advanced Robotics (ICAR), Tokyo (Japan)*, pages 9–17, Novembre 1993.

[10] F. Ingrand, M.P. Georgeff, and A.S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert. Intelligent Systems and Their Applications*, 7:pp.34–44, 1992.

[11] R. Laurette, R. A. de Saint Vincent (Matra-Espace) R. Alami, C. Chatila, and V. Pérébaskine (LAAS/CNRS). Supervision and Control for the AMR Intervention Robot. In *5th International Conference on Advanced Robotics (ICAR '91), Pisa (Italy)*, 1991.

[12] R.P. Paul. Model based teleoperation to eliminate feedback delay. Technical report, Computer and Information Science Department, University of Pennsylvania, 1989.

[13] R.P. Paul and G. Giralt. An autonomous sensor-controlled manipulation capability for an unmanned, untethered, submersible. In *International Symposium on Unmanned Untethered Robotics, New Hampshire (USA)*, 1988.

[14] T. Sheridan. Telerobotics. In *IEEE International Conference on Robotics and Automation, Scottsdale, (USA)*, 1989.

9-8

J.4 Planification de Tàches

10 *Automatic planning of pick and place operations in presence of uncertainties.*

IEEE International Workshop on Intelligent Robots and Systems (*IROS'90*), Tsuchiura (Japan), Juillet 1990.

Auteurs: I. Mazon, R. Alami, P. Violéro.

11 *Two manipulation planning algorithms*

The First Worshop on the Algorithmic Foundations of Robotics, A.K. Peters Pub., Boston, MA. 1994.

Auteurs: R. Alami, J.P. Laumond, T. Siméon

12 *Planning robust motion strategies for a mobile robot in a polygonal world.*

Revue d'Intelligence Artificielle, Vol 8, Nr 4/1994, pp 383-401, HERMES, 1994.

Auteurs: T. Siméon, R. Alami

IEEE International Workshop on Intelligent Robots and Systems
IROS '90

Automatic Planning of Pick and Place Operations in presence of uncertainties

I. Mazon, R. Alami and P. Violéro,
Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S)
7, Avenue du Colonel Roche
31077 Toulouse Cedex (France)

Abstract

This paper describes an implemented system that plans automatically "Pick and Place" tasks. The originality of this system comes from the way it takes into consideration the strong interdependencies existing between the different steps of a "Pick and Place" task.

Indeed, it deals explicitly with the interactions between grasp planning, transfer motions and approach motions. Besides, it makes use, at several steps during the planning phase, of a module that propagates and evaluates position uncertainties in a "Pick and Place" context.

We describe the different modules included in our system and show how the controller invokes them and how it manages their interaction. Finally, we discuss several control strategies which can be implemented in order to solve a "Pick and Place" planning problem.

1 Introduction

One key issue in robotics is the development of automatic Task Level Planning systems. When instanciated to "Pick and Place" tasks, this problem raises difficult issues that are only partially taken into account by existing or proposed systems.

A now classical approach to plan a "Pick and Place" task is to decompose it into several steps such as transfer motion, approach motion, grasp, depart motion etc ... The problem is that there are strong interdependencies between these steps.

These interdependencies can be classified into geometric interdependencies (e.g. the grasp position should be chosen not only to ensure a stable grasp but also taking into account the reachability of the grasp position, the way the object will be manipulated in the next action) and interdependencies related to position uncertainty (the choice of a grasp position depends on the relative position uncertainty between the robot and the object before the action is performed, but also on the action to be performed after the grasp action is completed).

This paper describes a system called "SPARA" (Système de Programmation Automatique pour la Robotique d'Assemblage). Given a description of the environment (manipulator, objects, obstacles) the system is able to produce automatically the sequence of effector level actions that are necessary to perform a Pick and Place task expressed in terms of initial and final positions of the robot and of an object. Besides, the obtained plan is robust with respect to position uncertainties.

In section 2, we briefly discuss related work. In section 3, we present the architecture of the SPARA system. Section 4 gives a brief description of the different modules involved in the planning process.

In section 5, we discuss different control strategies that can be used in order to solve the Pick and Place planning problem by invoking the different modules and managing their interactions.

2 Related Work

We will not discuss here all issues related to robot programming; the interested reader may refer for example to [4].

Concerning recent Task Level systems (in the context of Pick and Place tasks), to the best of our knowledge, the only systems that effectively address the problem in a complete way are HANDEY [5], [7] and SHARP [3], [13].

HANDEY system has the major advantage to have been implemented and tested in a real environment. One interesting feature is its ability to plan a regrasp action when it finds no way to perform the task with only one pick and one place operation.

In the SHARP system, the uncertainty constraints are taken into account in a verification/correction phase that takes place after the planning phase [11]. In case of failure, the plan is locally improved by adding a sensing action.

HANDEY and SHARP make the assumption that the task environment is not too constrained. Since the solution space is assumed to be sufficiently large, it is reasonable to make approximations without discarding all possible solutions.

Our system is based on the same assumption and performs a similar task decomposition. However it is designed to handle environments that might be a little more constrained, due to the following:

- a local motion planner which builds an exact representation of the obstacles in (x, y, θ) configuration space,
- an original method for propagating geometric constraints between gross motions and local motions,
- the processing of uncertainty constraints in the planning phase.

3 Architecture of SPARA

SPARA is based on a decomposition of a "Pick and Place" task into several steps:

1. a transfer motion (also referred to as gross motion) from the initial robot position to a point "near" the object to be picked,
2. an approach motion (in the vicinity of the obstacles) in order to reach the grasp position,

10.1

3. the grasp action itself,
4. a depart motion from the grasp position to a position where a transfer motion can be performed,
5. a transfer motion in order to bring the object "near" its final destination,
6. an approach motion in order to place the object,
7. an ungrasp action,
8. a depart motion from the place position to a position where a transfer motion can be performed,
9. a transfer motion in order to bring the robot to its final position.

In order to solve these problems, SPARA is mainly composed of a high-level module (referred to as the "controller") and a number of "specialized planners" : a gross motion planner, a local motion planner, a grasp planner and an uncertainty propagation module.

We discuss in this section the interdependencies that are explicitly taken into account by SPARA. Indeed, these interdependencies lead to a decomposition of the different specialized planners into several modules that can be called independently by the controller with different data. In the next section, we will show how this decomposition allows a great flexibility in the choice of various control strategies for planning the complete task.

SPARA deals with interdependencies caused by geometric interactions and by position uncertainties constraints.

SPARA propagates position uncertainty, step by step, through the sequence of planned actions and verifies at each step that it does not violate the constraints imposed by a given action. If it is not so, the system backtracks. Another way to take uncertainties into account is to use them in order to guide the planning process, for example by ordering the possible grasps with respect to the position uncertainty they will entail.

Geometric interactions can be classified into three types:

- interactions induced by the task itself: for example, the choice of a grasp must be compatible with the environment of the subsequent place action,
- interactions induced by the decomposition of the task into several steps: for example, the initial point of an approach motion must be reachable through a transfer motion,
- interactions induced by the limitations of the different specialized planners used by SPARA: for example, local motion planning is mainly based on the use of a (x, y, θ) motion planner that produces a trajectory for a projection of the gripper in a plane. However, such a trajectory must verify the following constraints: 1) it must not violate the mechanical limits of the manipulator, 2) it must not pass through a singularity, and 3) it must be collision-free for the whole robot.

An example of a task performed by the system is shown in figure 1. It consists of picking object B and placing it on a specified final position. The illustration of results produced by the different specialized planners are based on the same example.

4 SPARA basic modules

SPARA is composed of a number of specialized planners and a set of processing modules. The specialized planners are themselves decomposed into modules which can be invoked independently by the controller.

We give here below a brief description of SPARA basic modules.

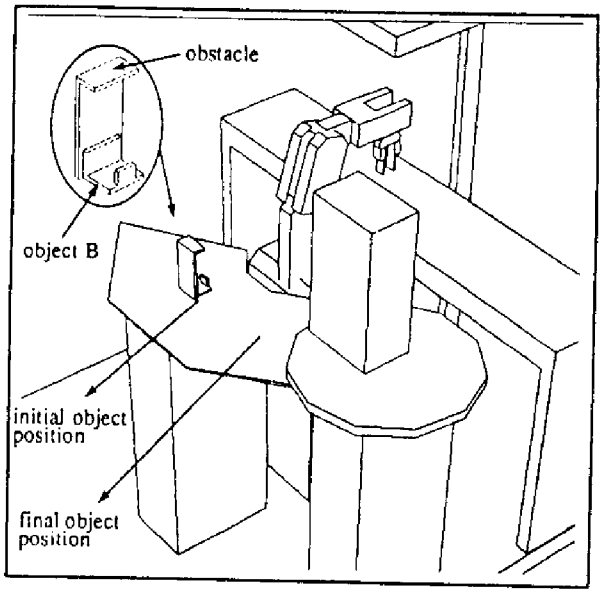


Figure 1: An illustrative example of a Pick and Place task

The interested reader may refer to [8] [14] [9] for a more detailed presentation.

Before presenting the different modules, let us first summarize the basic assumptions upon which our system is built :

- all objects and obstacles in the environment are represented by polyhedra
- the grasping tool is a parallel jaw gripper
- a grasp is performed via two plane-to-plane contacts
- the robot has an arm-wrist kinematic structure
- the approach and depart motions for picking and placing objects are carried out in a plane.

4.1 The Gross Motion Planner

The algorithms used in this planner [12] belong to the class of global methods based on a discretization of the robot configuration space. It is composed of two modules:

1. a module which builds a representation of the free space for an approximation of the actual manipulator: a manipulator with only three degrees of freedom corresponding to the first three joints; the volume swept by the last three links for the full range of values of the last 3 joints is represented by a sphere whose radius is the greatest distance between the center of the wrist and any point of the gripper. The representation of the free space is a connectivity graph. It is noted GFS_r when it is computed with an empty gripper, and GFS_{r+o} with the gripper holding an object.
2. a module which builds a transfer path between two configurations by performing a search in the connectivity graph. An example of such a trajectory is given in figure 2.

4.2 The Grasp Planner

This module [14] generates automatically all potential grasps (with planar contacts) for a polyhedral object and a parallel jaw gripper; it uses techniques similar to [10].

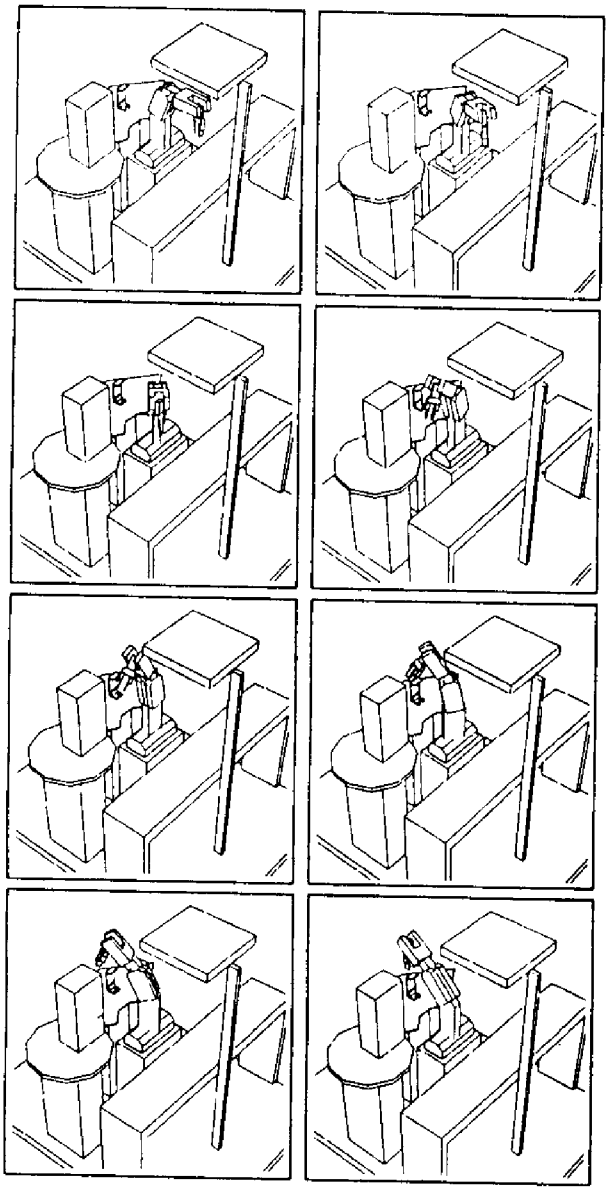


Figure 2: A transfer motion (read left to right, top to bottom)

This set of grasps is then refined in order to eliminate grasps which induce a collision between the gripper and the environment at pick and/or at the place locations.

A potential grasp is defined by two parallel object faces. We call "grasp plane" their median plane. The intersection of the projections of theses faces on the grasp plane is called the "visibility polygon".

As the final location of the object is known, we define, in a similar manner, a "place plane" which corresponds to the grasp plane at the object target location.

Note that a potential grasp does not completely determine the position of the gripper relatively to the object. The position and orientation of the gripper is only constrained to verify an overlap between the projection of the jaws in the grasp plane and the visibility polygon.

Several classification criterias can be chosen (by the controller) in order to select first the most "promising" grasp. We can use the uncertainty that the grasp action will induce after its the execution, or the stability of the grasp...

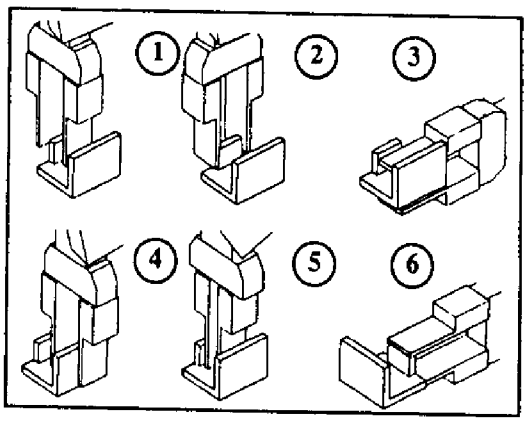


Figure 3: Six potential grasps

When the controller calls this module, it can ask to produce grasps that are compatible with the pick location only, or with the pick and place locations together. In the example of figure 3, only three grasps (numbered 1,2 and 5) out of six are considered as correct.

4.3 The Local Motion Planner

This planner solves the problem of path planning for a polygonal object in translation and rotation in a plane among polygonal obstacles [1], [14]. It is used in order to build trajectories for a projection of the gripper in a plane. In order to guarantee that the robot will be able to "follow" such a trajectory, we impose to the robot to stay in the same posture (geometric configuration) during all the local motion. This will be discussed in section 4.6

It is composed of two modules:

1. a module which builds an exact representation of the boundary of local free space.
2. a module which uses the obtained representation in order to compute a free collision trajectory between two (x, y, θ) configurations.

In our context, the algorithm has been modified in order to deal with a partially specified final configuration. The goal configuration is given with an arbitrary orientation. The motion is stopped when the current position verifies a given criteria. This feature has been implemented in order to search for a Pick approach motion, where the grasp is not totally specified.

An example of a pick depart motion is given in figure 4.

4.4 The position uncertainty propagation module

No programming system will produce robust programs if it does not take into account the fact that the location of objects and

10.3

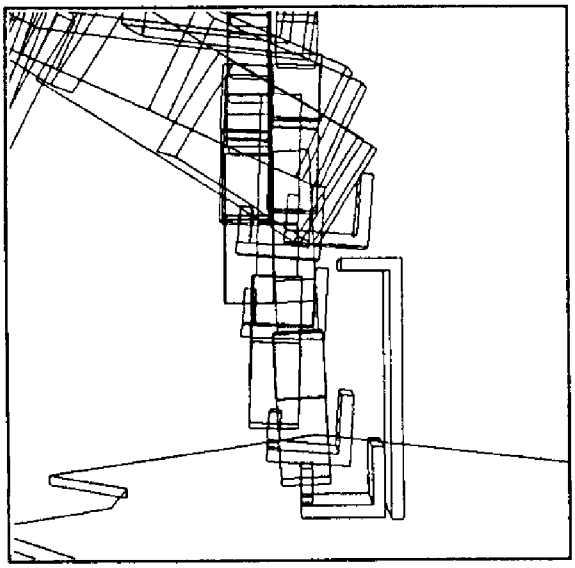


Figure 4: Local Pick depart motion (read from bottom to top)

of the robot itself are subject to error. Moreover, as the robot physically acts on objects, the position uncertainty associated with the objects may change and propagate from one situation to another.

One solution to the problem is to make use of compliant motion and sensing. However, such an approach is not sufficient. Uncertainty must also be taken into account from a global point of view, in order to estimate the position uncertainties at each step of a robot program. This is the aim of this module. It embeds a set of operators that allow to propagate uncertainties through a sequence of actions [9].

This module, is invoked each time the controller instanciates a new action, in such a way that the system maintains a plan of actions which is always robust.

This module can also be invoked in order to evaluate the quality of a potential grasp. As a potential grasp does not completely define the position of the gripper relatively to the object, the quality of the grasp is estimated by computing the maximum uncertainty that may occur [14].

4.5 The Workspace module

This module computes the set of positions that can be reached by a point attached to a manipulator and whose position depends only on the first 3 degrees of freedom of the robot. It is based on the notion of joint space decomposition into "aspects" as defined in [2].

This module can be called in two different ways. The first one consists in computing the images of the complete configuration space in the cartesian space. The second way consists in computing only the images of the free space (produced by the Gross motion planner) in the cartesian space.

A different image will be obtained for each aspect. They will be noted $WS^1, .., WS^4$ and $FWS^1_r, .., FWS^4_r$.

4.6 The Accessibility module

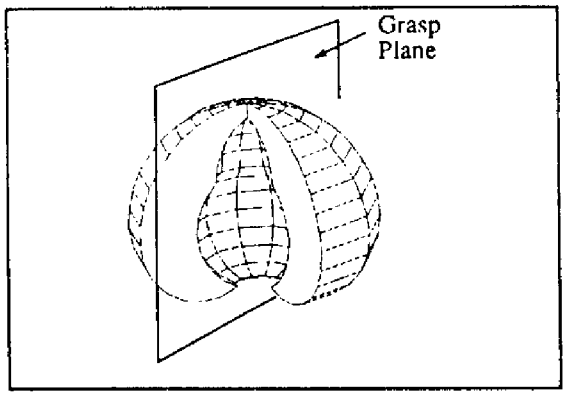


Figure 5: Workspace for the center of the wrist

This module computes a surface corresponding to all points in a plane which can be reached by a point attached to the manipulator wrist. This is done by cutting a representation of the workspace, for a given aspect, by a given plane.

Such computation is performed in order to determine:

1. a surface (noted B) representing all points (in the grasp plane or in the place plane) which can be accessed by the origin of a frame attached to the gripper.
2. or a surface (noted B') representing all points (in the grasp plane or in the place plane) which can be accessed by the origin of a frame attached to the gripper and which correspond to free configurations (as determined by the Gross Motion planner).

Surfaces of type B correspond to all points which can be potentially accessed by the local motion planner and which are compatible with the kinematic structure of the manipulator.

Surfaces of type B' represent the set of points which can be accessed by a gross motion and by a local motion. Figure 5 represents such surfaces.

It is worth noting that the same aspect must be used for an approach and a depart motion in order to guarantee that there is no posture change between the two motions. Consequently, a same surface B will be used for both motions, while a new surface B' must be computed after a grasp or an ungrasp operation.

4.7 The Local Environment Module

This module is in charge of computing the local (planar) environment (polygonal mobile body, and polygonal obstacles) that will be used by the local motion planner in a Pick and/or in a Place context.

The polygonal mobile body is the projection of the gripper (or the gripper and the grasped object) on the grasp (or the place) plane.

In order to determine the polygonal obstacles, the module uses surface B to build a prism (figure 6) which corresponds to the volume swept by the gripper (or the gripper and the grasped object).

The polygonal obstacles correspond to the projection of the parts of the environment obstacles which intersect the prism.

10.4

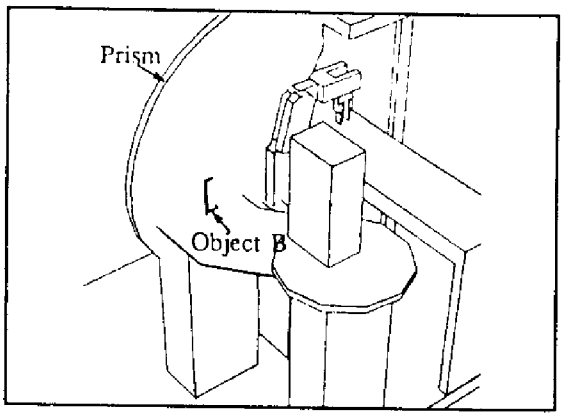


Figure 6: Local environment

The exterior of surface B is also considered as an obstacle in order to guarantee that any path produced by the local motion planner (for the gripper in the plane) will respect the robot kinematics constraints (for the first 3 d.o.f).

This module can also be invoked by the controller in order to "merge" in a same local environment the constraints imposed by the Pick and by the Place situations. Doing so, it becomes possible to plan a local motion which can be valid (locally):

- as a Pick approach motion and a Place depart motion
- or a Pick depart motion and a Place approach motion

4.8 Extremity Points for linking local and transfer motions

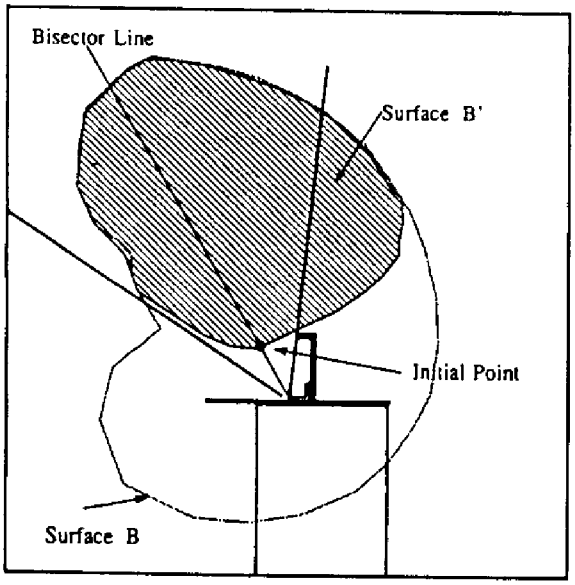


Figure 7: Choice of an initial point for a Pick approach motion

This module is in charge of selecting a set of points which can be accessed by a planar local motion and a transfer motion in a given context (Pick or Place, gripper empty or holding an object).

Such points belong to the frontier of surfaces of type B'. The selection of good candidate points is based (heuristically) on the visibility between such points and a path extremity (provided by the controller) among polygonal obstacles.

Figure 7 illustrates the computation done in order to find the initial points of an approach motion for the grasp action.

4.9 Verification

We describe here a set of verifications that must be performed in order to guarantee that a local motion is valid:

- there must be no collision for the whole robot
- there must be no posture change (on the last three degrees of freedom) along the trajectory

Besides, when a grasp transformation is completely determined, it is possible to propagate position uncertainties and to verify that the grasp action and the subsequent place action are robust with respect to uncertainties.

5 Controlling the planning process

The planning process is implemented in the controller. Two types of interactions are used:

1. the first type is a classical generate-and-test interaction. For example, the Grasp Planner produces a finite set of potentials grasps. This set can then be sorted heuristically and its elements are then used one after the other until a complete plan is produced;
2. the second type can be qualified as constraint propagation. An example of constraint propagation is the use of the free-space (computed by the gross motion planner) in order to compute the initial point of a local motion

5.1 A planning process

We describe the control flow of a complete planning process in order to show how the controller can be programmed to take into account the interdependencies existing between the different steps of a Pick and Place task. A step corresponds to a controller choice or to the call of one (or more) modules in a given context.

Figure 8 illustrates the different planning steps.
Similar steps are represented by the same pattern.
Bold patterns represent "backtrack" steps. A backtrack step generates a finite set of possible choices and produces a new choice each time it is executed until no choice is available. In such case, a backtrack to a preceding backtrack step is necessary.
Simple arrows indicate precedence between steps. Bold arrows represent a backtrack to a preceding backtrack step.

The presented planning process involves the following steps:

step 1: Building a representation (noted GFS_r) of the free-space for the robot alone (3 d.o.f, see §4.1.1).

step 2: Building a representation of the workspace for the center of the wrist for each robot aspect: $WS^1, .., WS^4$ (see §4.5).

step 3: Building, for each aspect, a representation of the part of the workspace that corresponds to the image GFS_r in the cartesian space: $FWS_r^1, .., FWS_r^4$ (see §4.5).

10.5

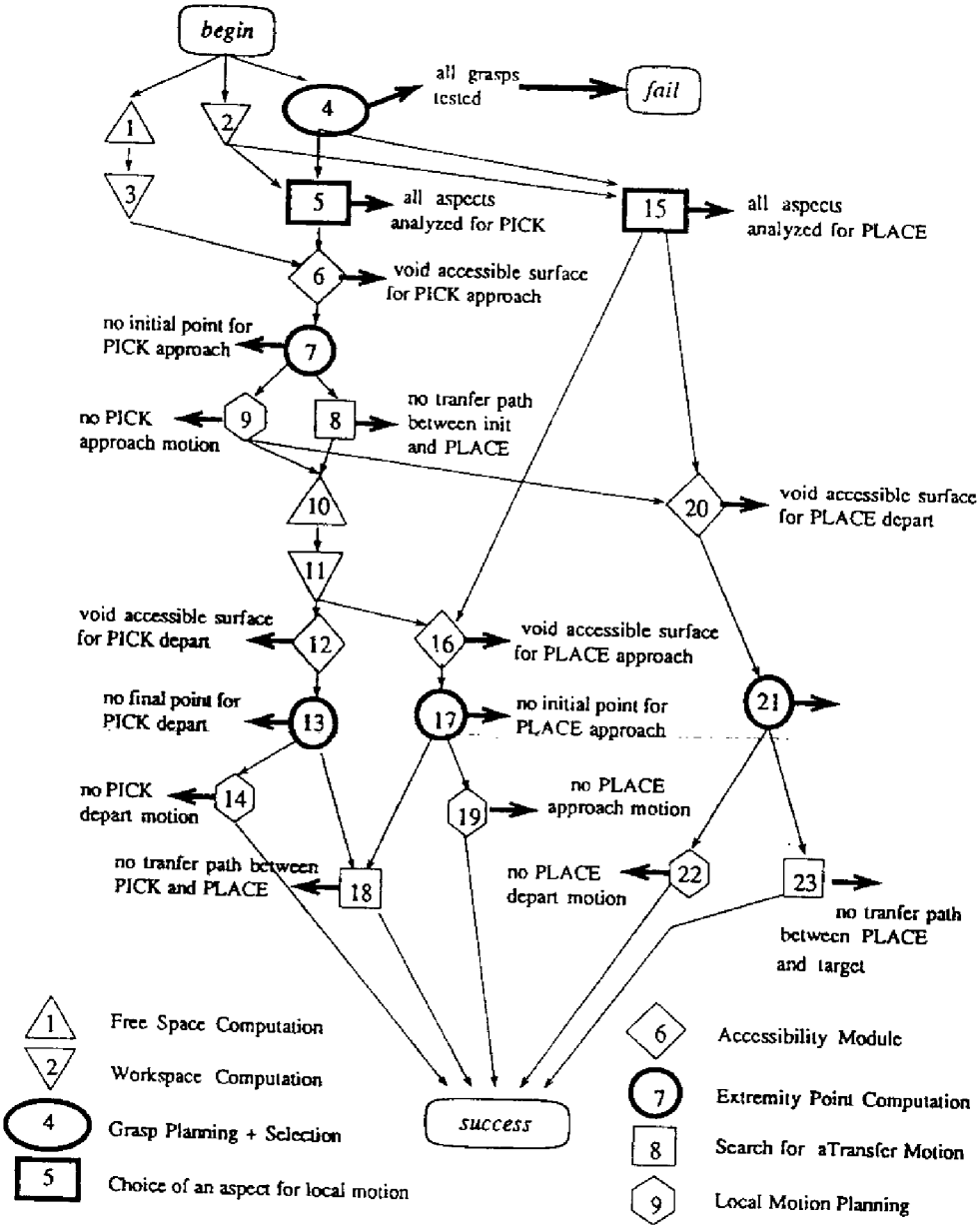


Figure 8: The different planning steps

step 4: Generation and selection of a potential grasp (see §4.2); the *grasp plane* and the *place plane* are induced by this choice.

step 5: Selection of a robot aspect (noted i) for planning the approach and depart motions of the PICK operation.

step 6: Computing surfaces B and B' using WS^i, FWS^i_r and the selected grasp plane (see §4.6). Testing that the visibility polygon intersects surface B.
Computing the (planar) environment (obstacles and mobile) for planning the approach motion for the PICK operation (see §4.7).

step 7: Determining an initial point (noted APP_{pick}) for the approach motion (see §4.8).

step 8: Search (in GFS_r) for a transfer motion between the robot initial position and APP_{pick} (see §4.1.2).

step 9: Building a representation of the local free space (see §4.3) and search for a local approach motion starting at APP_{pick}. If the search is successful then the grasp transformation is completely determined (see §4.3).
The obtained approach motion is then checked (see §4.9) in order to verify that it causes no-collision for the whole robot, no posture change...

10.6

step 10: Building a representation (noted GFS_{r+o}) of the free-space for the robot holding the object (§4.1.1).

step 11: Building, for each aspect, a representation of the part of the workspace that corresponds to the image GFS_{r+o} in the cartesian space: $FWS^1_{r+o}, ..., FWS^4_{r+o}$ (see §4.5).

step 12: Computing a new surface B' using FWS^i_{r+o} and the grasp plane (see §4.6). Computing the (planar) environment for planning the PICK depart motion (see §4.7). In this step, the mobile is the projection of the gripper and of the grasped object.

step 13: Determining a final point (noted DEP_{pick}) for the PICK depart motion (see §4.8).

step 14: Building a representation of the local free space and search for a local depart motion between the grasp position and APP_{pick} (see §4.3).
The obtained depart motion is then checked (see §4.9) in order to verify that it causes no-collision for the whole robot, no posture change...

step 15: Selection of a robot aspect (noted j) for planning the approach and depart motions of the PLACE operation.

step 16: Computing surfaces B and B' using WS^j, FWS^j_{r+o} and the selected place plane (see §4.6).
Computing the (planar) environment (obstacles and mobile) for planning the approach motion for the PLACE operation (see §4.7).

step 17: Determining an initial point (noted APP_{place}) for the PLACE approach motion (see §4.8).

step 18: Search (in GFS_{r+o}) for a transfer motion between DEP_{pick} and APP_{place} (see §4.1.2).

step 19: Building a representation of the local free space and search for a local approach motion starting between APP_{place} and the place position (see §4.3).
The obtained approach motion is then checked (see §4.9) in order to verify that it causes no-collision for the whole robot, no posture change...

step 20: Computing a new surface B' using FWS^j_r and the place plane (see §4.6). Computing the (planar) environment for planning the PLACE depart motion (see §4.7).

step 21: Determining a final point (noted DEP_{place}) for the PLACE depart motion (see §4.8). **step 22:** Building a representation of the local free space and search for a local depart motion between the place position and DEP_{place} (see §4.3).
The obtained depart motion is then checked (see §4.9) in order to verify that it causes no-collision for the whole robot, no posture change...

step 23: Search (in GFS_r) for a transfer motion between DEP_{place} and the robot target position (see §4.1.2).

5.2 Control strategies

The aim of the following remarks is to illustrate how the controller manages the interactions between SPARA modules, and to show the great variety of possible control strategies which can be used.

1. When a backtrack is necessary, the easiest way to implement an exhaustive search is to go back to the nearest upper step. However, this is not mandatory and more sophisticated searches can (should ?) be devised.
2. Several "paths" are possible. For example, after step 17, the controller can execute steps 18 and 19 in an arbitrary order. A possible strategy is to begin with the step that is assumed to be the most constrained. Another strategy is to begin with the less "expensive" step.
3. Gross motion planning is performed by steps 1 and 8 for the first transfer motion, by steps 10 and 18 for the second, and by steps 1 and 23 for the last. Besides, the output of step 1 (resp. 10) is also used in steps 3, 6 and 20 (resp. 11, 12 and 16).
4. The first "half" of the planning process (from step 1 to step 9) is almost "sequential" while the second half exhibits more "parallelism". This is due to the fact that the grasp planner (step 4) produces uncompletely instanciated grasps. The grasp transformation is completely determined only at step 9. This strategy gives more latitude to the first local motion planning step (which is considered as a very constrained step).
   Another possibility is to provide a grasp planner which produces completely instanciated grasps. In such case, the planning process can become more "parallel" after step 4; however this may entail a greater number of backtracks to step 4.
5. A possible heuristic is to "merge", for example, the sequences of steps 5-6-7 and 15-20-21. This corresponds to the "projection" in a same environment of the obstacles at the Pick and at the Place location. Such a merging can be very efficient because it allows to find, in one sequence, a grasp and a local path which can be used as a local Pick approach motion and as a local Place depart motion. However, the obtained environment may also be too constrained to allow such a motion.
6. As mentioned before, several steps can be executed in parallel. The implementation of SPARA has been designed in order to allow the controller to run the motion planners as independent processes in a UNIX environment.

6 Conclusion

A first version of the system is almost completely implemented. It includes the gross motion planner, the local motion planner, the grasp planner and the module which maintains and propagates the position uncertainties. However, the computation of the projection of the free space expressed in the configuration space, on the cartesian space is under implementation and is not yet integrated.

All the figures presented in the paper, have been obtained automatically, except the surface B' in figure 7.

The principal limitations of the system presented in this paper, come from the characteristics of local motions (planar motions in the grasp plane) and from the algorithms used to plan these motions

One major advantage of the system is the flexibility it provides for programming different control strategies.

More work remains to be done in order to improve the system. Extensions and future work will concern the following aspects: the integration of the complete system, the development of a local motion planner which deals with a 3D object moving in a plane, and the investigation of different control strategies.

Acknowledgements: This system is a result of a continuous team effort. Alain Giraud, Bruno Gorla, Thierry Siméon and Michel Taix have contributed to its development and implementation. We are also grateful to Jean Paul Laumond, Philippe Juhel, Philippe Moutarlier and to many other members of the team. Several figures presented in the paper have been drawn

using EUSLISP 3D functions [6].

I. Mazon is partially supported by MATRA-DATAVISION.

References

[1] F. Avnaim and J-D. Boissonnat. A practical exact motion planning algorithm for polygonal objects admist polygonal obstacles. In *IEEE International Conference on Robotics and Automation, Philadelphia (USA)*, 1988.

[2] P. Borrel and A. Liegeois. A study of multiple manipulator inverse kinematic solutions with applications to trajectory planning and workspace determination. *IEEE International Conference on Robotics and Automation, San Francisco (USA)*, 2:1180–1185, April 1986.

[3] C. Laugier and J. Troccaz. SHARP, a system for automatic programming of manipulation robots. In *Robotics Research : The Third International Symposium, O. Faugeras and G. Giralt (Eds), MIT Press, Cambridge, Massachusetts*, October 1985.

[4] T. Lozano-Perez. Robot programming. A.I Memo 698, Artificial Intelligence Laboratory, MIT, December 1982.

[5] T. Lozano-Perez, J.L. Jones, E. Mazer, and Al. Handey: A robot system that recognizes, plans and manipulates. In *IEEE International Conference on Robotics and Automation, Raleigh (USA)*, April 1987.

[6] T. Matsui. An object based robot programming system : EUSLISP. Technical report, Intelligent Systems Division, Electrotechnical Laboratory, Tsukuba-city (Japan), September 1989.

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

[8] I. Mazon. Raisonnement sur les contraintes géométriques et d'incertitudes pour la planification de tâches de manipulation robotisées. Thèse de l'Université Paul Sabatier, Toulouse (France), Laboratoire d'Automatique et d'Analyse des Systèmes (C.N.R.S.), May 1990.

[9] I. Mazon and R. Alami. Representation and propagation of positionning uncertainties through manipulation robot programs. integration into a task-level programming system. In *IEEE International Conference on Robotics and Automation, Scottsdale, Arizona (USA)*, May 1989.

[10] J. Pertin-Troccaz. On-line automatic robot programming : a case study in grasping. In *IEEE International Conference on Robotics and Automation, Raleigh (USA)*, pages 1292–1297, April 1987.

[11] P. Puget. *Vérification-Correction de programme pour la prise en compte des incertitudes en programmation automatique des robots*. PhD thesis, Institut National Polytechnique de Grenoble, February 1989.

[12] T. Simeon. Planning collision free trajectories by a configuration space approach. In *Geometry and Robotics, J.D. Boissonnat and J.P. Laumond (Eds)*. Lecture Notes in Computer Science, Vol 391, Springer Verlag, 1989.

[13] P. Thevenau. Utilisation du raisonnement géométrique pour la planification en robotique d'assemblage: Le Systeme PAMELA. Thèse de doctorat de l'Institut National Polytechnique de Grenoble, L.I.F.I.A., Grenoble (France), November 1988.

[14] P. Violero, I. Mazon, and M. Taix. Automatic planning of a grasp for a pick and place action. In *IEEE International Conference on Robotics and Automation, Cincinnati, Ohio (USA)*, May 1990.

Two manipulation planning algorithms

R. Alami, J.P. Laumond, and T Siméon
LAAS / CNRS, 7, Avenue du Colonel Roche, 31077 Toulouse, France

This paper addresses the motion planning problem for a robot in presence of movable objects. Motion planning in this context appears as a constrained instance of the coordinated motion planning problem for multiple movable bodies.

Indeed, a solution path (in the configuration space of the robot and all movable objects) is a sequence of transit-paths, where the robot moves alone, and transfer-paths where a movable object "follows" the robot. A major problem is to find the set of configurations where the robot has to "grasp" or "release" objects.

Based on [1, 5], the paper gives an overview of a general approach which consists in building a manipulation graph *whose connected components characterize the existence of solutions. Two planners developed at LAAS/CNRS illustrate how the general formulation can be instantiated in specific cases.*

1 The manipulation planning problem

Robot motion planning usually consists in planning collision-free paths for robots moving amidst fixed obstacles. Nevertheless a robot may have to perform tasks which are more difficult than planning motions only for itself. In some situations, a robot may be able to move objects and to change the structure of its environment. In such a context, the robot moves amidst obstacles but also movable objects. A movable object cannot move by itself; it can move only if it is grasped by the robot. According to the standard terminology, considering movable objects appears as a constrained instance of the *coordinated motion planning problem*, that we call the *manipulation planning problem* [1].

[1] Note that this problem is related to Pick&Place and re-Grasping tasks and not to the dextrous manipulation of an object by a multi-fingered robot hand.

As stated in [1] (see also Latombe's book [10]), a general geometric formulation of the problem can be defined as follows.

1.1 Manipulation task and configuration space(s)

The environment is a 3D (resp. 2D) workspace which consists of three types of bodies: (1) static obstacles, (2) movable objects and (3) a robot.

For the robot and for each object we consider its associated configuration space. Object configuration spaces are 6D (resp. 3D); the robot configuration space is n-dimensional, where n is its number of degrees of freedom. Let CS denote the cartesian product of all objects and robot configuration spaces.

In the following, we will say that c is an incompletely specified configuration when some parameters in c are left unspecified; besides, we will say that a configuration c' "verifies" c when it is included in the subspace defined by c. Such a terminology will be used in order to denote partially specified goals.

Furthermore, we introduce a function Free which gives, for each domain of CS, the set of its free configurations (i.e. configurations where the bodies do not overlap).

A manipulation task is clearly a particular path in Free(CS). The converse does not hold: all paths in Free(CS) do not necessarily correspond to a manipulation task. Indeed a manipulation path is a constrained path in Free(CS). We have now to define geometrically these constraints. There are two types of constraints:

- constraints on the placements of objects; these constraints model the physics of the manipulation context (any object must be in a stable position in the environment),

- constraints on object motions; any object motion is a motion induced by a robot motion.

1.2 Placement constraints

All configurations in Free(CS) do not necessarily correspond to a physically valid environment configuration. For example an object can not "levitate", and must be in a stable position. Geometrically speaking, we have to reduce the space of free configurations to a subspace which contains all valid configurations. These constraints concern only the objects. For example, if we constrain a polyhedron to be placed only on top of horizontal faces of polyhedral obstacles or of other objects (which are already in a stable position); its placement constraints will then define a finite number of 3-dimensional manifolds in its configuration space.

We call $PLACEMENT$ the subspace of Free(CS) containing all valid placements for all objects, i.e. placements which respect the physical constraints of the manipulation context. With this definition, all the objects have a fixed and known geometrical relations with the obstacles or with other objects.

$PLACEMENT$ is not more precisely defined; the definition depends on the context, and appears clearly for each context. For a mobile robot in a 2-dimensional euclidean space, amidst movable objects, $PLACEMENT$ = Free(CS).

For the planner presented in Section 2, we assume that each object has a finite number of placements in the environment; then $PLACEMENT$ appears as a finite union of n-dimensional manifolds, where n is the number of degrees of freedom of the robot.

For the planner presented in Section 3, the movable object can be placed anywhere in the environment.

1.3 Motion constraints

We define a grasp mapping G_O^{T} as a mapping from the configuration space of the robot (noted CSR) into the configuration space of a given object O (noted CSO), which verifies $G_O^{\mathsf{T}}(cr) = co$, where $cr \in CSR$ and $co \in CSO$. This mapping models the geometrical relation which is defined by a grasping operation (T denotes a homogeneous transform between the robot gripper frame and the object reference frame). Such mappings define geometrically the semantics of grasping for a particular manipulation context. They can be in finite or infinite number, and can be given explicitly (as in Section 2) or implicitly (they are defined for example by a contact relation between the robot or the object as in Section 3).

1.4 Problem statement

Definition A *transfer-path* is a path in Free(CS) such that there is one object O and one grasp mapping G_O^{T} verifying:

- the configuration parameters of any object $O' \neq O$ are constant along the path
- for any configuration of the path, $G_O^{\mathsf{T}}(cr) = co$, where cr and co designate respectively the configuration parameters of the robot and O.

Two configurations of Free(CS) connected by a transfer-path are said to be *g-connected*.

We call $GRASP$ the subspace of Free(CS) containing the configurations which are *g-connected* with a configuration of $PLACEMENT$.

Definition: A *transit-path* is a path in Free(CS) such that the configuration parameters of the objects are constant along the path. Two configurations in Free(CS) connected by a transit-path are said to be *t-connected*.

Remark: a transit-path is included in $PLACEMENT$ (but every path in $PLACEMENT$ is not necessary a transit-path).

We are now in position for defining any manipulation task as a manipulation path in Free(CS):

Definition: A *manipulation-path* is a path in Free(CS) which is a finite sequence of transit-paths and transfer-paths. Two configurations in Free(CS) connected by a manipulation-path are said to be *m-connected*.

A manipulation planning problem can then be defined as:

> **Manipulation planning problem:** An initial configuration i and a final (completely or incompletely specified) configuration f being given, does there exists a configuration verifying f which is *m-connected* with i ? If the answer is yes, give a *manipulation path* between i and some configuration verifying f.

1.5 Manipulation graph

The previous definitions lead to a property which models the structure of the solution space:

Lemma: A transit-path and a transfer-path are connected iff both have a common extremity in $PLACEMENT \cap GRASP$.

The manipulation planning problem then appears as a constrained path finding problem inside the various connected components of $PLACEMENT \cap GRASP$ and between them.

In the case of a discrete number of placements and grasps, $PLACEMENT \cap GRASP$ consists of a finite set of configurations.

When the environment contains only one movable object, even if there is an infinite number of placements and grasps, we can prove that two configurations which are in a same connected component of $GRASP \cap PLACEMENT$ are *m-connected* (see Appendix). This property leads to reduce the problem.

In both cases, it is sufficient to study the connectivity of the various connected components of $GRASP \cap PLACEMENT$ by transit-paths and transfer-paths.

We then define a graph whose nodes are the connected components of $GRASP \cap PLACEMENT$. There are two types of edges. A *transit* (resp. *transfer*) edge between two nodes indicates that there exists a transit-path (resp. transfer-path) path linking two configurations of the associated connected components.

This graph is called a *manipulation graph (MG)*. It verifies the fundamental property:

Property: An initial configuration i and a goal (completely or incompletely specified) configuration g being given, there exists a configuration f verifying g and m-connected with i iff:

- there exist a node N_i in MG and a configuration c_i in the associated connected component of $GRASP \cap PLACEMENT$, such that i and c_i are t-connected or g-connected;
- there exist a node N_f in MG and a configuration c_f in the associated connected component of $GRASP \cap PLACEMENT$ such that:
  - c_f and f are t-connected or g-connected;
  - N_i and N_f are in the same connected component of MG

In order to use this method for particular instances of the problem, one needs:

1. to compute the connected components of $GRASP \cap PLACEMENT$;
2. to determine the connectivity of these connected components using transit-paths and transfer-paths;
3. and to provide a method for planning a path in a given connected component of $GRASP \cap PLACEMENT$.

We present, in the sequel, two manipulation planners working respectively when $PLACEMENT \cap GRASP$ is reduced to a finite set of points (Section 2) and when the environment contains only one movable object (Section 3).

2 The case of discrete placements and grasps for several movable objects

In this section, we present a description of a manipulation task planner for the case of discrete placements and grasps for objects[2]. It is directly derived from the general scheme above. It is based on the fact that the connected components of $GRASP \cap PLACEMENT$ are given *a priori* by some discretization and that the construction of transit-paths and transfer-paths can be obtained using a collision-free path planner for a robot amidst stationary obstacles.

It leads to an effective construction of the manipulation graph.

For simplicity reasons, we give a presentation considering only two objects. The extension to a finite number of objects is straightforward. The presentation will be illustrated using the example of Figure 1, i.e. a 2D world where all bodies are polygonal and where the robot is allowed to move only in translation. However, the solution we propose is general.

2.1 Notations

We designate the robot by R and the objects by A and B. Let cr, ca and cb be the configuration pa-

[2]This planner has been first introduced in [1]. In this current presentation we have added new experimental results.

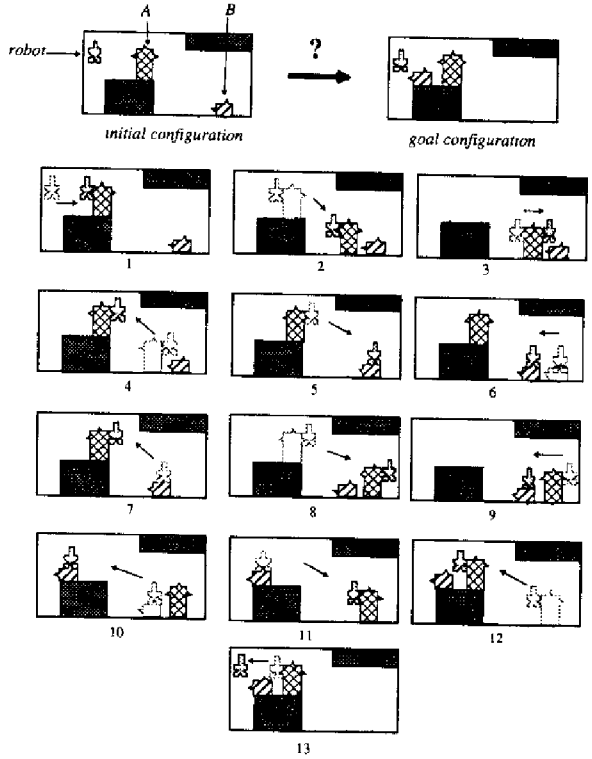


Figure 1: *A manipulation task generated automatically*

rameter vectors and CSR, CSA and CSB the associated configuration spaces. Let n be the dimension of CSR. The configuration space of all movable bodies is: $CS = CSR \times CSA \times CSB$.

Object placements: We assume that each object has a finite number of possible placements in the environment that will physically correspond to stable positions when the robot does not hold the object.

We designate by $p_A^1, \ldots p_A^i \ldots \in CSA$ and by $p_B^1, \ldots p_B^j \ldots \in CSB$ the authorized placements for A and B respectively (Figure 2).

Remark: we may also give explicitly - or give means to compute - all authorized placements combinations $(p_A^i, p_B^i) \in CSA \times CSB$, in order to take into account, for example, the possibility of stacking an object on another object.

Object grasps: We assume that each object has a finite number of possible grasps. A given grasp for object A is specified by providing a mapping $G_A^i : CSR \longrightarrow CSA$.

Let G_A^1, G_A^2 G_A^3 and G_B^1, G_B^2 be the authorized grasps for A and B (Figure 2).

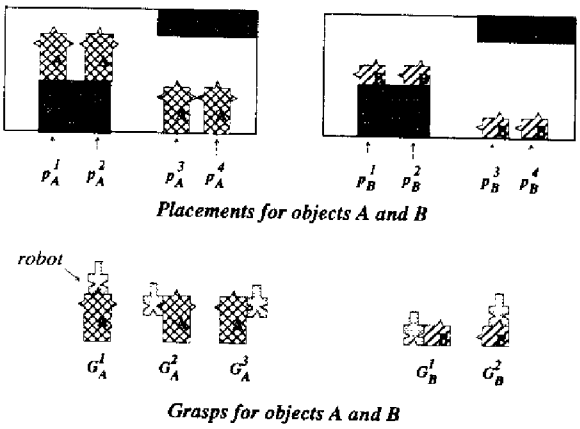


Figure 2: *Placements and grasps for objects* A *and* B

2.2 Building the manipulation graph

2.2.1 Building the nodes

Intuitively, $GRASP \cap PLACEMENT$ is simply the set of all configurations where the robot is authorized to grasp or un-grasp an object in a given placement, taken into account all possible placement combinations for the other objects.

Let I(*grasp* ; *placementA* ; *placementB*) designate the set of all free configurations where the robot can grasp one object using *grasp* while objects are placed in *placementA* and *placementB*.

Thus, $GRASP \cap PLACEMENT$ is the union of all $I(G_A^i ; p_A^j ; p_B^k)$ and $I(G_B^i ; p_A^j ; p_B^k)$.

By definition, $I(G_A^i ; p_A^j ; p_B^k)$ equals

$$\mathsf{Free}(\{cr \in CSR \mid G_A^i(cr) = p_A^j\} \times \{p_A^j\} \times \{p_B^k\})$$

The computation of $\{cr \in CSR \mid G_A^i(cr) = p_A^i\}$ is done using the robot inverse geometric model. If we assume that the robot is not redundant and that the solution does not include a robot singular configuration, then $I(G_A^i ; p_A^j ; p_B^k)$ consists of a finite (and small) number of configurations.

Each node of MG corresponds to a configuration which can be computed easily; it represents a connected component of $GRASP \cap PLACEMENT$ reduced to a single configuration.

2.2.2 Building the edges

The second step in building MG consists of establishing edges between nodes. Two nodes N_1 and N_2 are linked by a transit (resp. transfer) edge if there exists a transit-path (resp. transfer-path) between two configurations of their associated connected components.

All paths involving a node have an extremity in common: the single configuration represented by the node. For a node in $I(G_A^i\ ;\ p_A^j\ ;\ p_B^k)$, all transit-paths will correspond to paths where the robot moves alone while A and B are in p_A^j and in p_B^k, and all transfer-paths will correspond to paths where the robot holds A using grasp G_A^i while B is in p_B^k. This is why we define the concept of *task state* which denotes the fact that the robot is in a situation where it moves alone, or it is holding a given object.

Task states and CS slices: We define a *task state* by giving for each object its current placement and, for at most one object, its current grasp: $(grasp\ ;\ placementA\ ;\ placementB)$.

When no object is grasped, *grasp* will be noted "_", for example $(\ \_\ ;\ p_A^1\ ;\ p_B^2)$. We call such a state a *transit state*.

When object X is held by the robot, *placementX* will be noted "_", for example $(G_A^2\ ;\ \_\ ;\ p_B^4)$. We call such a state an *transfer state*.

Let $C(grasp\ :\ placementA\ ;\ placementB)$ denote the set of configurations in CS associated to a given task state. $C(\ \_\ ;\ p_A^i\ ;\ p_B^j)$ and $C(G_A^i\ ;\ \_\ ;\ p_B^j)$ correspond to n-dimensional "slices" of CS where n is the dimension of CSR.

For a transit state:

$$C(\ \_\ ;\ p_A^i\ ;\ p_B^j) = \mathsf{Free}(CSR \times \{p_A^i\} \times \{p_B^j\})$$

i.e. all configurations such that the robot does not overlap object A in placement p_A^i nor object B in placement p_B^j nor the obstacles.

For a transfer state:

$$C(G_A^i\ ;\ \_\ ;\ p_B^j) = \mathsf{Free}(CSR \times G_A^i(CSR) \times \{p_B^j\})$$

i.e. all configurations such that the robot holding object A in grasp G_A^i does not overlap object B in placement p_B^j nor the obstacles.

Remark: When $C(G_A^i\ ;\ \_\ ;\ p_B^j) = \emptyset$ the corresponding state is invalid.

A node models a transition between a transit state and a transfer state. We say that the node "belongs" to these states. For example, a node in $I(G_A^i\ ;\ p_A^j\ ;\ p_B^k)$ "belongs" to the transit state $(\ \_\ ;\ p_A^j\ ;\ p_B^k)$ and to the transfer state $(G_A^i\ ;\ \_\ ;\ p_B^k)$.

Transit edges: A transit edge can be built between a node N and any node N' which belongs to the same transit state and which is "directly" reachable. This simply means that N and N' represent configurations that are in a same connected component of $C(\ \_\ ;\ p_A^i\ ;\ p_B^j)$.

Transfer edges: A transfer edge can be built between a node N and any node N' that belongs to the same transfer state and that is "directly" reachable. This simply means that N and N' represent configurations that are in a same connected component of $C(G_A^i\ ;\ \_\ ;\ p_B^j)$ or $C(G_B^i\ ;\ p_A^j\ ;\ \_\ )$

We have then to construct two CS slices for any given node. However, a given CS will be used for a great number of nodes.

Figure 3 represents several nodes in the manipulation graph that corresponds to the example. The drawing at the center of the figure represents the node $I(G_B^2\ ;\ p_A^3\ ;\ p_B^2)$. Transit and transfer edges are built using $C(\_\ ;\ p_A^3\ ;\ p_B^2)$ and $C(G_B^1\ ;\ \_\ ;\ p_B^2)$.

Figure 4 illustrates the "links" between several configuration space slices that are traversed by the system when it executes the sequence represented in Figure 1. Several states are represented; for each state, the regions in white represent the projection of the connected components of its CS slice onto the robot configuration space. In the initial state of Figure 1, the robot is in the "left" connected component of $C(\ \_\ ;\ p_A^2\ ;\ p_B^4)$. The only possible transition (arc 11) is to move the robot until it is able to grasp object A in G_A^2. The transitions sequence is 11-10-7-9-6-5...Note that the solution involves state $(\ \_\ ;\ p_A^2\ ;\ p_B^4)$ twice, but it traverses only once a given connected component of $C(\ \_\ ;\ p_A^2\ ;\ p_B^4)$.

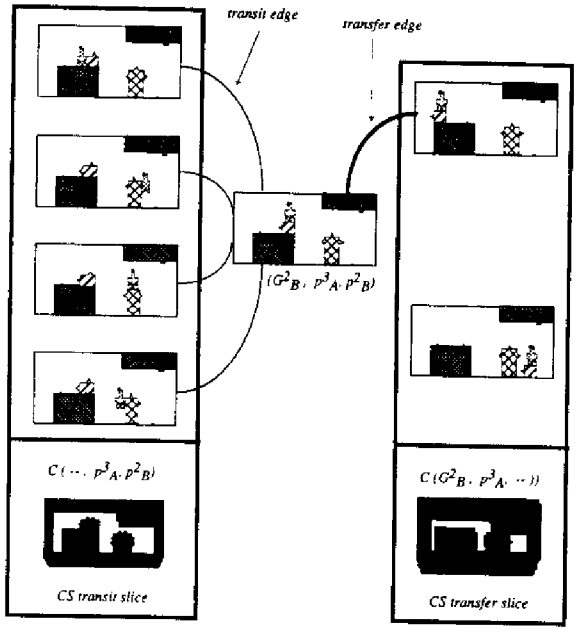


Figure 3: *A partial representation of a manipulation graph*

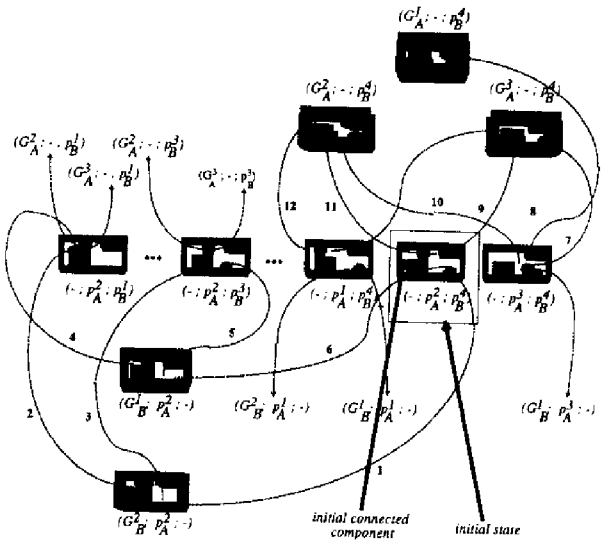


Figure 4: *Links between Configuration Space slices*

2.3 Search Strategies in the Manipulation Graph

The size of the graph grows rapidly depending on the number of grasps and placements. The number of nodes corresponds to *number of grasps* × *number of placements* (in our simple example, there are $(3+2) \times (4 \times 4) = 80$ nodes). The number of transit slices is equal to the number of legal placements ($(4 \times 4) - 4 = 12$ in the example). The number of transfer slices is equal to the number of combinations of grasps for an object and placements for the other objects ($3 \times 4 + 2 \times 4 = 20$ in the example).

The cost of building an edge is expensive and depends mainly on the cost of computing a n-dimensional CS slice (where n is the number of degrees of freedom of the robot). However, a CS slice is used several times; for example $\mathsf{C}(\ \_\ ;\ p_A^j\ ;\ p_B^k)$ will be used for all nodes in $\mathrm{I}(\ \mathsf{G}_A^i\ ;\ p_A^j\ ;\ p_B^k)$ and in $\mathrm{I}(\ \mathsf{G}_B^i\ ;\ p_A^j\ ;\ p_B^k)$. The first time, it has to be computed; and then, it will only be used in order to find a path.

MG has not to be built completely before execution. It can be explored and built incrementally. Powerful heuristics remain to be explored in order to "drive" the system towards the goal. However, even simple heuristics based only on the distance between the positions of objects allow to limit substantially the construction of the graph.

Note that, if several objects have the same shape and the same grasps and placements, the number of different CS slices to build can be considerably reduced. In the case, similar to the example, of two identical objects with 3 different grasps and 4 placements, we have only 12 transfer slices and 6 transit slices. Then, another way to limit the complexity, when exploring the graph edges, is to consider only a gross approximation of objects shape (by classifying them into a limited number of classes: small, elongated, big...) in order to use a same CS slice for a great number of nodes.

2.4 Implementation

We have implemented a system based on the method described above. It is composed of two modules: a *Manipulation Task Planner* and a *Motion Planner*.

The *Manipulation Task Planner* builds incrementally the manipulation graph and searches solution paths in it. It makes use of the Motion Planner in order to

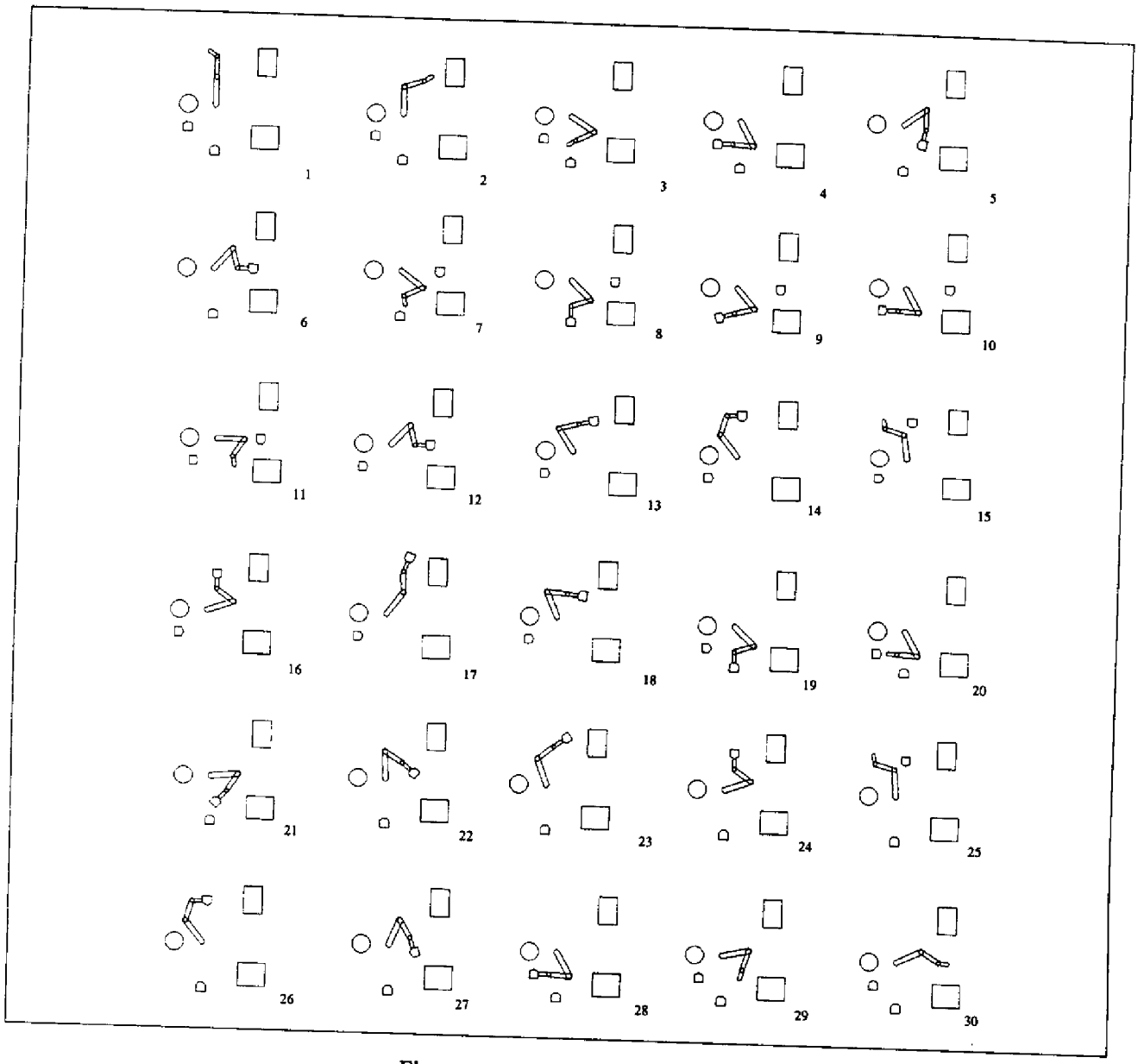


Figure 5: *A manipulation task.*

build the different *CS* slices corresponding to transfer and transit states, to structure them into connected components and to find paths between two robot configurations.

The search in the manipulation graph is performed using a A* algorithm. The cost functions currently used are based on the length of the movable bodies trajectory. The incremental graph construction allows a nice feature: for the first plans, the system is quite slow but it becomes more efficient progressively as it

re-uses parts of the graph already developed.

The *Manipulation Task Planner* is implemented in order to be used for an arbitrary number of objects and does not depend on a specific motion planner module.

For the *Motion Planner*, we have first implemented a method working for a polygonal body in translation amidst polygonal obstacles (the obstacles are grown using Minkowsky sum and the trajectory is built using a visibility graph). This has been done mainly in order to demonstrate the feasibility of the approach. Figure 1 shows the plan produced for the example.

A second implementation is based on a general motion planner [17] which works for manipulators is a 3-dimensional workspace. It allows to solve manipulation planning problems as complicated as the example of Figure 5. Note that the sub-sequences 14-15-16 and 24-25-26 correspond to re-grasping operations of an object.

3 The polygonal case for one movable object with an infinite set of grasps

This section describes a method for solving the manipulation problem in the case of a polygonal robot and a polygonal object moving in translation amidst polygonal obstacles. It has been implemented in the case where both polygons are convex. In order to illustrate the different steps, we will rely on the simple example of Figure 6.

Let us consider $CS = CSR \times CSO$ the configuration space of the robot and the object together. In order to simplify the notations, we denote by :

- ACS the admissible (i.e. without any collision between the bodies) configurations space ($ACS = \mathsf{Free}(CS)$),
- $ACSR(co)$ the admissible configuration space of the robot when the movable object is placed at $co \in CSO$, and
- $ACSO(cr)$ the admissible configuration space of the movable object when the robot lies at $cr \in CSR$.

We assume that the robot has to avoid any contact with the obstacles (hypothesis H, see Appendix). We define $GRASP$ as the subset of all the configurations verifying hypothesis H and such that the robot touches the object. Finally, the movable object may be placed anywhere in the environment, as long as the bodies do not collide. As a consequence, $GRASP \cap PLACEMENT = GRASP$.

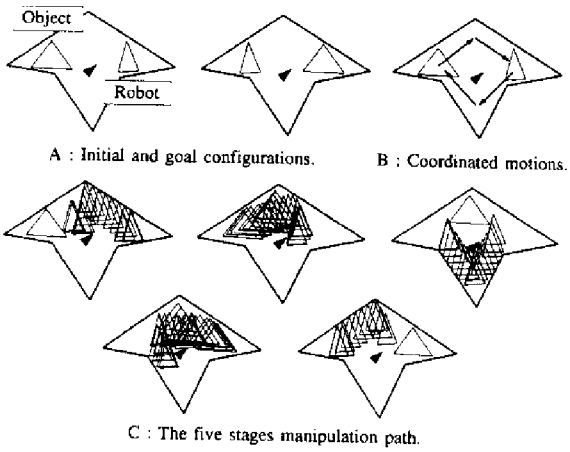


Figure 6: *An illustration of manipulation constraints.*

Let $\mathcal{C}$ be the set of connected components of $GRASP$. Let us consider the graph whose nodes are the elements of $\mathcal{C}$ and whose edges correspond to the existence of a transit-path between two configurations of the associated nodes. Thanks to the reduction property (see Appendix), two configurations in $GRASP$ are connected by a manipulation path if and only if they belong to two elements of $\mathcal{C}$ which are in the same connected component of the graph.

Therefore, according to the resolution scheme stated in Section 1, a manipulation graph can be built by :

1. computing the connected components of $GRASP$,
2. and linking them by transit-paths.

In a first step, we compute a cell decomposition of ACS (which solves the coordinated motion problem); then a retraction on the boundary of ACS gives a cell decomposition of $GRASP$. Finally, the connectivity by transit-paths between the various connected components of $GRASP$ is given by a study of the connectivity of $ACSR$, whose structure can be extracted from ACS cells.

3.1 *ACS* cell decomposition and coordinated motions

To compute the cell decomposition of ACS, we have chosen to use an adaptation of the projection method developed by Schwartz and Sharir in [15] for the case of two discs. Any other, and perhaps better ([7, 14, 16]) method could have been used. However, the purpose of this paper is not to give some optimal algorithm, but to demonstrate the feasibility of our approach.

Let us consider an object position co. $ACSR(co)$ is obtained by removing, from $ACSR$ (robot admissible configurations without the object), the set $COL(co)$ of all configurations where the robot and the object collide [3].

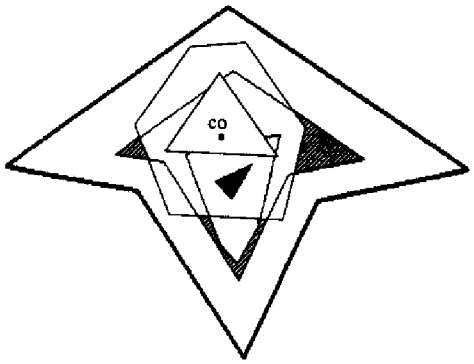


Figure 7: *The hatched areas represent the connected components of $ACSR(co)$.*

Let us observe now the evolution of $ACSR(co)$ with respect to co. In a neighborhood of most object positions, $ACSR(co)$ varies only quantitatively, keeping the same structure. However, at some object positions, some $ACSR(co)$ connected components may appear, disappear, be split or merged. More precisely, the geometrical structure of the connected components of $ACSR(co)$ are modified when some vertices appear or disappear. These changes correspond to specific values of co which constitute a set of *critical curves* (see [15] for a proof). Figure 8 gives an example of a critical curve along which a connected component of $ACSR(co)$ is divided into two separate components.

[3] $COL(co)$ is the polygon obtained by Minkowski difference between the robot and the object, and placed in co.

The critical curves provide a decomposition of $ACSO$ into *non-critical regions*.

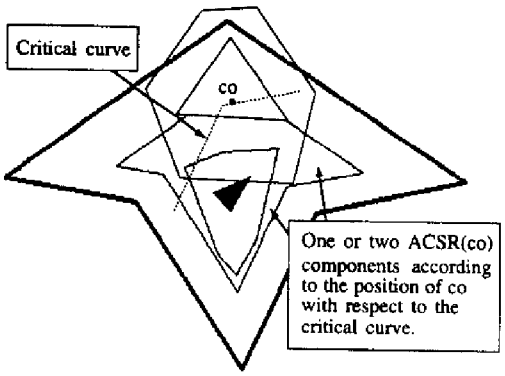


Figure 8: *A critical curve*

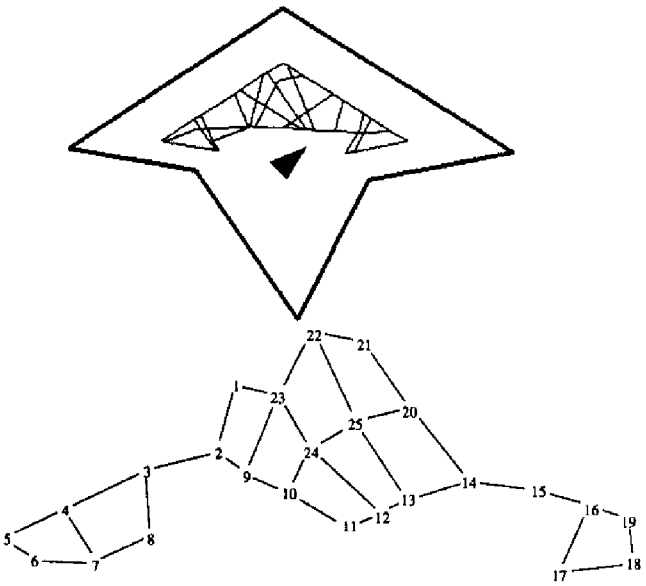


Figure 9: *Cell decomposition of ACSO and the associated graph*

Let us consider a non-critical region R. $\{\{co\} \times ACSR(co) \mid co \in R\}$ constitutes cells of ACS (there are as many cells as the number of connected components of $ACSR(co)$). The set of all such cells is the

expected cell decomposition.

In order to compute the critical curves, we introduce a symbolic description of $ACSR(co)$. Let us recall that the boundary of $ACSR(co)$ is constituted by $ACSR$ edges and $COL(co)$ edges. We assign a numerical label to all the vertices of $ACSR$ and a literal symbol to the edges of $COL(co)$. We denote by $b[8,9]$ the intersection between the edge b of $COL(co)$ and the segment $[8,9]$ of $ACSR$. Therefore, the bottom connected component in the example of Figure 10 is labeled by the sequence $(3,4,[4,5]b,b[8,9],9,[9,10]b,b[1,2],2)$.

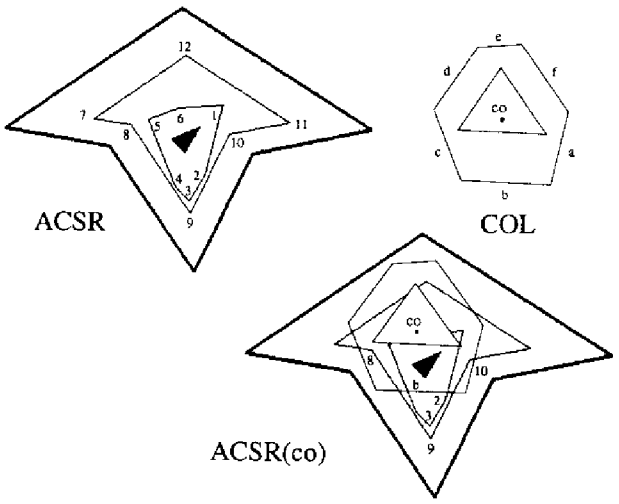


Figure 10: *An illustration of the labels used to characterize the connected components of ACSR(co).*

To compute the critical curves, we do not consider all the possible changes in this sequence. We just need to consider the changes *on the letters* (i.e. the changes induced by $COL(co)$ and not by $ACSR$ vertices). In Figure 10 for instance, when co moves to the bottom, the disappearance of vertex 2 in the sequence above does not induce a critical curve, while the fusion of the two b labels (when edge b meets vertex 3) does.

The critical curves of our example are shown in Figure 9, together with the graph of non-critical regions of $ACSO$.

Let us recall that each non-critical region induces as many cells in ACS as the number of connected components in $ACSR(co)$ when co belongs to the region.

Now we structure all these cells into a *coordinated motion graph*. Two cells are adjacent in this graph if and only if :

- the associated non-critical regions are adjacent in $ACSO$,
- and the symbolic descriptions of the associated $ACSR(co)$ components just differ by a letter.

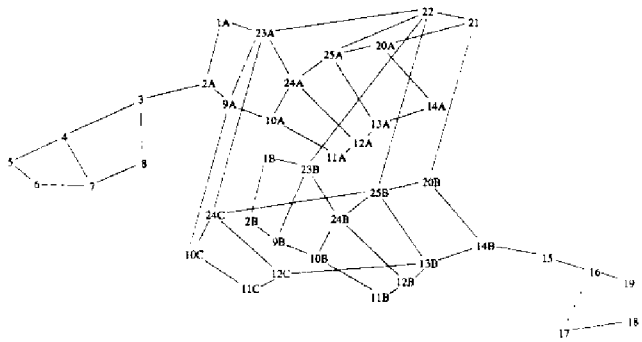


Figure 11: *The coordinated motion graph.*

Figure 11 shows the coordinated motion graph of our example. In Figure 10, co belongs to the non-critical region 10 (Figure 9). This region gives rise to three ACS cells numbered $10A$, $10B$ and $10C$ in Figure 11. $10C$ is the node corresponding to the label mentioned above.

The fundamental property is : there exists a coordinated motion between two configurations in ACS if and only if they belong to nodes of a same connected component of the coordinated motion graph. The proof is exactly the same as in [15].

3.2 *GRASP* cell decomposition and contact motion

Let us consider the above $ACSR(co)$ component labeled by $(3,4,[4,5]b,b[8,9],9,[9,10]b,b[1,2],2)$. There are two edges labeled by b in it. This means that there are two connected sets of configurations where the robot is in contact with the object. It is not possible to go from one set to the other one without leaving the contact. These connected sets are easily extractable from the symbolic description of the $ACSR(co)$ components. In our example,

$([4,5]b, b[8,9])$ and $([9,10]b, b[1,2])$ are the symbolical descriptions of the two classes of contact. By definition, they always contain two terms.

Now, let us consider a non-critical region R. The set $\{\{co\} \times COL(co) \cap ACSR(co) \mid co \in R\}$ constitutes cells of $GRASP$ (there are as many cells as the number of connected components of $COL(co)$ along the $ACSR(co)$ boundary).

We follow the same method as for the coordinated motion problem. We structure the $GRASP$ cells into a *contact graph*. Two cells are adjacent in this graph if and only if :

- the associated non-critical regions are adjacent in $ACSO$,
- and their symbolic descriptions just differ by one term.

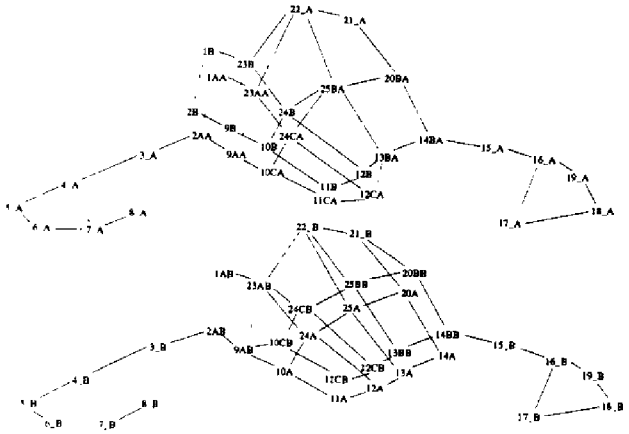


Figure 12: *The contact graph.*

Figure 12 shows the contact graph of our example. Region 10 gives rise to three ACS cells. The frontier of two of them ($10A$ and $10B$) contain one connected component in $GRASP$. They then give rise to two $GRASP$ cells (which keep the same name in the contact graph). The frontier of $10C$ contains two connected components in $GRASP$; they give rise to $GRASP$ cells $10CA$ and $10CB$.

This graph verifies the following property : there exists a motion keeping the contact between the robot and the object, between two configurations in $GRASP$, if and only if both configurations belong to nodes of a same connected component of the contact graph. The proof is exactly of the same kind as for the coordinated motion graph.

3.3 The manipulation graph

Let us come back to our manipulation planning problem. At this time we have captured the connectivity of $GRASP$. We know that two configurations in the same connected component of $GRASP$ may be linked by a manipulation path (Reduction Property).

Now we have to study the existence of transit-paths between $GRASP$ components. This study is very easy from the above labeling. Indeed let us consider $ACSR(co)$ in Figure 10. There are two grasp classes in the bottom component. Nevertheless, it is possible for the robot to move *alone* in this component; that means that the robot can go from a position in the first grasp class to any other one in the second grasp class. Then, these two classes are linked by transit-paths.

The existence of such transit-paths is very easy to compute from the labeling of $ACSR(co)$. Indeed, two $GRASP$ cells are connected by a transit-path if and only if they belong to the frontier of the same ACS cell. In our current example, only the cells $10CA$ and $10CB$ (which come from the same ACS cell $10C$) are linkable by a transit path.

Computing the connectivity of $GRASP$ components by transit-path is equivalent to adding to the contact graph edges between nodes defined from a same ACS cell. These additional edges are referred as "transit edges" in Figure 13. With our notations, two nodes in the contact graph whose "names" contain the same number and the same first letter are linked by a transit edge. The resulting graph is the *manipulation graph.*

3.4 Manipulation path finding

Let us consider an initial configuration c_i and a final one c_f, defining the initial and final positions of the robot and the object. According to the property of the manipulation graph, we use a three-steps procedure :

1. First, we compute the ACS cells C_i and C_f containing c_i and c_f. Then, we compute the set $\mathcal{G}_i$ (resp. $\mathcal{G}_f$) of $GRASP$ cells reachable from c_i (resp. c_f). This computation is a 2-dimensional problem

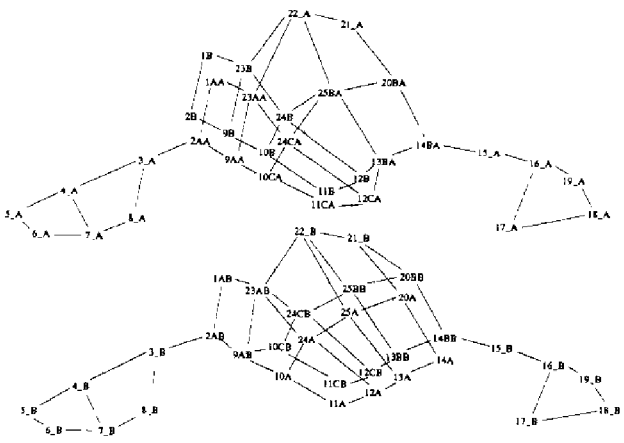


Figure 13: *The manipulation graph, where dotted lines describe transit edges.*

since the transit-paths have to lie in $ACSR(co_i)$ (resp. $ACSR(co_f)$).

2. The second step consists in searching a path in the manipulation graph between a cell in $\mathcal{G}_i$ and a cell in $\mathcal{G}_f$. If no such path exists, the procedure stops. There is no solution. Otherwise, we obtain a sequence $\mathcal{P}ath$ of $GRASP$ cells.
3. Finally the complete path is built from elementary manipulation paths lying in the $GRASP$ cells of $\mathcal{P}ath$ and from transit-paths associated to the transit edges contained in $\mathcal{P}ath$.

Comments on Step 1 : The computation of C_i (resp. C_f) is a 2-dimensional location problem performed in $ACSR(co_i)$ (resp. $ACSR(co_f)$) : we have to determine the connected component of $ACSR(co_i)$ (resp. $ACSR(co_f)$ containing cr_i (resp. cr_f). This fully characterizes C_i (resp. C_f). Then the computation of $\mathcal{G}_i$ (resp. $\mathcal{G}_f$) is very easy, since these $GRASP$ cells belong to the frontier of C_i (resp. C_f) : with our notations, a $GRASP$ cell belonging to the frontier of some ACS cell appears in the contact graph with the same numerical label and the same first letter as the ACS cell appears in the coordinated motion graph.

Comments on Step 2 : The second step is performed using a A^* algorithm. Several cost criteria can be introduced : the length of the complete path, the number of grasping changes...The experimental results will give an illustration of the influence of the cost definition.

Comments on Step 3 : Step 3 consists in computing the complete manipulation path. $\mathcal{P}ath$ is a sequence of $GRASP$ cells. Two consecutive cells in this sequence are linked either by an edge already appearing in the contact graph, or by a transit edge. Moving inside a $GRASP$ cell needs a specific procedure : we have implemented the method presented in the proof of the reduction property . Finally, there remains to compute the transit-path associated to a transit edge : this is a 2-dimensional problem solved in some $ACSR(co)$ slice by a visibility graph method for instance (which is the method we have implemented).

Remark : The algorithm can be adapted in order to take into account partial goals : in this case the goal configuration describes only a goal position for the object (resp. the robot) without specifying a goal position for the robot (resp. the object). Such an extension is easy when the robot goal is unspecified and the object goal is known (the identification of the goal node is given by the non-critical region containing the object). The case of an unspecified object goal is also tractable, but would require some tedious algorithmic details.

3.5 Complexity

In order to evaluate the complexity of the algorithm, we have to distinguish the *decision* part of the manipulation problem (i.e. proving the existence of a solution) from the *complete* problem (i.e. the computation of a solution if any).

In our algorithm, the decision problem is solved by building and searching the graph. The complexity of this part is clearly dominated by the construction of the non-critical regions. Let us denote by n_e, n_r and n_o the number of vertices of the environment, the robot and the object respectively. We assume that both object and robot are convex.

All the critical curves lie in $ACSO$ (Figure 9), whose computation can be done in $O(n_e n_o \log(n_e n_o))$; moreover, the complexity of the admissible configuration space of the object is in $\Omega(n_e n_o)$ (see for instance [6]). Similarly the complexity of the admissible robot configurations subset $ACSR$ is in $\Omega(n_e n_r)$ and can be computed in $O(n_e n_r \log(n_e n_r))$.

The critical curves are defined by the coincidence between an edge (resp. a vertex) of $ACSR$ and a vertex (resp. an edge) of $COL(co)$ boundary (i.e. the set of contacts between the object and the robot). The edge number of $COL(co)$ is exactly $(n_o + n_r)$. Then, the number of critical curves is in $\Omega(n_e n_r(n_o + n_r))$.

The cell decomposition of $ACSO$ is given by the computation of the intersections between the $\Omega(n_e n_r(n_o + n_r))$ critical curves, and the $\Omega(n_e n_o)$ edges of $ACSO$ boundary. This can be done in $O((n_e n_r(n_o + n_r) + n_e n_o)^2 \log(n_e n_r(n_o + n_r) + n_e n_o))$ and gives $\Omega((n_e n_r(n_o + n_r) + n_e n_o)^2)$ non-critical regions.

Finally, in each non-critical region, the number of connected components of $ACSR(co)$ is in $O(n_e n_r)$. They are computed by intersection of the $(n_r + n_o)$ edges of $COL(co)$, and the $\Omega(n_e n_r)$ edges of $ACSR$, for each non-critical region, each time placing the object on some point of the region. Such an intersection is computed in $O((n_r+n_o)n_e n_r \log(n_r+n_o+n_e n_r))$. The manipulation graph is then obtained in $O((n_e n_r(n_o + n_r) + n_e n_o)^2(n_r + n_o)n_e n_r \log(n_r + n_o + n_e n_r))$.

The number of robot and object edges may be considered to be small comparing with n_e. Then, defining n as the environment complexity, previously called n_e, our algorithm runs in $O(n^3 \log(n))$ time. We did not try to optimize it at this time. Perhaps more sophisticated cell-decomposition (like [16]) could be used in the same framework.

Finally, the complexity of the complete problem (i.e. the computation of a solution path), is dominated in the worst case by the number of elementary manipulation paths (see Appendix). Therefore the complexity of the complete problem not only depends on the complexity of the environment but also on the "clearance" of the robot in the environment.

3.6 Experimental results

Figures 14 to 17 show results obtained with the above described implementation.

The solution given by the planner to our illustrative example is shown in Figure 6C. The initial and final object positions are respectively in the non-critical regions 6 and 19 (Figure 9). The initial and final configurations of the manipulation problem are respectively in the ACS cells 5 and 18 (Figure 11).

Let us consider now the solution path (5_B, 4_B, 3_B, 2AB, 9AB, 23AB, 22_B, 22_A, 25BA, 13BA, 14BA, 15_A, 16_A, 19_A, 18_A). Figure 6Ca shows the transit-path along which the robot first reaches the object. In Figure 6Cb, the object is moved with a sequence of transit and transfer paths along the same connected component of $GRASP$. If we refer to the manipulation graph (Figure 13), the manipulation path is built, at this time, from the sequence (5_B, 4_B, 3_B, 2AB, 9AB, 23AB, 22_B) of $GRASP$ cells. A transit edge appears between vertices 22_B and 22_A; the associated transit path is shown in Figure 6Cc. Figure 6Cd describes the subsequence (22_A, 25BA, 13BA, 14BA, 15_A, 16_A, 19_A, 18_A). A last transit-path allows to reach the robot goal configuration.

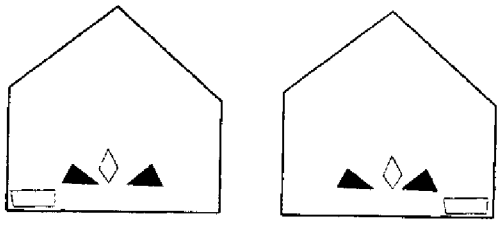


Figure 14: *Initial and goal configurations.*

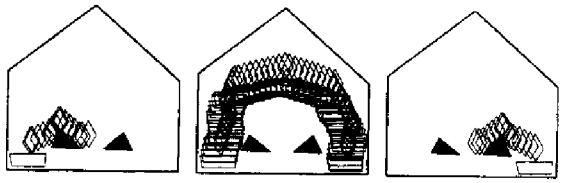


Figure 15: *Solution using a cost function that minimizes the number of grasps.*

Figures 16 to 15 illustrate the influence of heuristics used to find a path through the manipulation graph.

In Figure 16, the weights of transit and transfer links of manipulation graph are nearly equivalents, each referring to the straight line distance between reference points of the $GRASP$ cells. Then the path found by the search algorithm involves numerous transit-paths and re-grasps.

On the contrary, Figure 15 describes the obtained

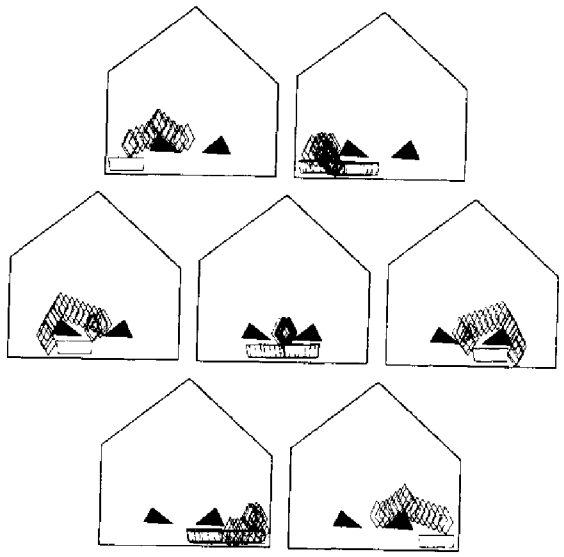


Figure 16: *Solution using a cost function that minimizes the length of the object path.*

solution when transit-paths are heavier, due to a simple multiplying coefficient. A path is then found that allows to keep the same grasp along the whole path.

Finally, Figures 17 and 19 show the solution given to a more intricate situation, involving numerous re-grasps. The second and third images detail a transit and a transfer path at the beginning of a contact motion. The heuristic used there takes into account the average length of available grasp frontier for each grasp cell. Then it avoids narrow grasps which could involve numerous re-graspings.

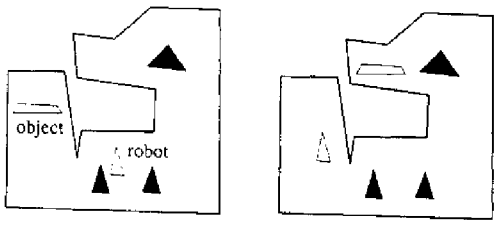


Figure 17: *A more intricate situation.*

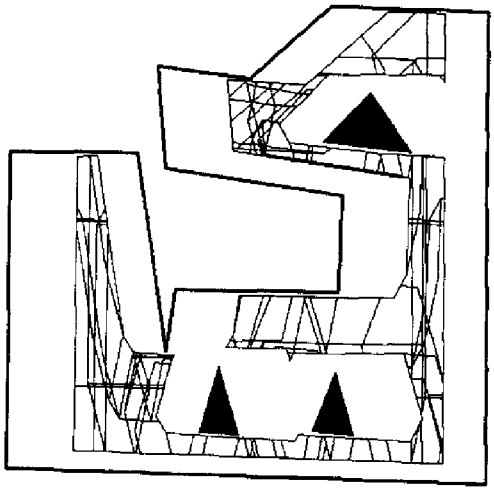


Figure 18: *Non-critical regions.*

The algorithm has been implemented in C on a Sun SPARC Station 2. The computation of the manipulation graph of Figure 13 (including the computation of the non-critical regions) takes 3.4 seconds. The search for a path takes 0.25 seconds for the example in Figure 6. In this simple case, the cell decomposition of the admissible configurations of the object gives 25 non-critical regions, and the manipulation graph consists of 69 nodes and 126 edges. In the example of Figure 14, we obtain 146 non-critical regions, 257 nodes and 475 edges. The case of Figure 17 gives 288 non-critical regions (shown in Figure 18) and its manipulation graph consists of 619 nodes and 1115 edges.

4 Related work and open problems

The first paper that attacks motion planning in presence of movable obstacles is [18]; in this paper, Wilfong gave the first results on the complexity of the problem : he proved that the problem is $PSPACE$-hard (resp. NP-hard) in two dimensional environments where only translations are allowed and when the final configuration specifies (resp. does not specify) the final positions of all the movable objects. In the same reference, Wilfong gives a solution in $O(n^3 \log^2 n)$ (where n is the number of vertices of a polygonal environment) for the case of a convex polygonal robot moving in translation

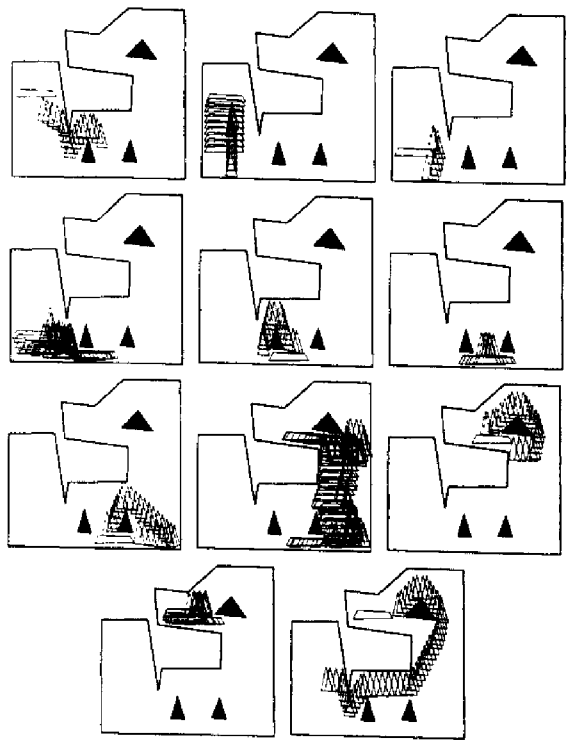


Figure 19: *A manipulation path.*

amidst one polygonal object (with a finite grasp set) and obstacles.

In [12] we presented a general algorithm for the case of one robot and one movable object. We showed [4] how to decompose the space of grasping configurations into a finite number of cells in order to make the problem tractable. However the method makes use of general tools from algebraic geometry, leading to an inefficient algorithm in practice.

More recently, several authors attacked the manipulation planning problem in various contexts.

Koga and Latombe [8, 9] addressed the case of multi-arm manipulation planning where several robots have to cooperate to carry a single movable object amidst obstacles. In this context, re-grasps of the object are often required along the path to avoid collisions and may involve changing the arms grasping the object. In [8] they propose several implemented planners to deal with problems of increasing difficulty for two identical arms in a 2D workspace. For problems involving many degrees of freedom, which is usually the case in multi-arm manipulation, they use an adapted version of the randomized potential field planner [2]. An extension to a robot system made of three 6 DOF arms in a 3D workspace is also presented in [9]. In this work, the number of legal grasps of the object is finite and the algorithms require the object to be held at least by one robot at any time during a re-grasp operation. The approach relies on several simplifications, but it yields to impressive results for complex and realistic problems.

The approach developed by Barraquand and Ferbach [3] consists in translating the manipulation planning problem into convergent series of less constrained subproblems increasingly penalizing the motions that do not satisfy the constraints. Each subproblem is solved using variational dynamic programming.

[4] describes a heuristic algorithm for a circular robot and where all the obstacles can be moved by the robot in order to find its way to its goal.

Finally, let us pinpoint the interesting extension of the manipulation planning problem attacked by Lynch and Mason [13]. In their context, grasping is replaced by pushing. The space of stable pushing directions imposes a set of nonholonomic constraints on the robot motions, which opens issues of controllability.

All the above mentioned studies contribute to establish the manipulation planning problem as a specific and challenging instance of the motion planning problem with constraints. However, several open questions remain, ranging from a theoretical analysis of the problem to the investigation of new practical instances. One key theoretical aspect concerns the conditions under which the reduction property can be extended to the case of several objects and robots. On a practical point of view, the problem represents also a challenge to motion planning techniques because of its additional complexity.

Acknowledgement : This work has been partially supported by the Esprit Programme through Basic Research Action 6546 PROMotion. We are grateful to B. Dacre-Wright who implemented the algorithm described in Section 3.

[4] The principles are similar to those presented in Section 3, but they are established for general robot systems.

Appendix : Reduction property

In this appendix, we consider only one movable object. In this case, two configurations belonging to a same connected component of $GRASP \cap PLACEMENT$ are m-connected. In fact, this property holds up to a precise definition of what is a grasping configuration.

Let us consider $CS = CSR \times CSO$ the configuration space of the robot and the object together. Let us denote by $CONTACT$ the domain of the configurations where the robot and the object are in contact. $CONTACT$ is a subset of Free(CS) boundary.

Hypothesis H: We assume that the robot has to avoid any contact with the obstacles.

We then define $GRASP$ as the subset of $CONTACT$ of all the configurations verifying hypothesis H.

Property 1 *Two configurations of a same connected component of* GRASP ∩ PLACEMENT *are connected by a manipulation path.*

Proof: Let a and b be two configurations in a connected component of $GRASP \cap PLACEMENT$. There exists a path $p : [0,1] \mapsto GRASP \cap PLACEMENT$ linking these two configurations[5] ($p(0) = a$, $p(1) = b$). We define p_r and p_o as the projections of p onto CSR and CSO respectively.

Let $c = p(t)$ be any configuration on the path. Thanks to the hypothesis H, $p_r(t)$ lies in an open set of Free(CSR). We then can find an open disc $D_\epsilon \subset$ Free(CSR) centered on $p_r(t)$ and with a radius $\epsilon > 0$.

Since p is continuous, there exists $\eta_1 > 0$ such that :

$$\forall \tau \in]t-\eta_1, t+\eta_1[, p_r(\tau) \in D_{\epsilon/2}.$$

Similarly, $p_r - p_o$ is a continuous function. Then there exists $\eta_2 > 0$ such that, for any $\tau \in]t-\eta_2, t+\eta_2[$,

$$\| (p_r(\tau) - p_o(\tau)) - (p_r(t) - p_o(t)) \|_{\mathbf{R}^2} < \epsilon/2.$$

This last assertion means that the relative grasp configuration does not vary more than $\epsilon/2$ along the path p between $p(t-\eta_2)$ and $p(t+\eta_2)$.

Let us consider $\eta = min\{\eta_1, \eta_2\}$:

$$\forall \tau, \sigma \in]t-\eta, t+\eta[, \quad p_o(\sigma) + (p_r(\tau) - p_o(\tau)) \in D_\epsilon. \quad (1)$$

[5] p designates a path as well as the associated function.

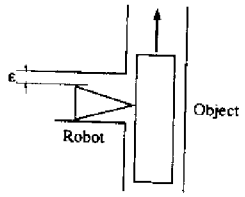


Figure 20: *A case with finite (but great) number of transit and transfer paths.*

Let $c_1 = p(\tau_1)$ and $c_2 = p(\tau_2)$ be any two configurations on p, with τ_1 and τ_2 in $]t-\eta, t+\eta[$ (we assume that $\tau_1 < \tau_2$). We prove now that c_1 and c_2 can be linked by one transfer path followed by one transit path.

Let us consider the path $(p_o(\tau), p_o(\tau) + (p_r(\tau_1) - p_o(\tau_1)))$, with $\tau \in [\tau_1, \tau_2]$. This path is clearly a transfer path with constant grasp $(p_r(\tau_1) - p_o(\tau_1))$, between $p(\tau_1)$ and $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau_1) - p_o(\tau_1)))$. According to relation 1, this path is admissible. Let us consider the path $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau) - p_o(\tau)))$, with $\tau \in [\tau_1, \tau_2]$. This path is clearly a transit path between $(p_o(\tau_2), p_o(\tau_2) + (p_r(\tau_1) - p_o(\tau_1)))$ and $p(\tau_2)$. Again, according to relation (1), this path is admissible. The concatenation of both paths constitute a manipulation between $p(\tau_1)$ and $p(\tau_2)$.

As path p_r is a compact set included in an open set of Free(CSR), we can apply this local transformation on a *finite* covering of $[0,1]$. We have then a finite number of elementary manipulation paths which constitutes a manipulation path linking a and b. □

Remark : the number of elementary manipulation paths used in the proof of the reduction property depends on the clearance of p_r in Free(CSR). The worst case is reached in the example Figure 20 where the number of elementary paths is clearly in $O(\frac{1}{\epsilon})$.

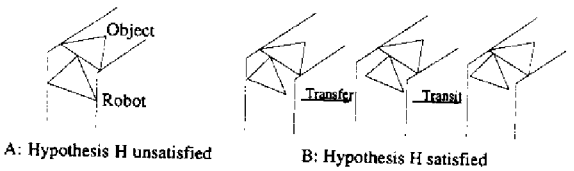


Figure 21: *Illustration of hypothesis H.*

Note that the reduction property does not hold when hypothesis H is not satisfied. Figure 21 illustrates this fact. Figure 21A shows an example where both the robot and the object touch the environment. There exists a coordinated path, but no feasible manipulation path (the robot cannot move the object with a constant grasp). Figure 21B shows how the robot can "manipulate" the object even when this one is in contact.

References

[1] R. Alami, T. Siméon, J.P. Laumond, "A geometrical Approach to planning Manipulation Tasks. The case of discrete placements and grasps." International Symposium on Robotics Research, Tokyo, August 1989.

[2] J. Barraquand and J.C. Latombe, "Robot motion planning : a distributed representation approach," in *Int. Journal of Robotics Research*, 10 (6), 1991.

[3] J. Barraquand and P. Ferbach, "A penalty function method for constrained motion planning," in *IEEE* Conf. on Robotics and Automation, 1994.

[4] P.C. Chen, Y.K. Hwang, "Practical path planning among movable obstacles." in *IEEE* Conf. on Robotics and Automation, 1991.

[5] B. Dacre-Wright, J.P. Laumond, R. Alami, "Motion planning for a robot and a movable object amidst polygonal obstacles," in *IEEE* Conf. on Robotics and Automation, 1992.

[6] S.J. Fortune, "A fast algorithm for polygon containment by translation", ICALP, 1985.

[7] S. Fortune, G. Wilfong, C. Yap, "Coordinated motion of two robot arms." in *IEEE* Conf. on Robotics and Automation, 1986.

[8] Y. Koga and J.C. Latombe, "Experiments in dual-arm manipulation planning," in *IEEE* Conf. on Robotics and Automation, 1992.

[9] Y. Koga and J.C. Latombe, "On multi-arm manipulation planning," in *IEEE* Conf. on Robotics and Automation, 1994.

[10] J.-C. Latombe, *Robot Motion Planning*, Kluwer Press, 1991.

[11] J.P. Laumond, R. Alami, "A new geometrical approach to manipulation task planning: the case of a circular robot amidst polygonal obstacles and a movable circular object." Technical Report LAAS/CNRS 88314, Toulouse, Octobre 1988.

[12] J.P. Laumond, R. Alami, "A geometrical approach to planning manipulation tasks in Robotics.", Technical Report LAAS/CNRS 89261, Toulouse, August 1989.

[13] K. M. Lynch, "Stable pushing : mechanics, controllability and planning," in this volume.

[14] D. Parsons, J. Canny, "A motion planner for multiple mobile robots." in *IEEE* Conf. on Robotics and Automation, 1990.

[15] J.T. Schwartz, M. Sharir, "On the piano mover problem III : Coordinating the motion of several independant bodies : the special case of circular bodies amidst polygonal barriers." Int. J. of Robotics Research, 2 (3), 1983.

[16] M. Sharir, S. Sifrony, "Coordinated motion planning for two independent robots." ACM Symposium on Computational Geometry, 1988.

[17] T. Siméon, "Planning collision-free trajectories by a configuration space approach," in *Geometry and Robotics*, J.D. Boissonnat and J.P. Laumond Eds, Lecture Notes in Computer Science, 391, Springer Verlag, 1989.

[18] G. Wilfong, "Motion planning in the presence of movable obstacles." in *ACM* Symp. on Computational Geometry, 1988.

Revue d'Intelligence Artificielle. Vol 8 - n° 4/1994

Planning robust motion strategies for a mobile robot in a polygonal world

Thierry Siméon - Rachid Alami

Laboratoire d'Automatique et d'Analyse des Systèmes du CNRS
7 avenue du Colonel Roche 31077 Toulouse, FRANCE

ABSTRACT. *This paper addresses the problem of planning the motions of a mobile robot moving in a polygonal workspace in presence of uncertainty in sensing and control. We consider a robot equipped with two sensors. The position sensor is based on dead-reckoning; the error then results into a cumulative position uncertainty. A proximity sensor may also be used to overcome the uncertainty accumulated during the motions by localizing the robot with respect to the obstacles of the environment. We propose a planner which produces robust motion strategies composed of sensor-based motion commands which guarantee that, given an explicit model of the error accumulated by the commands, the robot can reach safely its goal with an error lower than a pre-specified value. The approach is based on a geometric analysis of the reachability of appropriate features in the environment.*

RÉSUMÉ. *Cet article aborde le problème de planification de trajectoires pour un robot mobile se déplaçant dans un environnement polygonal en présence d'incertitudes provenant des erreurs de contrôle ainsi que de l'imprécision des capteurs. Le robot considéré est équipé de deux capteurs: le capteur de position utilise l'odométrie et conduit à une erreur cumulative. Un capteur de proximité permet de limiter cette accumulation d'erreur en relocalisant le robot par rapport aux obstacles de l'environnement. Nous proposons un planificateur qui produit une stratégie de déplacement robuste, composée de primitives référencées capteurs. Etant donné un modèle explicite de l'erreur accumulée par ces primitives, le planificateur garantit que le robot atteindra son but avec une erreur inférieure à un seuil donné. L'approche proposée est basée sur une analyse géométrique de l'atteignabilité d'éléments caractéristiques de l'environnement.*

KEYWORDS : *mobile robots, motion planning, uncertainty.*
MOTS CLÉS : *robots mobiles, planification de mouvements, incertitudes.*

1 Introduction

Robot motion planning has received a widespread attention over the past decade [14]. Dealing with uncertainties constitutes an important issue of the motion planning problem since robots operating in real world settings, are faced to several sources of uncertainties arising from control errors, limited sensing accuracy and inaccurate models of the environment. Consequently, the use of motion planners which do not take explicitly into account uncertainties is limited to simple situations where the errors remain small with respect to the task tolerances. This cannot be the case in assembly planning or in the context of mobile robot navigation which constitutes a challenging problem for motion planning in presence of uncertainties.

Indeed, mobile robots are not generally equipped with an absolute localization procedure and accumulate position errors (since the error cannot be bounded to some value, a growing of the obstacles is not sufficient). To overcome the uncertainty accumulated during the motions, the mobile robot has to be equipped with sensors that can provide additional information by identifying some appropriate features of the environment. Among all the collision free paths which may connect two given configurations of the robot, only some of them may allow at execution to acquire enough information from the sensors to reliably guide the robot toward its goal. Dealing with uncertainty in control and sensing may therefore completely change the strategy used to plan the motions and it is preferable to *reason in advance*, on the capacity of the available sensing functions, to generate a robust motion plan composed of sensor-based motion primitives.

This paper reports on a recent work we have conducted concerning the development and the implementation of a robust motion planner for a mobile robot moving in a polygonal environment in presence of uncertainty in control and sensing. Such a planner takes explicitly into account the uncertainty in robot control and produces a robust motion plan composed of sensor-based motion commands. The main originality of this approach is that it considers a set of motion commands which may accumulate errors. It is based on a geometric analysis of the reachability and it generates motion strategies which may allow the robot to reduce its uncertainty.

The paper is organized as follows. In section 2, we briefly discuss related work. Section 3 defines more precisely the problem we propose to tackle. In section 4 and 5, we describe how we perform a geometric analysis of the reachability taking into account the uncertainties, and how this analysis is used by the planner (section 6). We finally present some experimental results and discuss possible extensions.

2 Background and Related work

We do not have here the ambition to provide a detailed analysis of the relevant literature. The reader may refer to a very interesting state of the art and discussion in [14, 15, 12]. Roughly speaking, the problem has been tackled in the literature following two general approaches: several contributions are based on a *two-phase* approach, while other contributions stem from the *preimage backchaining* approach.

- In the *two-phase approach*, a motion plan (or more generally a task plan) is first generated assuming no uncertainty and then, this plan is analyzed

and patched in order to finally produce a robust task plan. The analysis is mainly based on the propagation of uncertainty through the plan and its consequences on plan correctness [23, 17, 3, 21, 24, 20, 13]. The plan is then modified by inserting complementary actions or sensor readings allowing to reduce uncertainty. When a plan cannot be patched, a backtrack can occur when planning is based on *skeleton* or *script* selection [18]. The main interest of such an approach is that it can applied to problems which can be more general than motion planning (e.g. assembly, manipulation) and to robots with a high number of degrees of freedom. Its main drawback is that it is based on an *a priori* decomposition of the planning process and on the assumption that, in most situations, the plan can be locally patched.

- The second approach is the *preimage backchaining approach* which is based on the geometry of the Configuration Space. It has been originally proposed by [19] and analyzed or extended through several contributions [10, 8, 5, 14, 15]. The underlying assumption is that it is necessary to take uncertainty explicitly into account in the planning process itself since it can have drastic consequences on the plan structure.

  A *preimage* for a given motion command and for a given goal region (subset of the C-space), is a set of free configurations from which the command can be started and which guarantee that the robot will reach the goal region and stop inside it. Given a goal region and a start region,*preimage backchaining* consists in finding a sequence of commands (i.e. a plan) such that the inverse sequence allows an iterative construction of preimages starting from the goal and resulting in a preimage which contains the initial region.

  While this problem, in its general formulation gives a useful computational framework, it raises "discouraging" complexity issues [5]. However this, by no means, affects its interest. It remains to find relevant instances of the general problem:

  - which integrate new sensor modalities (proximity sensors, vision) and control algorithms [4, 11];
  - which provide sub-classes of the problem with "better" properties (the complete and polynomial algorithm proposed in [16] applies the backchaining framework to navigating a circular mobile robot through circular landmarks where sensing is supposed to be perfect).
  - or which allow to implement algorithms of practical use even though they are not complete [7].

  The instance of the problem addressed in this paper [1] and the algorithm that we propose fall in these categories.

Note that we also need more elaborate models of different sensor modalities in order to be able to compute where (in the C-Space) they can be used and the nature of the measure they can provide together with its (pre-computed) estimated uncertainty. This should of course be done, taking into account the sensor characteristics (position relative to the robot frame, range, specularity...) in a given environment [9, 22, 6].

[1] a shorter version was published in [1].

3 Problem statement

3.1 The robot and the environment

Let us assume that the robot is a point moving in the plane amidst polygonal obstacles denoted by a set of oriented edges $\mathbf{e}_i$. We assume that the environment is perfectly known. The robot is equipped with a position sensor (dead reckoning), and a proximity sensor which allows to detect contact with obstacles and to follow walls (edges). The error on position depends on past history (trajectory) and results in cumulative uncertainty.

We assume that the robot initial and final positions are in the free space or along a known edge with an associated uncertainty represented either by a disk or by an interval. The planner will provide a plan which allows, at any moment to know if the robot is at an uncertain position $\mathbf{p}$:

- in the free space; $\mathbf{p} = (x, y, \epsilon)$, where x, y denote the coordinates and ϵ the value of the maximum error of the position sensor (dead reckoning)[2];
- in the contact space, on a known edge $\mathbf{e}_i$; $\mathbf{p} = (x, \epsilon)$ where x denotes the nominal position along $\mathbf{e}_i$ and ϵ the value of maximum error;
- or on a known obstacle vertex; the uncertainty is then set to zero[3].

3.2 The commands and their models

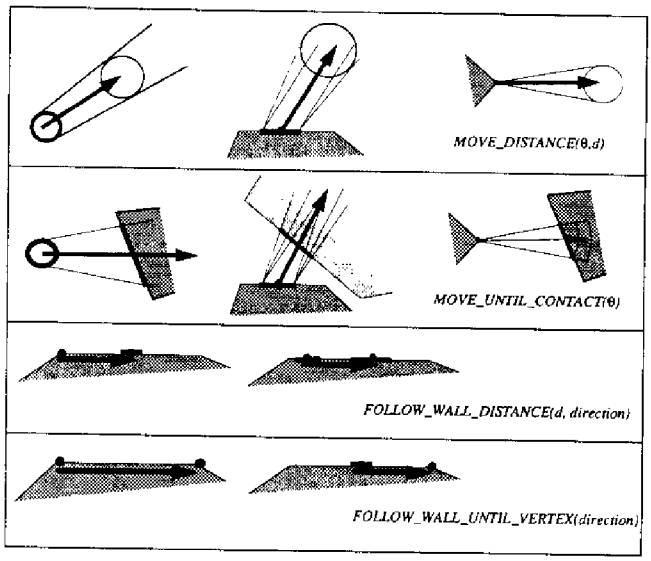


Figure 1: Model of the different primitives

[2] the disk centered on (x, y) with a radius equal to ϵ may overlap obstacles
[3] it is straightforward to extend it to a fixed value

There are several available motion primitives which are modelled by the type of initial situations where they can be applied, the final situation they can reach and the envelope of possible actual trajectories they can induce (Fig. 1):

- `MOVE_DISTANCE`(θ, d)
- `MOVE_UNTIL_CONTACT`(θ)
- `FOLLOW_WALL_DISTANCE`$(d,$ `[LEFT | RIGHT]`$)$
- `FOLLOW_WALL_UNTIL_VERTEX(`[`LEFT | RIGHT`]`)`

These primitives are based on dead-reckoning and contact (or proximity) sensors for primitives designed to maintain contact (or fixed distance) with an edge.

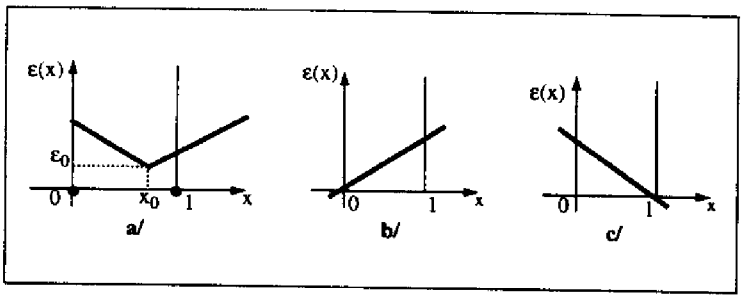


Figure 2: Evolution of the uncertainty along an edge. a/ after arriving onto the edge at x_0 with an uncertainty ϵ_0. b/ and c/ after a vertex relocalization

- When the robot is commanded to move in a direction θ (`MOVE_DISTANCE`, `MOVE_UNTIL_CONTACT`), it follows a path such that the tangent to the path at any point makes an angle with the direction θ which is smaller than a pre-specified value $\Delta\theta$ called the *angle of the control uncertainty cone.*
- When it reaches non ambiguously an edge (after a `MOVE_UNTIL_CONTACT`), the new uncertainty is modelled as an interval whose endpoints correspond to the intersection of the edge with the envelope of all possible trajectories.
- When the robot is commanded to follow an edge (`FOLLOW_WALL_DISTANCE`), we assume that the maximum value of the positioning error, increases linearly: $\epsilon(x) = a.\|x - x_0\| + \epsilon_0$ with $a \in [0, 1]$ (Fig. 2-a).
- When it reaches a vertex (after a `FOLLOW_WALL_UNTIL_VERTEX`), ϵ is reset to zero. If, from the reached vertex, the robot follows again the same edge (in the opposite direction) or an adjacent edge, using `FOLLOW_WALL_DISTANCE`, the maximum value of the positioning error will again increase linearly, but with an initial value equal to zero: $\epsilon(x) = a.x$ (Fig 2-b and 2-c).

This will allow the planner to produce, when necessary, re-localization strategies along a given edge. In order to ensure a coherence between the different commands, we take: $a = \tan(\Delta\theta)$. However, the algorithm we propose does not need such an assumption.

3.3 The planning problem

Given an initial and a goal position, in the free space or along a known edge together with their uncertainties (represented by a disk or by an interval), the planner should generate a sequence of motion commands whose execution guarantees to reach the goal with the desired uncertainty if the actual errors lie inside the control uncertainty cone $\Delta\theta$. We restrict the plan, except the initial and final steps, to contain a sequence of commands which allow the robot to follow edges or to reach an edge starting from another one.

In order to implement such a planner, we need a basic step which computes if a given entity (a disk or an edge) can be reached from another entity using the available commands. If a command exists, the entities will be said *adjacent*. Informally, a target entity ent_j can be reached from an uncertain position $\mathbf{p}$, if there exists at least one direction θ such that, taking into account a given value of the uncertainty cone $\Delta\theta$:

- **C1:** the robot cannot miss ent_j. That is, any path lying inside the uncertainty cone issued from any initial position contained in $\mathbf{p}$, intersects ent_j. When this condition is satisfied, the target entity ent_j is said to be *Strongly visible* from the uncertain position $\mathbf{p}$;
- **C2:** the robot cannot collide any other entity ent_k before the target entity ent_j is reached (i.e. none of these paths intersects ent_k before the intersection with ent_j). When this condition is not verified for some ent_k, this entity is said to be *Weakly visible* from the uncertain position $\mathbf{p}$.

In the next sections, we develop the computation of the *disk-edge* and *edge-edge* adjacencies. While the first is quite simple (§4), the second gives rise to the possibility of characterizing the adjacency from a *complete* edge and not only from an uncertain position on an edge (§5). It will provide us with a coherent way to combine and to compute the parameters of the motion commands.

4 Disk-Edge Adjacency

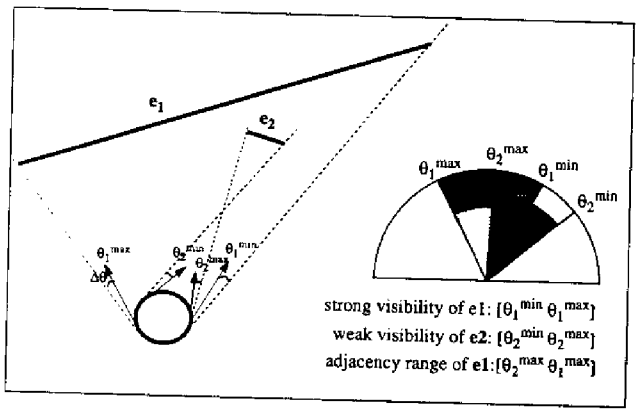


Figure 3: θ-ranges associated to the disk-edge adjacency

This adjacency can be easily computed as illustrated by Fig. 3. The strong and weak visibility of each edge are satisfied for only one range of θ values. These ranges are determined using the tangents to the disk going through the edges endpoints. The adjacency between the disk associated to an uncertain position $\mathbf{p}$ and a given edge is characterized by the existence of a non empty set of *adjacency ranges*. This set of θ-ranges is obtained from the set difference between the strong visibility range computed for this edge and the weak visibility ranges computed for the other edges lying between the disk and the target edge. Figure 4 shows an example of workspace and the adjacencies computed by the algorithm.

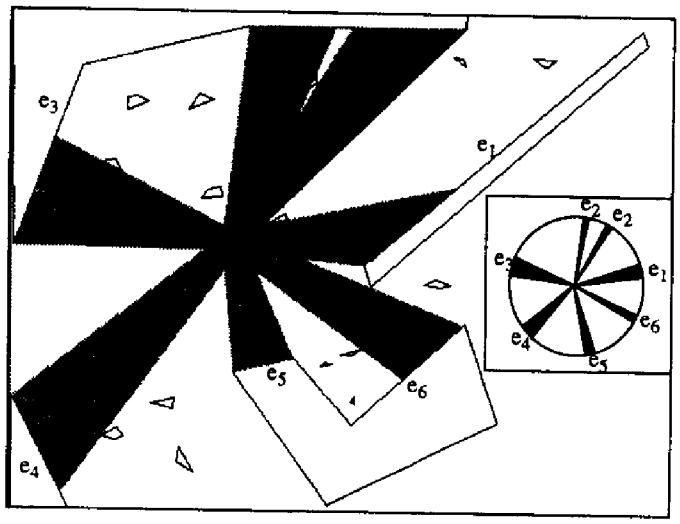


Figure 4: An example of disk/edge adjacency

5 Edge-Edge adjacency

Let us consider that the robot arrived on a given edge $\mathbf{e}_i$ at position (x_0, ϵ_0). We want to characterize the parameters of the motion commands which allow to reliably reach the other edges of the environment from this position. Some of these edges can be directly reached with a `MOVE_UNTIL_CONTACT` using a computation similar to the case described in §4. However there can also be other edges which are not directly "visible" from the current position but which can be reached if the robot first executes a `FOLLOW_WALL_DISTANCE` along $\mathbf{e}_i$ from x_0 to another position x. In some other cases, the additional uncertainty induced by `FOLLOW_WALL_DISTANCE` will be too important and a re-localization (using `FOLLOW_UNTIL_VERTEX`) will be required. In this section, we first introduce a notion of visibility between edges; then we explain how this notion is used to decompose the (x, θ)-space (which parametrizes the set of motion commands) into regions where the reachability of specific edges is guaranteed. These regions allow to determine the adjacencies between the edges of the environment.

5.1 Visibility regions

An edge $\mathbf{e}_j$ is said to be *strongly* (resp. *weakly*) visible from a position x along $\mathbf{e}_i$ and for a commanded motion in direction θ, if for **any** (resp. **some**) uncertain position $\bar{x}$ of the interval $[x - \epsilon(x), x + \epsilon(x)]$, the straight line issued from $\bar{x}$ in direction θ intersects the edge $\mathbf{e}_j$.

Note that this definition of the strong and weak visibility slightly differs from the one expressed in section 3.3 by the conditions **C1** and **C2** since we do not consider here the uncertainty cone $\Delta\theta$. As explained later (see section 5.2), this allows a simpler way to construct the adjacency regions between the edges of the environment.

The regions of the (x, θ)-space which verify this condition are called the **strong** (resp. **weak**) **visibility** regions from the edge $\mathbf{e}_i$ to the edge $\mathbf{e}_j$. These visibility regions, respectively denoted $\mathcal{SV}_j$ and $\mathcal{WV}_j$, are limited by two curves $\theta_{min}(x)$ and $\theta_{max}(x)$. For a linear evolution of $\epsilon(x)$, the tangents of these extremal orientations also depend linearly on x. Thus, $\mathcal{SV}_j$ and $\mathcal{WV}_j$ can be simply represented in the $(x, \tan(\theta))$-space by trapezoidal regions. We detail below how these regions can be computed.

5.1.1 Strong visibility regions

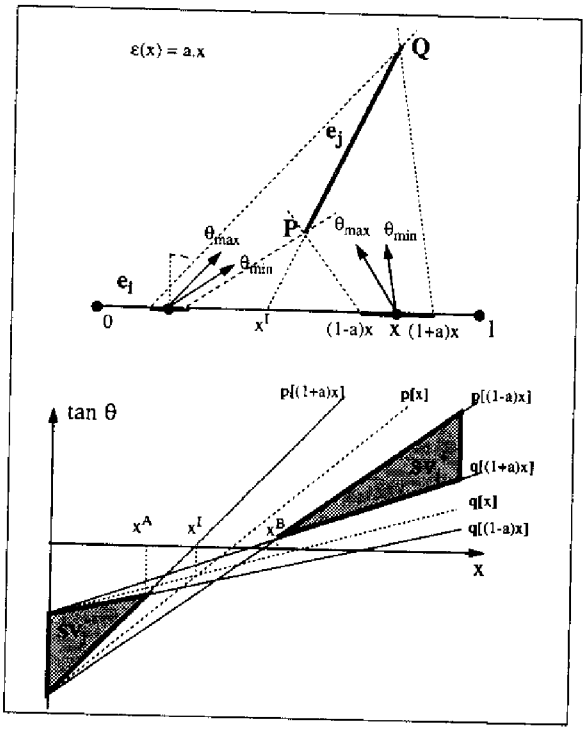


Figure 5: Strong visibility regions

Figure 5 illustrates the construction of SV_j for an error [4] modelled by $\epsilon(x) = a.x$. Thus, for a given nominal position x, the robot can be anywhere inside the interval $U_a(x) = [(1\text{-}a)x, (1+a)x]$. As shown by the figure, two cases need to be considered depending on the location of the interval $U_a(x)$ relatively to the intersection x^I between the edge e_i and the support line of e_j.

- When $U_a(x)$ is located on the left of x^I (ie. $x \in [0, x^I/(1+a)]$), the minimal value $\theta_{min}(x)$ (resp. $\theta_{max}(x)$) corresponds to the direction of the half-line starting at position $(1+a)x$ (resp. $(1\text{-}a)x$) and going through the endpoint P (resp. Q). The tangents of these extremal orientations are therefore defined by:

$$t_{min}(x) = p[(1+a)x] \text{ and } t_{max}(x) = q[(1\text{-}a)x]$$

  where $p[x] = (x - x^P)/y^P$ and $q[x] = (x - x^Q)/y^Q$.
  However, only the x values such that $t_{min}(x) < t_{max}(x)$, belong to the strong visibility region $\mathcal{SV}_j$. It can be easily shown that this inequality is only verified for $x \in [0, x^A]$ where $x^A \leq x^I/(1+a)$. We call $\mathcal{SV}_j^-$, the region defined on this interval by the two linear functions $t_{min}(x)$ and $t_{max}(x)$.

- Similarily, when $U_a(x)$ is located on the right of x^I (ie. $x \in [x^I/(1\text{-}a), l]$), the same reasoning yields:

$$t_{min}(x) = q[(1+a)x] \text{ and } t_{max}(x) = p[(1\text{-}a)x]$$

  The condition $t_{min}(x) < t_{max}(x)$ is only verified for $x \in [x^B, l]$ with $x^B \geq x^I/(1\text{-}a)$. We call $\mathcal{SV}_j^+$ the associated region.

When x^I belongs to $U_a(x)$ (ie. $x \in [x^I/(1+a), x^I/(1\text{-}a)]$), the set of orientations verifying the definition of the strong visibility region, is clearly empty. Therefore $\mathcal{SV}_j$ is the union of the two[5] disjoint regions $\mathcal{SV}_j^-$ and $\mathcal{SV}_j^+$.

5.1.2 Weak visibility regions

As illustrated by Fig. 6, the weak visibility region $\mathcal{WV}_j$ consists of the three following connected regions:

- $\mathcal{WV}_j^-$ is defined for $x \in [0, x^I/(1+a)]$ by the region lying between the functions:

$$t_{min}(x) = p[(1\text{-}a)x] \text{ and } t_{max}(x) = q[(1+a)x]$$

- $\mathcal{WV}_j^0$ is defined for $x \in [x^I/(1+a), x^I/(1\text{-}a)]$ by the region lying between the functions:

$$t_{min}(x) = p[(1\text{-}a)x] \text{ and } t_{max}(x) = p[(1+a)x]$$

[4] the extension to the general case $\epsilon(x) = a.\|x - x_0\| + \epsilon_0$ is straightforward.
[5] When x^I lies outside e_i only one of the these regions is defined.

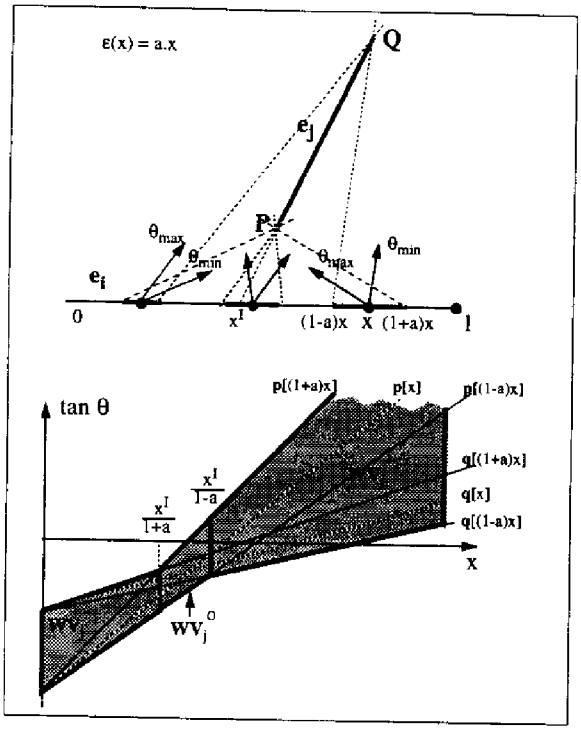


Figure 6: Weak visibility regions

- $\mathcal{WV}_j^+$ is defined for $x \in [x^I/(1\text{-}a), l]$ by the region lying between the functions:

$$t_{min}(x) = q[(1\text{-}a)x] \text{ and } t_{max}(x) = p[(1\text{+}a)x]$$

Figure 6 shows the weak visibility regions in the $(x, \tan(\theta))$-space. Note that we have presented the general case for which x^I lies on the edge $\mathbf{e}_i$. For the cases such that x^I is exterior to $\mathbf{e}_i$, $\mathcal{WV}_j$ consists of less than three regions.

5.2 Adjacency regions

The **adjacency regions** $\mathcal{A}_j$ from $\mathbf{e}_i$ to $\mathbf{e}_j$, consist of the (x, θ) values such that the motions issued from **any** uncertain position $x \pm \epsilon(x)$ along $\mathbf{e}_i$, and in **any** direction $\theta \pm \Delta\theta$ are guaranteed to endup on $\mathbf{e}_j$; that is, the robot will not miss $\mathbf{e}_j$ and it will not be stopped on its way by another edge. We describe below the two steps of the algorithm we implemented in order to compute the $\mathcal{A}_j$ regions.

The first step is performed without considering the directional uncertainty $\Delta\theta$. In this case, $\mathcal{A}_j$ clearly corresponds to the sub-regions of $\mathcal{SV}_j$ which are not contained into the union of the $\mathcal{WV}_k$ regions associated to the edges lying between $\mathbf{e}_i$ and $\mathbf{e}_j$. $\mathcal{A}_j$ is first initialized to $\mathcal{SV}_j$; then it is iteratively

decomposed by considering each of the $\mathcal{WV}_k$. The basic operation of each iteration consists in computing the set difference between two regions. This computation can be easily performed by considering their representations in the $(x, \tan(\theta))$-space; it results in decomposing each trapezoid into at most four trapezoidal regions.

The second step is needed to account for $\Delta\theta$. Hence, the adjacency regions $\mathcal{A}_j$ only contain the points (x, θ) such that $(x, \tan(\theta \pm \Delta\theta)$ belongs to the trapezoids produced by the first step. Using the equations of the upper and lower lines defining each trapezoid, we derive the expression of the curves $\theta_{min}(x)$ and $\theta_{max}(x)$ which limit the corresponding sub-region of $\mathcal{A}_j$.

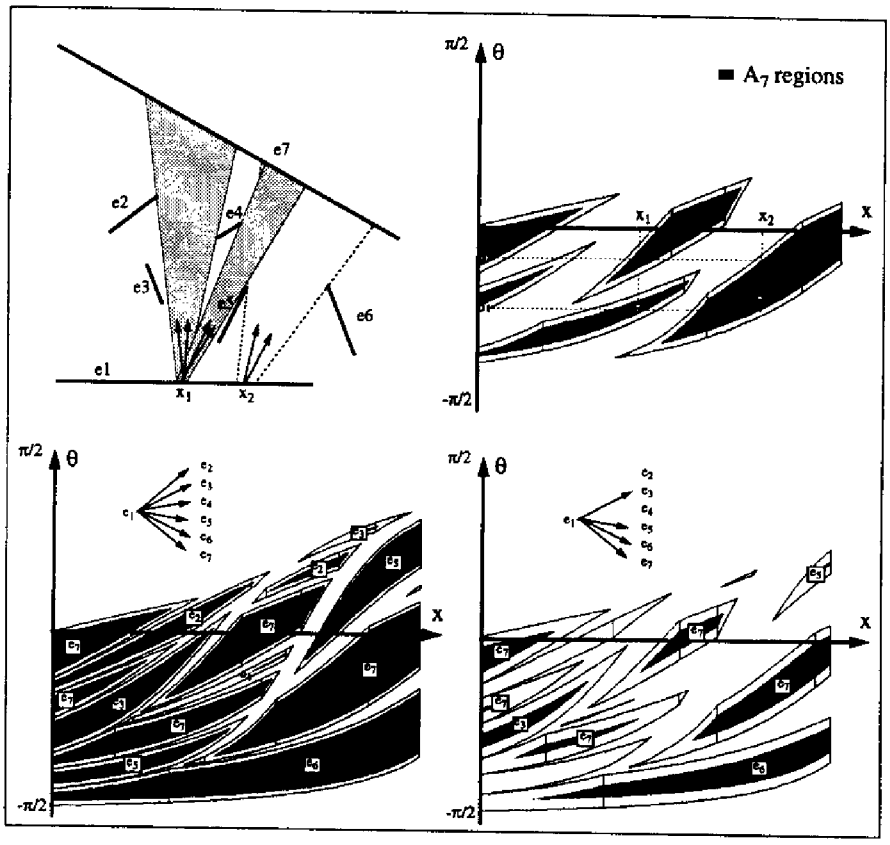


Figure 7: Example of adjacency regions from e_1 to e_7

Figure 7 shows a simple environment and the adjacency regions associated to the edge e_1. Only the regions $\mathcal{A}_7$ computed from e_1 to the edge e_7 are represented on Figure 7-b. For a given a position x of the robot, the different angular ranges guaranteed to reach the target edge e_7 can be simply obtained from these regions; as illustrated on the Figure, while two ranges are possible from position x_1, only one range remains valid when the robot reaches x_2 (with

a larger positional uncertainty). The lower Figures (Fig. 7-c and 7-d) show all the adjacency regions computed for e_1 and illustrate the influence of the control uncertainty $\Delta\theta$ on the edge adjacency. For example, when $\Delta\theta$ is set to 3 deg. (Fig. 7-c), all the edges are adjacent to e_1; the adjacency to the edges e_2 and e_4 disappears (Fig. 7-d) when the control uncertainty is increased to 6 degrees.

6 The Planner

The planner is given a set of polygonal obstacles, a value for the angle uncertainty cone $\Delta\theta$, the initial and final positions represented by two discs C_I and C_G. The search space is a directed graph G of uncertain positions $\mathbf{p} = (x, y, \epsilon)$. The nodes are the initial and final positions $\mathbf{p}_I$ and $\mathbf{p}_G$ and a discrete set of positions lying on the edges of the environment. An arc between two nodes corresponds to a given motion command as defined in §3.2.

The planner initially constructs a graph G containing only the two nodes associated to $\mathbf{p}_I$ and $\mathbf{p}_G$. In a first step, it simply tries to connect them by checking that the half-lines D_1 and D_2 (see Fig. 8) issued from $\mathbf{p}_I$ do not intersect any edge e_i and both intersect the circular region C_G. In this case, the single motion command `MOVE_DISTANCE`(θ_{IG}, d_{IG}) issued in direction θ_{IG} and terminating after a nominal distance d_{IG} is guaranteed to end somewhere inside the goal region.

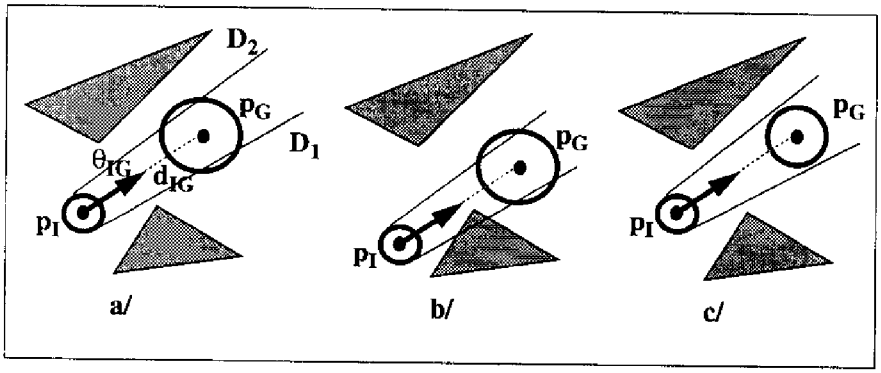


Figure 8: a/ $\mathbf{p}_G$ and $\mathbf{p}_I$ are connected. b/ and c/ $\mathbf{p}_G$ and $\mathbf{p}_I$ cannot be connected (potential collision with the environment or final error too large)

If not, the initial node is expanded as illustrated by Fig. 9 in order to generate a discrete set of positions lying on the "visible" edges of the environment. These successors are created for the extremal values of the legal θ-ranges. The arcs issued from $\mathbf{p}_I$ correspond to a `MOVE_UNTIL_CONTACT` motion command.

The other nodes of the graph are incrementally created during the search of a minimum-cost path connecting $\mathbf{p}_I$ to $\mathbf{p}_G$. The costs assigned to the arcs are simply computed from the length of the nominal path connecting two adjacent nodes, and the search is carried out by a classical A^* algorithm. Each iteration of the algorithm consists in expanding the "best" node of the $OPEN$ list. We now describe the expansion of a current node denoted $\mathbf{p}_{cur}$ lying on an edge e_{cur}. If $\mathbf{p}_{cur}$ can be directly connected to $\mathbf{p}_G$, the expansion simply consists in linking both nodes with a `MOVE_DISTANCE` arc and the planner returns success.

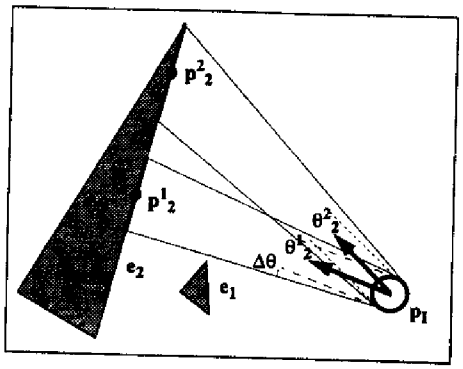


Figure 9: Expansion of the initial node: given the positional uncertainty ϵ_I and the control uncertainty $\Delta\theta$, only the edge e_1 is "visible" from p_I. Two successors p^1_1 and p^2_1 are generated for the extremal values θ^1_1 and θ^2_1 of the adjacency orientation range

Otherwise, the successors are generated as follows: p_{cur} is first connected with FOLLOW_UNTIL_VERTEX motions to the two endpoints of edge e_{cur}. As the robot is assumed to perfectly detect a corner of the environment, a null uncertainty is associated to both corresponding nodes (p_{left} and p_{right} for the example of Fig. 10 which illustrates the node expansion mechanism).

For all the other edges e_i, the adjacency regions computed from e_{cur} to e_i are used to generate motion commands guaranteed to reach the edge e_i. Three cases can occur:

- The set of adjacency regions $\mathcal{A}_i$ is empty (ie. the edge is too short to be reachable from e_{cur} with the current uncertainty or it is completely obstructed by other edges): no successors are associated to this edge. This is the case of edge e_1 on the example.
- The location p_{cur} along the edge e_{cur} is such that one (or more) feasible range of orientations exists in the $\mathcal{A}_i$'s regions (case illustrated by the $\mathcal{A}_2$ region shown in Fig. 10-b). A successor is then created for each of the two extremal orientations of the ranges (θ^1_2 and θ^2_2 for the example). These new nodes are connected to p_{cur} by MOVE_UNTIL_CONTACT commands.
- When the vertical line issued in the (x, θ)-space at the current x position does not intersect any cells of the $\mathcal{A}_i$ regions (case illustrated by the $\mathcal{A}_3$ regions of Fig. 10-b associated to the edge e_3), the robot first needs to follow e_{cur} toward the left (resp. the right) until it reaches a position denoted p' (resp. p'') on the figure, and corresponding to the beginning of the nearest legal range. From this intermediate position p' (resp. p'') reached via a FOLLOW_DISTANCE command, a MOVE_UNTIL_CONTACT motion command issued in direction θ^1_3 (resp. θ^2_3) allows to generate the successor p^1_3 (resp. p^2_3) on edge e_3.

Figure 10-c shows the successors created for the environment of Fig. 10-a; it also indicates the motion commands produced to connect p_{cur} to these nodes.

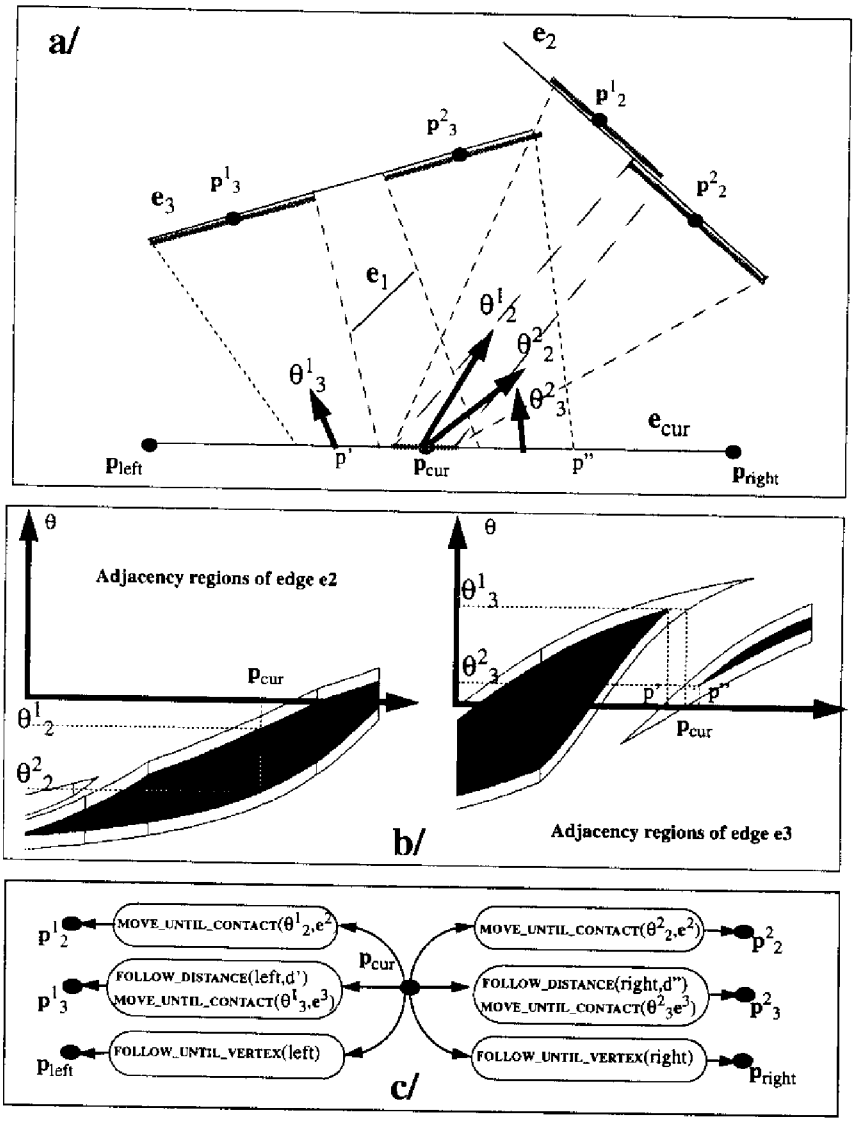


Figure 10: Node expansion mechanism

7 Experimental Results

The algorithms presented above have been implemented in C on a Sun Sparc 10 computer. We ran the planner on several workspaces, with various sizes of the initial and goal regions (ϵ_I and ϵ_G) and with different values of the directional uncertainty $\Delta\theta$.

Figures 11, 12 and 13 illustrate some results obtained for a same polygonal workspace. The preprocessing of the adjacency regions between all the 98 edges of this workspace required 6 seconds on a Sparc 10 workstation.

The black polyline joining the initial and goal regions represent the nominal trajectory followed by the motion primitives produced by the planner. The grey regions show the areas that may be swept by the robot during the execution of these primitives because of the uncertainties.

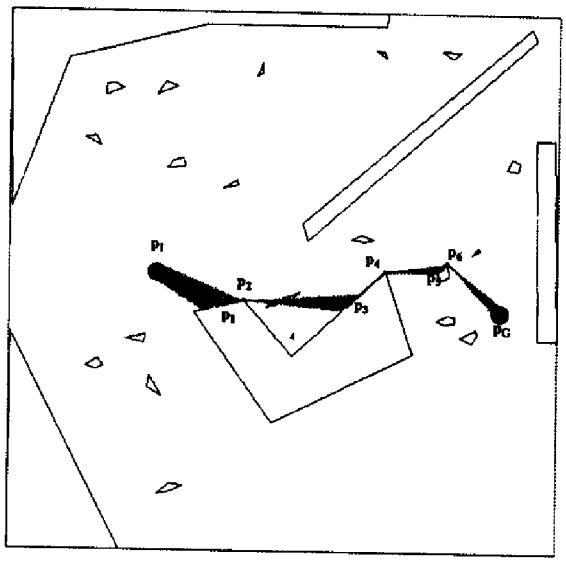


Figure 11: Solution obtained for small initial and control uncertainty

The first example (Fig. 11) was run for small values of both initial and control uncertainties ($\Delta\theta = 5deg.$). Note that the positional uncertainty is reset at the end of the `FOLLOW_UNTIL_VERTEX` commands (positions $\mathbf{p}_2$, $\mathbf{p}_4$ and $\mathbf{p}_6$) because of the vertex based re-localization. Only 35 nodes were developed during the search and the solution was obtained in half a second after the preprocessing step.

The second example was run with the same value of $\Delta\theta$ but with a larger radius for the initial disk. In this case, the initial uncertainty is too important to directly reach the polygon displayed at the center of the figure. After a first motion guaranteed to stop on the left-down wall of the workspace, and a precise vertex localization at position $\mathbf{p}_2$, this large polygon can be reliably reached with the next `MOVE_UNTIL_CONTACT` command (position $\mathbf{p}_3$).

Finally, the third example was run with $\Delta\theta$ set to 10 degrees. The small obstacles around the goal region cannot be used anymore to localize the robot

and a completely different motion strategy is required as illustrated by Fig. 13.

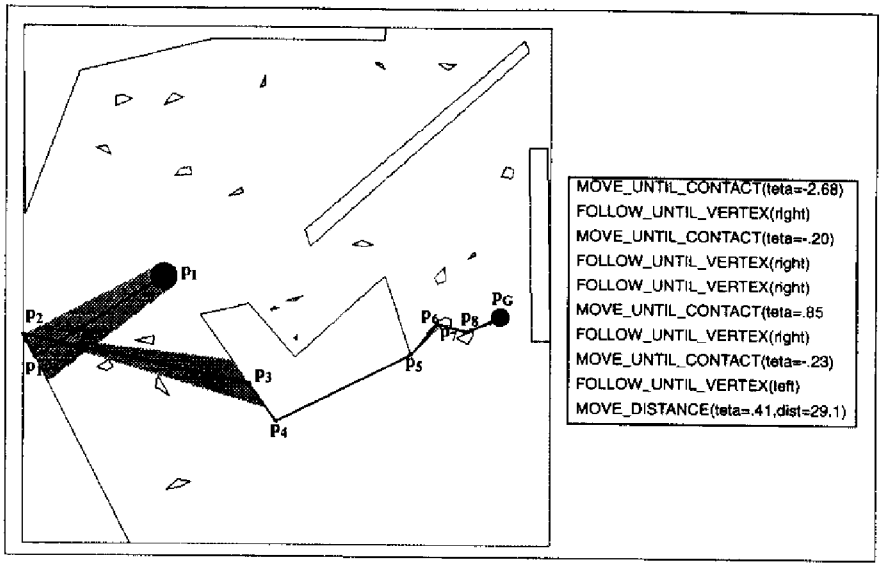


Figure 12: Same example with a larger initial uncertainty

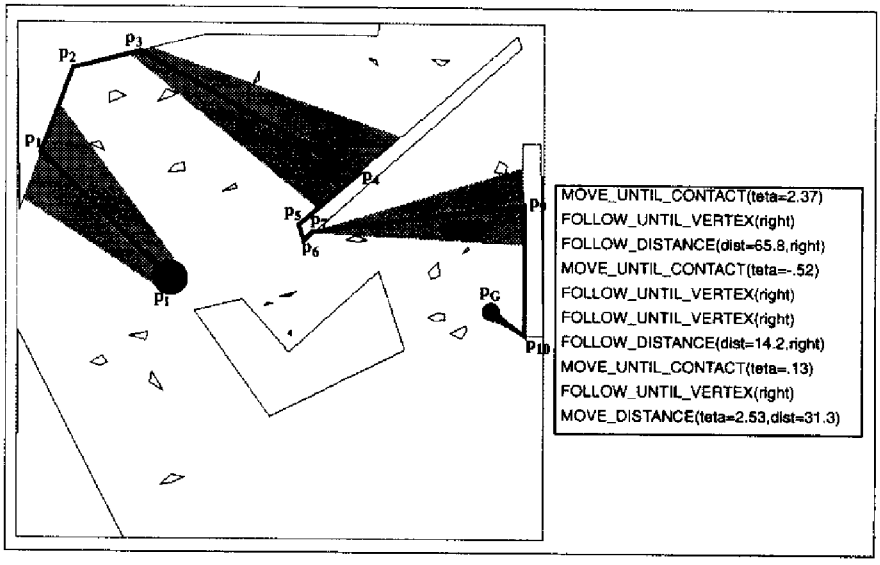


Figure 13: Same example with a larger control uncertainty

Note that the solution produced by the planner contains two FOLLOW_DISTANCE commands between $\mathbf{p}_2$ (resp. $\mathbf{p}_6$) and $\mathbf{p}_3$ (resp. $\mathbf{p}_7$). In both cases, the edge containing $\mathbf{p}_4$ (resp. $\mathbf{p}_9$) could not be reached directly from position $\mathbf{p}_2$ (resp. $\mathbf{p}_6$) and the planner used the adjacency regions between the corresponding edges to compute the first position at which the next MOVE_UNTIL_CONTACT command was guaranteed to succeed.

For the two last examples, 46 and 78 nodes were respectively developed during the search and the solution was obtained in both cases after less than one second of computation.

8 Discussion and Extensions

The main originality of the approach described above is its ability to produce robust motion plans composed of sensor-based motion commands which may accumulate errors, and to determine where the robot needs to re-localize in order to guarantee a successful execution of the trajectory.

Although the edge-edge adjacency described in § 5 captures exactly all the guaranteed motions between two given edges of the workspace, the planner described in § 6 is not complete since the number of paths explored is a finite set (discrete points (x, θ) selected into the computed adjacency regions). Moreover, the current version of the planner does not allow to generate inter-

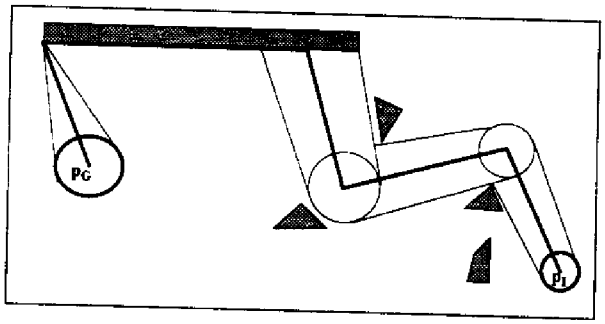


Figure 14: Solution not found by the planner

mediate positions in free-space. Therefore, there possibly exist situations where some obstacles are too small to be reached by a MOVE_UNTIL_CONTACT command, but could be avoided by a sequence of MOVE_DISTANCE commands, allowing to reach the goal (or an intermediate edge) with a sufficient precision. Figure 14 illustrates such a situation where the planner fails to find a solution [6].

However, the planner becomes complete if we restrict the class of solutions to strategies where a vertex re-localization is systematically performed after the MOVE_UNTIL_CONTACT commands (the errors do not depend anymore on the complete past history). Note that in this case, and when the error $\Delta\theta$ tends to zero, the search space of the algorithm degenerates to a classical visibility graph.

[6] In [2], we propose a different algorithm based on the propagation of a numerical potential which allows, whenever possible to navigate without localizing the robot when the task does not impose it.

The algorithm described in this paper solves a relatively simple instance of the motion planning problem in presence of uncertainty. We are currently investigating several extensions of the approach. Some of them are rather straightforward; the algorithms can be easily adapted to deal with a circular robot and to relax the assumption of a perfect proximity sensor. Another interesting extension would be to consider the information which can be provided by other types of sensors (distance to the obstacle, orientation of the edges) and to explore the possibility of producing conditional strategies.

Nevertheless, there are numerous situations where it should be enough to use a planner like the one we described, which is really able to find in a very reasonable time, non-trivial sensor-based motions strategies for a mobile robot equipped with proximity sensors.

Acknowledgment

This work was partially supported by the CEC Esprit 3 BRA Project 6546 PROMotion.

References

[1] R. Alami and T. Siméon. Planning robust motion stategies for a mobile robot. In *IEEE ICRA'94*, San Diego (USA).

[2] B. Bouilly, T. Siméon and R. Alami. A numerical technique for planning motion strategies for a mobile robot in presence of uncertainties. In *IEEE ICRA'95*, Nagoya (Japan).

[3] R.A Brooks. Symbolic error analysis and robot planning. *Int. Journal of Robotics Research*, 1(4), 1982.

[4] R. Brost. Dynamic analysis of planar manipulation tasks. In *IEEE ICRA'92*, Nice (France).

[5] J. Canny. On computability of fine motion plans. In *IEEE ICRA'89*, Scottsdale (USA).

[6] I. Collin, D. Meizel, N. Le Fort and G. Govaert. Local map design and task function planning for mobile robots. In *IROS'94*, Munich (Germany).

[7] J. Najera, F. De la Rosa and C. Laugier. Planning robot motion strategies under geometric constraints. In *IROS'94*, Munich (Germany).

[8] B.R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37:223–271, 1988.

[9] B.R. Donald, J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. TR 91-1248, Cornell University, Dec 1991.

[10] M. Erdmann. Using backprojections for fine motion planning with uncertainty. *Int. Journal for Robotics Research*, 5(1):19–45, Spring 1986.

[11] A. Fox, S. Hutchinson. Exploiting visual constraints in the synthesis of uncertainty-tolerant motion plans (parts I & II) In *IEEE ICRA'93*, Atlanta.

[12] K. Goldberg, M. Mason, A. Requicha. Geometric uncertainty in motion planning: Summary report and bibliography. Technical report IRIS TR 297, USC Los Angeles, Aug 1992.

[13] S. N. Gottschlich, A. C. Kak. AMP-CAD: An assembly motion planning system. In *IEEE ICRA'92*, Nice.

[14] J.-C. Latombe. Robot Motion Planning. *Kluwer Academic Publishers*, 1991.

[15] J-C. Latombe, A. Lazanas, S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1):1–47, Nov 1991.

[16] A. Lazanas, J-C. Latombe. Landmark-based robot navigation. to appear in *Algorithmica*. Technical Report STAN-CS-92-1428, Stanford University, May 1992.

[17] T. Lozano-Perez. The design of a mechanical assembly system. A.I Memo 397, MIT Artificial Intelligence Lab., MIT, Cambridge, Dec 1976.

[18] T. Lozano-Perez. Robot programming. A.I Memo 698, MIT Artificial Intelligence Lab, Dec 1982.

[19] T. Lozano-Perez, M.T. Mason, R.H. Taylor. Automatic synthesis of fine motion strategies for robots. *Int. Journal of Robotics Research*, 3(1), 1984.

[20] I. Mazon, R. Alami. Representation and propagation of positionning uncertainties through manipulation robot programs.. In *IEEE ICRA'89* , Scottsdale.

[21] J. Troccaz, P. Puget. Dealing with uncertainty in robot planning using program proving techniques. *The Fourt ISRR*, Santa Cruz (USA), March 1987.

[22] H. Takeda, J-C. Latombe. Sensory uncertainty field for robot navigation. In *IEEE ICRA'92*, Nice.

[23] R.H. Taylor. Synthesis of manipulator control programs from task-level specifications. AIM 228, AI Laboratory, Stanford, July 1976.

[24] R.H. Taylor, V.T. Rajan. The Efficient Computation of Uncertainty Spaces for Sensor-based Robot Planning. In *IEEE IROS'88*, Tokyo (Japan)

J.5 Coopération multi-robot

13 *Multi-robot Cooperation through Incremental Plan-Merging.*

IEEE International Conference on Intelligent Robots and Systems (*ICRA '95*), Nagoya (Japon), Mai 1995.

Auteurs: R. Alami, F. Robert, F. F. Ingrand, S. Suzuki

14 *Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment*

IEEE *IROS'95*, Pittsburgh (USA) 1995.

Auteurs: L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert.

15 *A Multi-Robot Cooperation Scheme Based on Incremental Plan-Merging*

In, R. Hirzinger and G. Giralt (Eds), *Robotics Research: The Seventh International Symposium*, Springer Verlag, 1996 (à paraitre)

Auteurs: R. Alami

Multi-robot Cooperation through Incremental Plan-Merging*

Rachid ALAMI, Frédéric ROBERT, Félix INGRAND, Sho'ji SUZUKI
LAAS / CNRS
7, Avenue du Colonel Roche
31077 Toulouse - France

Abstract: This paper presents an approach we have recently developed for multi-robot cooperation. It is based on a paradigm where robots incrementally merge their plans into a set of already coordinated plans. This is done through exchange of information about their current state and their future actions. This leads to a generic framework which can be applied to a variety of tasks and applications. The paradigm, called *Plan-Merging Paradigm* is presented and illustrated through its application to planning, execution and control of a large fleet of a autonomous mobile robots for load transport tasks in a structured environment.

1 Introduction

We present, in this paper, an approach we have recently developed for multi-robot cooperation. It is based on a paradigm, called *Plan-Merging Paradigm*, where robots incrementally merge their plans into a set of already coordinated plans. This is done through exchange of information about their current state and their future actions. This paradigm leads to a generic framework which can be applied to a variety of tasks and applications.

In section 2 we define the framework where multi-robot cooperation takes place. In section 3 we briefly present and discuss the proposed paradigm. Section 4 discusses a typical application: a fleet of autonomous mobile robots navigating in a route network. We then discuss implementation issues and related work (section 5).

2 A framework for cooperation

2.1 Problem statement

Let us assume that we have a set of autonomous robots and a central system which, from time to time, assigns and sends goals to robots individually.

Whenever it receives a goal, a robot is assumed to elaborate and execute a plan which achieves it. Goals may be sent asynchronously to the robots even if they are still processing a goal previously sent.

Each robot processes sequentially the goals it receives, taking as initial state the final state of its current plan. Doing so, it incrementally appends new sequences of actions to its current plan.

However, before executing any plan step, a robot has to ensure that it is valid in the current multi-robot context, i.e. that it is compatible with all the plans currently under execution by the other robots. This will be done *without modifying* the other robots plans, in order to allow the other robots to continue execution.

We call this operation, a *Plan-Merging Operation* (PMO) and its result a *Coordination Plan* (i.e. a plan valid in the current multi-robot context).

Planning, plan merging and execution may run in parallel. In fact, considering the time needed for planning and plan merging operation, and considering the average range of the obtained coordination plans, execution run most often without waiting for a new coordination plan.

2.2 The "global plan" and its properties

Everything works as if there was a *global plan* produced and maintained by the set of robots. In fact, neither the robots nor the central station elaborate, store and maintain such a *global plan*[1].

At any moment, a robot has its own *coordination plan* under execution. Such a *coordination plan* consists of a sequence of actions and events to be signaled to other robots as well as events which are planned to be signaled by other robots. Such events correspond to state changes in the multi-robot context and represent temporal constraints (precedence) between actions involved in different individual coordination plans.

At any moment, the "global plan" is the graph representing the union of all current robot coordination plans. Such a global plan is valid (i.e. it does not contain inconsistent temporal constraint) if it can be represented by a *directed acyclic graph* (*dag*).

The key point here is how to devise a system composed of a set of robots which should, as much as possible, plan independently to achieve their tasks while maintaining such property of the global plan.

*This paper has been submitted to 1995 IEEE International Conference on Robotics and Automation, Nagoya, Japan.

[1]Note that the central station maintains a higher level description of the set of missions allocated to the robots. However it does not need to know the plans elaborated by the robots to achieve their missions and how these plans are coordinated.

13.1

3 The Plan-Merging Paradigm

Let us assume here that:

1. there exists a mean which allows a robot to get the right to perform a PMO while having the guarantee that it is the only robot doing so. This right should be thought of as a resource allocation[2].
2. there is a mean allowing a robot (which has obtained the right to perform a PMO) to ask for and obtain all the other robots coordination plans.
3. there exists a mean allowing robots to ask or inform one another about the occurrence of an event.

3.1 Performing a *Plan Merging Operation*

Robots are assumed to plan from time to time (whenever it is necessary).

When a robot is not planning and even when it is waiting to obtain the right to perform a PMO, it must be able to send its current coordination plan to another robot (which currently has the right to perform a PMO).

When a robot has to plan, it uses the following protocol which we call the *Plan Merging Protocol* (see Figure 1):

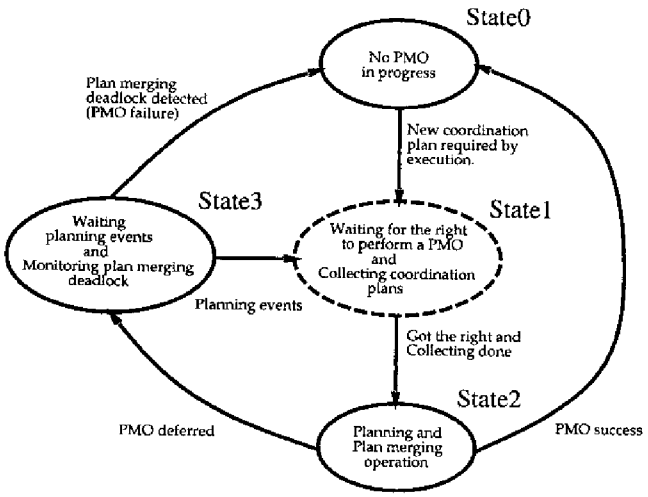


Figure 1: The general protocol state graph

1. It asks for the right to perform a PMO and waits until it obtains it together with the coordination plans of all the other robots.
2. It then builds the *dag* corresponding to the union of all coordination plans (including its own coordination plan)
3. It then tries to produce a new plan which can be inserted in the *dag*, *after* its current coordination plan. The new plan insertion may only add temporal constraints which impose that some of its actions must be executed after some time-points from other robots coordination plans.
   Besides, the insertion must maintain the fact that the obtained global plan is still a *dag*.
4. If it succeeds in producing the desired plan, the robot appends it to its current coordination plan.
5. And finally, it releases the right to perform a PMO.

When a robot executes its coordination plan, if it reaches a step with a temporal constraint linked to another robot time-point, it asks that robot if it has passed that time-point or not. Depending on the answer, the robot will wait until the other robot informs it or will immediately proceed.

3.2 Situations where PMO is deferred or where deadlock is detected

When a robot tries to perform a PMO, it may fail to produce a plan which satisfies the properties discussed earlier.

This may happen in two situations:

1. the goal can never be achieved. This can be detected if the robot cannot produce a plan even if it were alone in the environment. The robot informs the station and waits for a new goal.
2. the robot can generate a plan but this plan cannot be inserted in the global plan. This means that the final state of another robot forbids it to insert its own plan.
   In such situation, the robot can simply abandon the PMO and decide to wait until the robots, that it has identified, have performed a new PMO which may possibly make them change the states preventing it to insert its plan.

Hence, we have introduced two types of events:

1. *execution events:* i.e. events which occur during plan execution and which allow robots to synchronize their execution.
2. *planning events:* i.e. events which occur whenever a robot performs a new PMO. These events can also be awaited for.

Note that, even when a robot fails in its PMO, it leaves the global plan in a correct state (it is still a *dag* and its execution can continue).

In order to detect deadlocks, a robot which finds itself in a situation where it has to wait for a *planning event* from a particular robot, must inform it. Then, it becomes possible for a robot to monitor and detect *deadlock situations* by propagating and updating, the list of robots waiting (directly or by transitivity) for

[2] A simple way to do it is to maintain a "token" through communication; but this is not always desirable nor even possible (see section 4.2).

13.2

planning events from itself. Indeed, a deadlock is detected when a robot finds itself in the list of robots waiting for itself.

When a deadlock occurs, it is necessary to take explicitly into account, in a *unique planning operation*, a conjunction of goals (which have been given separately to several robots).

This simply means that the global mission was too constrained to be solved using the Plan-Merging Paradigm. It is then the responsibility of the central station to produce a multi-robot plan.

Here we must recall that we do not claim that the Plan-Merging paradigm can solve or help to solve multi-robot planning problems. The main point here is that the Plan-Merging paradigm is *safe* as it includes the detection of the deadlocks.

Note also that, in the case where only a small number of robots are involved in a deadlock, one can decide to allow the robot, which detected the deadlock, to plan for all the concerned robots . The Plan-Merging paradigm remains then applicable: the inserted plan will then concern several robots at a time.

A detailed discussion on the properties of the Plan-merging paradigm as well as on its ability to cope with execution failures can be found in [2].

3.3 Discussion

The paradigm and the protocol presented so far is generic. We believe that it can be used in numerous applications. Several instances of the general paradigm can be derived, based on different planners: action planners in the stream of STRIPS, as well as more specific task planners or motion planners.

One class of applications which seems particularly well suited is the control of a large number of robots in a route network.

We present in the sequel an application in the case of a fleet (dozens) of autonomous mobile robots. The use of the Plan-Merging paradigm allowed us to deal with several types of conflicts in a general and systematic way.

4 A fleet of autonomous mobile robots

We have applied the Plan-Merging Paradigm in the framework of the MARTHA project[3] which deals with the control of a large fleet of autonomous mobile robots for the transportation of containers in harbors, airports and railway environments.

In such context, the dynamics of the environment, the impossibility to correctly estimate the duration of actions (the robots may be slowed down due to obstacle avoidance, and delays in load and un-load operations, etc..) prevent a central system from elaborating efficient and reliable detailed robot plans.

The Plan-Merging paradigm is well suited to such applications where conflicts are local and involve a limited number of robots. Indeed, its use allowed us to limit the role of the central system to the assignment of tasks and routes to the robots (without specifying any synchronization between robots) taking only into account global traffic constraints.

[3]MARTHA: European ESPRIT Project No 6668. "Multiple Autonomous Robots for Transport and Handling Applications"

4.1 Mission processing

In the MARTHA application, the robots are regularly assigned missions: destination points together with routes and operations to perform when the destination points are reached (docking, loading..).

The environment is a route network: lanes, crossings, open areas. In order to allow efficient and incremental plan merging, we have decomposed the route network into smaller entities called "cells" or "spatial resources" which will be used as a basis for dealing with local conflicts. Basically, the robots navigate through an oriented graph of cells.

When a robot receives a mission, it first refines into an executable plan: a set of trajectories planned autonomously together with the sequence of resources it has to allocate.

Plan-merging will essentially consist in synchronizing the use of such resources by the different robots through inter-robot communication. It takes place from time to time, for a limited number of spatial resources ahead in order to not constrain unnecessarily the other robots.

Mission planning (refinement), plan-merging and execution may run in parallel (see Figure 2), allowing most often the robots not to stop at all (unless they have to effectively wait for a resource which is still occupied by another robot).

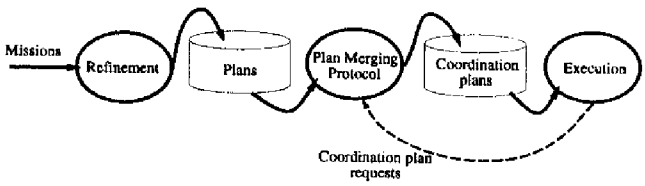


Figure 2: Interactions between mission refinement, plan-merging and execution

4.2 A Plan-Merging Protocol for multi-robot navigation in a route network

We have devised a specific Plan-Merging protocol based on resource allocation. It is an instance of the general protocol described in §3.1, where *State_1* is decomposed into several sub-states (see Figure 3).

To present the *Plan-Merging Protocol* in the particular case of traffic application, let us recall that, in *State_1*(see Figure 3), the robot should obtain the right and the necessary data to perform a *Plan-Merging Operation*.

As, in this context, *Plan-Merging Operation* is done for a limited list of required resources (the cells which will be traversed during the plan to merge), a robot,

13.3

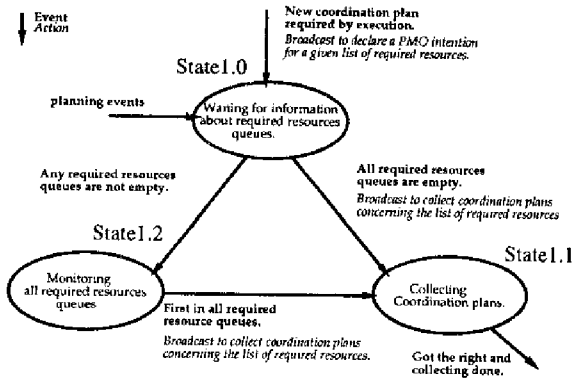


Figure 3: The state1 for traffic application in the protocol state graph

by broadcasting this list, announces its intention to start a PMO:

- No response, means that all required resource queues are empty.
- If another robot is in *State_1.1*, *State_1.2* or *State_2* and if its list of required resources intersect the list contained in the broadcast message, it sends immediately a message *(WAIT-FOR-PMO, robot-name, resource-intersect-list)*.
- If several robots enter simultaneously in *State_1.0* for common resources, the conflict is solved by taking into account the robots priority. Doing so, robots maintain together for each critical resource a queue, without risk of starvation or deadlock.

Due to place limitations, we will not describe in more detail this protocol. A full description may be found in [13].

One of the most interesting attributes of this protocol is that it allows several PMOs to be performed simultaneously if they involve disjunctive resource sets. This is particularly useful when there are several local conflicts at the same time .

4.3 When reasoning about cells is not sufficient

While, most of the time, the robots may restrict their cooperation to cell allocation, there are situations where this is not enough. This happen when they have to cross non-structured regions (called "areas") or when an unexpected obstacle, encountered in a lane or in a crossing, forces a set of robot to maneuver simultaneously in a set of cells.

When this happens, a more detailed cooperation (using the same protocol but a different planner: the motion planner) takes place allowing robots to coordinate their actions at trajectory level.

Thus, we have a hierarchy of PMOs

1. first, at the cell level, based on resource (cells) allocation
2. then , depending on the context, at trajectory level: motion planning in a set of common cells determined by the first level

This scheme authorizes a "light" cooperation, when possible, and a more detailed one, when necessary, obtained by a further refinement of the *Global Plan* without altering the properties.

4.4 An example of Plan-Merging at a crossing

We shall now illustrate the plan merging paradigm and protocol with a concrete example from the MARTHA application. We have chosen an allocation strategy which makes the robots allocate one cell (at least) ahead when they traverse lanes, while they allocate all the cells necessary to cross and leave a crossing.

The example involves several robots at a crossing divided into four cells with entry and exit cells belonging to four lanes (Figure 4):

(Instant: t0) Robot R_1 is entering c_4 and has a plan to go through c_{11}, c_{12}, and enter c_6. We shall now examine different steps (Figure 4) and show how other robots willing to go through this crossing will coordinate their plans together.

t1: R_1 has already merged its plan to traverse the crossing. R_2 needs to go through c_{12}, c_6 (i.e. a crossing cell and an exit cell); it cannot produce and merge a plan compatible with R_1's plan, since R_1 has not yet merged a plan in which it frees c_6. As a consequence R_2 defers its PMO (switch from *State_2* to *State_3*), and asks R_1 to produce a "planning-event" as soon as it has elaborated a new plan (which supposedly will contain a release operation for c_6). R_2 will remain in *State_3* until R_1 signals a *"planning-event"*. Note that R_2's execution will continue until its current coordination plan is done i.e. until the c_{12}.

t2: R_3 and R_4 attempt to start a PMO at (almost) the same time. Let us assume than R_4 has a lower priority than R_3. R_4 switches to *State_1.2* and R_3 proceeds. It merges a new plan which makes use of c_{10}, c_9 and c_3 without interfering with any other robot coordination plan.

t3: R_4 receives a message from R_3 which is now done with its PMO. R_4 switches from *State_1.2* to *State_1.1*, it broadcasts the list of its required resources (c_{11}, c_{12}, c_{10} and c_2) and gets back coordination plans from R_1 and R_3. It merges a plan with temporal constraints referring to R_1 and R_3 plans.

t4: After a while, R_3 and R_1 need again new PMOs. There is no interference between their respective required resources. Therefore, they can perform their PMOs in parallel.

13.4

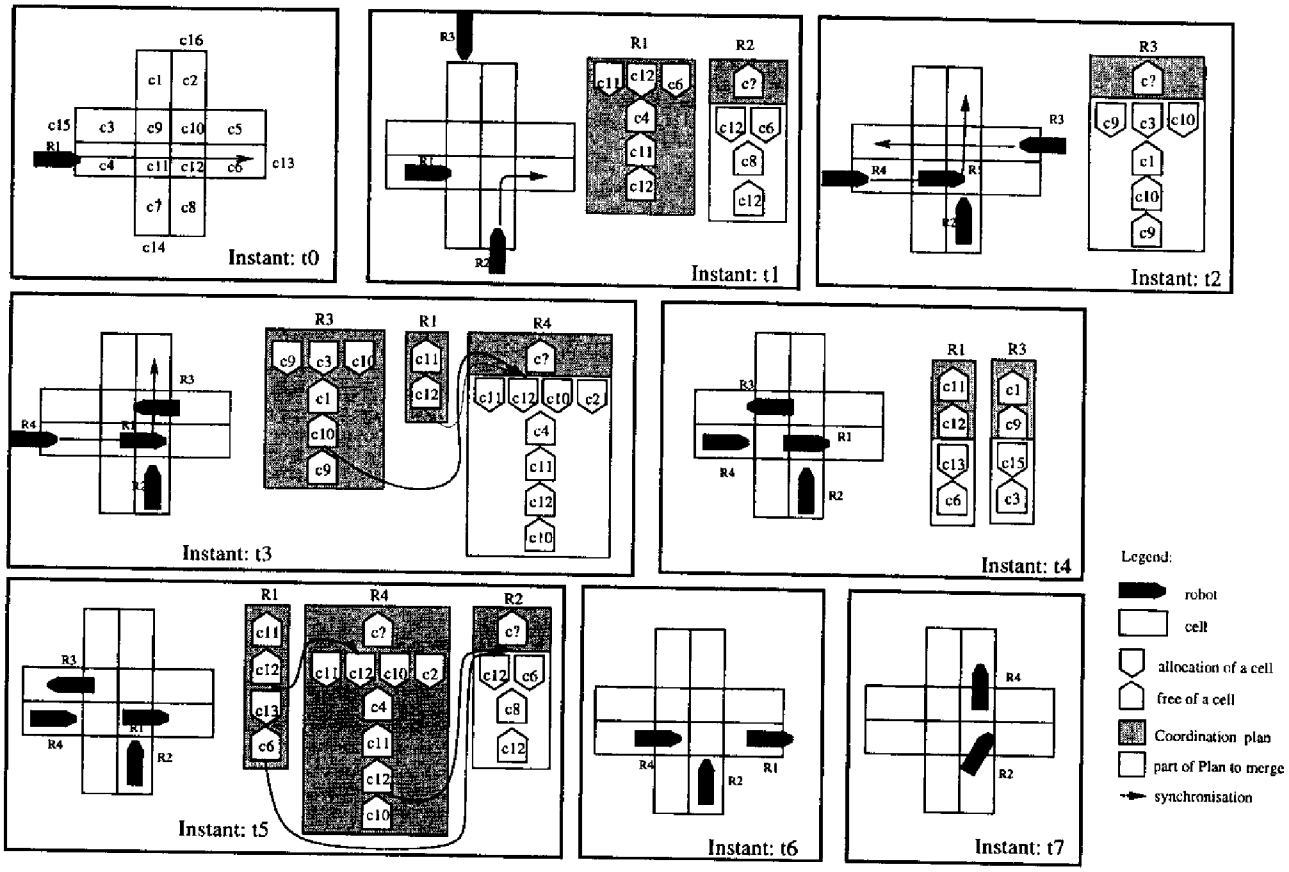


Figure 4: Plan Merging in a Crossing

t5: R_1 has now produced a new plan. It sends a *"planning-event"* to R_2 which can now switch from *State_3* to *State_1.0* and proceed.

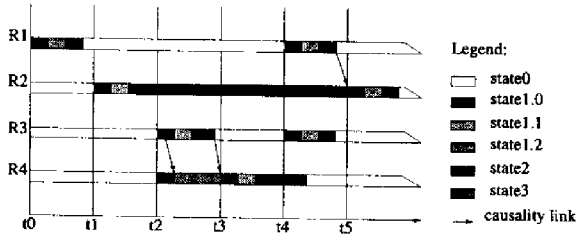


Figure 5: State change chronogram

We can make a number of remarks from this example and its associated state change chronogram (see Figure 5).

- One can see that planning and execution is done in parallel.
- More than one robot can perform a PMO at the same time for disjunctive lists of resources (e.g. R_1 and R_3 at instant **t4**).
- Several robots may use the crossing simultaneously (e.g. R_1 and R_3).
- The example exhibits the two types of synchronization: synchronization based on *execution events* (e.g. R_4 will wait until R_1 leaves c_{11}), and synchronization based on *planning events* (e.g. R_2 waits R_1 has produced and merged a new plan, at instant t5).
- Each robot produces and merges its plans iteratively, and the global plan for the use of the crossing is incrementally built through several PMOs performed by various robots.
- It is not a first arrived first served execution; for example R_2 arrived second, was blocked by R_1, but did not block the crossing and let R_3 and R_4 enter the crossing before it.

4.5 The current implementation

We have developed a complete robot control system which includes all the features described. Its ar-

13.5

chitecture is based on generic control architecture for autonomous mobile robots developed at LAAS [4, 1].

It is instantiated in this case by adding an intermediate layer for performing Plan-Merging operations (Figure 2).

The robot supervisor is coded using a C-PRS [7]; it performs Plan-Merging Operations based on a topological planner and on a motion planner. Figure 6 illustrates two trajectories planned and synchronized using a PMO at trajectory level.

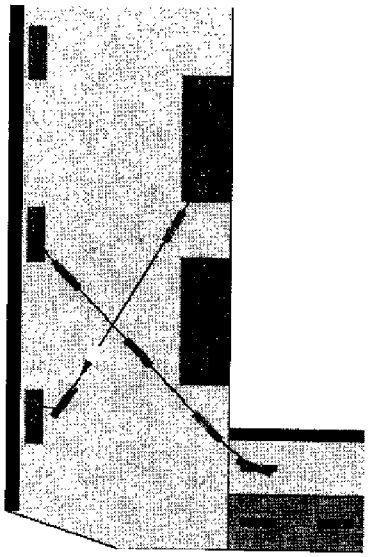


Figure 6: The result of a PMO at trajectory level

For testing and demonstration purposes, it has been linked to a robot simulator.

Experiments have been run successfully on a dozen of workstations (each workstation running a complete robot simulator) communicating through Ethernet. A 3-d graphic server has been built in order to visualize the motions and the load operations performed by all the robots in a route network environment (Figure 7). The simulated robots where able to achieve navigation missions, performing hundreds of PMOs and solving local conflicts. Motion planning and PMOs were sufficiently efficient to allow most often the robots to elaborate and merge their plans without stopping unless necessary.

The software is currently been installed under a real-time multi-processor operating system for future experimentations on real robots.

5 Related work

There are numerous contributions dealing with multi-robot cooperation. However, the term "cooperation" has been used in several contexts with different meanings.

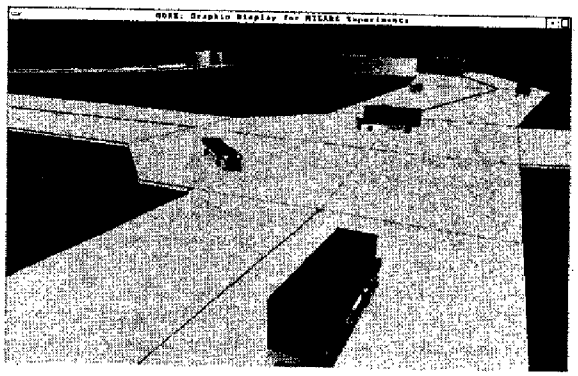


Figure 7: Several simulated robots at a crossing

We will not consider here contributions to cooperation schemes at servo level (e.g. [10]) nor contributions which aim at building an "intelligent group" of simple robots (e.g. [11]). We will limit our analysis to contributions which involve an effective cooperation (at plan or program level) between several robots.

Several approaches have been proposed, such as generation of trajectories without collision (e.g. [5, 14]), traffic rules [6, 8], negotiation for dynamic task allocation [9, 3], and synchronization by programming [12, 16].

Inter-robot communication allows to exchange various information, positions, current status, future actions, etc [3, 16, 15] and to devise effective cooperation schemes.

Traffic rules have been proposed as a way to allow several robots to avoid collision and to synchronize their motion (with limited or even without communication). However, many aspects should be taken into account in order to build the set of traffic rules: the tasks, the environment, the robot features, and so on. This entails that the generated rules are valid only under the considered assumptions. If some of them are changed, the rules have to be modified or sometimes be regenerated completely. Besides, these systems are generally built heuristically and do not provide any guarantee such as deadlock detection.

Negotiation have been used for dynamic task [3] or resource [9] allocation to several robots depending on the situation. One robot or a station allocates tasks to negotiators determined through communication.

Most contributions which make use of synchronization through communication are based on a predefined set of situations or on task dependent properties.

Indeed, most of the methods listed here, deal essentially with collision avoidance or motion coordination and cannot be directly applied to other contexts or tasks.

We claim that our Plan-Merging paradigm is a generic framework which can be applied in different

13.6

contexts, using different planners (action planners as well as motion planners). It has some clean properties (and clear limitations) which should allow, depending on the application context, to provide a coherent behavior of the global system without having to encode explicitly all situations that may encoded.

6 Conclusion and future work

The Plan-Merging paradigm we propose has the following properties;

1. It makes possible for each robot to produce a coordination plan which is compatible with all plans executed by other robots.
2. No system is required to maintain the global state and the global plan permanently. Instead, each robot updates it from time to time by executing a PMO.
3. The PMO is safe, because it is robust to plan execution failures and allows to detect deadlocks.

We believe that it can be applied to a large variety if contexts and with different planners (from action planners to task or motion planners), and at different granularities.

Such a multi-robot cooperation scheme "fills the gap" between centralized, very high level planning and distributed execution by a set of autonomous robots in a dynamic environment.

Indeed, it appears to be particularly well suited to the control of a large number of robots navigating in a route network. The application that we have implemented clearly exhibits its main features. It allowed us to make a large number of autonomous robots behave coherently and efficiently without creating a huge activity at the central system.

Besides the demonstration of real robots and the investigation of other classes of applications, our future work will concentrate on developing new cooperation schemes by embedding a multi-robot planning activity inside a PMO.

References

[1] R. Alami, R. Chatila, and B. Espiau Designing an Intelligent Control Architecture for Autonomous Robots. in ICAR'93, Tokyo (Japan), October 1993.

[2] R. Alami, F. Robert, F. F. Ingrand, and S. Suzuki. A paradigm for plan-merging and its use for multi-robot cooperation. In *IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas (USA), October 1994.

[3] H. Asama, K. Ozaki, et al. Negotiation between multiple mobile robots and an environment manager. In *'91 International Conference on Advanced Robotics (ICAR),Pisa (Italy)*, June 1991.

[4] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. Integrated planning and execution control of autonomous robot actions. In *IEEE Int. Conf. on Robotics and Automation, Nice, (France)*, 1992.

[5] H. Chu and H.A. EiMaraghy. Real-time multi-robot path planner based on a heuristic approach. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, May 1992.

[6] D.D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4(5):491–497, Oct. 1988.

[7] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6), Dec. 1992.

[8] S. Kato, S. Nishiyama, and J. Takeno. Coordinating mobile robots by applying traffic rules. In *IEEE IROS '92, Raleigh (North Carolina, USA)*, July 1992.

[9] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. In *IEEE International Conference on Robotics and Automation, Cincinnati, (USA)*, May 1990.

[10] Z.W. Luo, K. Ito, and M Ito. Multiple robot manipulators cooperative compliant manipulation on dynamical environments. In *IEEE IROS '93, Yokohama, (Japan)*, July 1993.

[11] M. Mataric. Minimizing complexity in controlling a mobile robot population. In *IEEE International Conference on Robotics and Automation, Nice, (France)*, May 1992.

[12] F.R. Noreils. Integrating multi-robot coodination in a mobile-robot control system. In *IEEE IROS '90, Tsuchiura (Japan)*, July 1990.

[13] F. Robert, R. Alami, F. F. Ingrand, and S. Suzuki. A Paradigm for Plan-Merging and its use for Multi-robot Cooperation. Technical Report feb-94, LAAS/CNRS, Toulouse, France, 1994.

[14] T. Tsubouchi and S.Arimoto. Behavior of a mobile robot navigated by an "iterated forecast and planning" scheme in the presence of multiple moving obstacles. In *IEEE Int. Conf. on Robotics and Automation, San Diego, (USA)*, May 1994.

[15] J. Wang. On sign-board based inter-robot communication in distributed robotic systems. In *IEEE International Conference on Robotics and Automation, San Diego, (USA)*, pages 1045–1050, May 1994.

13.7

[16] S. Yuta and S.Premvuti. Coordination autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. In *IEEE IROS '92, Raleigh (North Carolina, USA)*, pages 1566–1574, July 1992.

A3.8

Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment

L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert
LAAS / CNRS,
7, Avenue du Colonel Roche - 31077 Toulouse - France

Abstract: This paper presents an implemented system which allows to run a fleet of autonomous mobile robots in a route network with a very limited centralized activity. The robots are endowed with all the necessary ingredients for planning and executing navigation missions in a multi-robot context.

Multi-robot cooperation is based on a generic paradigm called *Plan-Merging Paradigm*, where robots incrementally merge their plans into a set of already coordinated plans.

The robot architecture is derived from the generic architecture developed at LAAS. A 3D graphic environment system allows to display a complete system composed of a dozen (or more) robots, each running on an independent workstation. An example is presented together with numerical results on the system behavior.

1 Introduction

We describe, in this paper, a system which allows to run a fleet of autonomous mobile robots in a route network with a very limited centralized activity. The robots are endowed with all the necessary ingredients for planning and executing navigation and load handling missions - expressed at a very high level of abstraction - as well as for multi-robot cooperation.

In order to implement a robust and efficient multi-robot cooperation capability, we use a generic paradigm called *Plan-Merging Paradigm*, where robots incrementally merge their plans into a set of already coordinated plans. This is done through exchange of information about their current state and their future actions.

The robot architecture is derived from the generic architecture developed at LAAS. The software tools we use allow us to run the robot software under VxWorks on real robots (from the Hilare family) as well as on Unix workstations emulating low level robot primitives.

A test and evaluation environment has been developed which includes a 3D graphic system to display a complete fleet of robots (a dozen or even more) in a structured environment. This testbed provides mechanisms such that each robot is fully functional and can be ran on an independent Unix workstation.

Typical runs on this testbed are presented together with numerical results on the system behavior.

2 Related Work

Various methods have been proposed which deal with multi-robot systems. Besides, the term "cooperation" has been used in several contexts with different meanings. We will not consider here cooperation schemes at servo level nor contributions dealing with "intelligent groups" of simple robots.

A thorough analysis of the literature is still to be made; we simply mention here some representative contributions which involve an effective cooperation (at plan or program level) between several robots.

Several approaches have been proposed, such as generation of trajectories without collision (e.g. [7, 21]), traffic rules [11, 14], dynamic adaptation of trajectories [10], negotiation for dynamic task allocation [15, 4], and synchronization by programming [17, 23, 22].

Concerning more particularly multi-robot motion planning methods, numerous contributions have been made which are generally based on a central planner, specially designed to cope with the intrinsic complexity of the problem [20, 8, 18]. While these methods are not complete or cannot be used for a "large" number of robots (more than 3), recent techniques based on randomized search in the Global Configuration Space [19], allow most often to obtain a solution in a reasonable time, even though, in the worst case, they fall into the unavoidable problem complexity.

Most of these contributions are based on a predefined set of situations or on task specific planners. We claim that our Plan-Merging Paradigm is a generic framework which can be applied in different contexts, using different planners (action planners as well as motion planners). It has some clean properties (and clear limitations) which should allow, depending on the application context, to provide a coherent behavior of the global system without having to encode explicitly all situations that may be encountered. Another advantage of our method is that it allows, most of the time, to solve a conflict without using a full multi-robot planner and even without stopping the execution of the other robots.

3 A fleet of Autonomous Mobile Robots

The problem consists in devising a system which allows to run a large number of autonomous mobile robots in a route network composed of lanes, crossings and open areas. Typical applications of this prob-

14.1

lem are cargo transfer as dealt with in the MARTHA project[1] which requires the development of a large fleet of autonomous mobile robots for the transportation of containers in harbors, airports and railway station environments.

3.1 Autonomous Robots need Decentralized Cooperation

The system is composed of a Central Station and a set of autonomous mobiles robots.

The current state of the art allows a substantial level of autonomy for a unique mobile robot in a structured environment: autonomous motion planning, localization based on the perception of known features in the environment, obstacle avoidance, and so on.

However, if there are several robots, and if all necessary synchronizations are performed at a central system, this will result in a very high activity at the central system and a drastic limitation of the range of the plans allocated to the robots.

The *Plan-Merging Paradigm* we propose is well suited to such applications as it allows to run a great number of robots, locally dealing with conflicts while maintaining a global coherence of the system. Indeed, it limits the role of the central system to the assignment of tasks and routes to the robots (without specifying any trajectory or any synchronization between robots) taking only into account global traffic constraints.

3.2 Mission specification

We now present the Environment Model used for mission elaboration and motion planning and the role of the Central Station in the mission elaboration.

The Environment Model: In order to allow efficient and incremental plan merging, we have decomposed the route network into smaller entities called "cells" or "spatial resources" which will be used as a basis for dealing with local conflicts.

Basically, the robots navigate through an oriented graph of cells. Lanes and crossings are composed of a set of connected cells, while areas consist of only one cell.

Thus, the environment model, which is provided to each robot, mainly contains topological and geometric information (Figures 1 and 2):

- A network describing the connections of areas and crossings by oriented lanes. This is the only information used by the Central Station to elaborate routes for robots.
- A lower level topological description (cell level). The graph of cells is oriented, in order to provide a nominal (but not exclusive) direction for lanes and crossings use.
- A geometrical description of cells (polygonal regions).
- Additional information concerning landmarks (for re-localization), station descriptions for docking and load handling actions.

[1] MARTHA: European ESPRIT Project No 6668. "Multiple Autonomous Robots for Transport and Handling Applications"

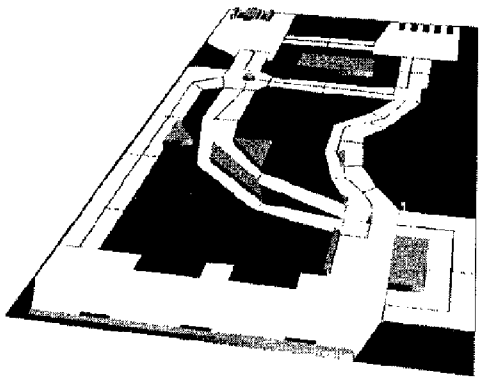


Figure 1: An environment (real size: 550 m x 250 m)

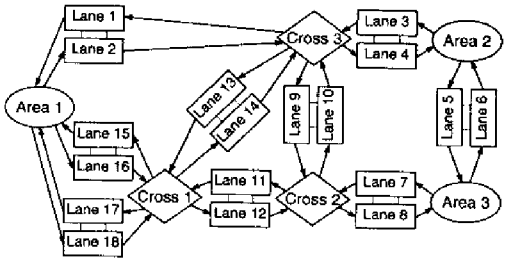


Figure 2: The associated topological graph

The role of the Central Station: The central station is in charge of producing high level plans to load/unload ships, trains or planes. A plan is produced taking into account the route network topology as well as the availability of such or such robot. However, it does not further specify the sequence of robots going through a crossing (this decision is left to the robot locally concerned), nor does it require the robot to remain on the specified lanes (in case it needs to move away from an unexpected obstacle).

3.3 A Plan-Merging Protocol for Multi-Robot Navigation

The cooperation scheme we use is based on a general paradigm, called *Plan-Merging Paradigm* [2], where robots incrementally merge their plans into a set of already coordinated plans. This is done through exchange of information about their current state and their future actions.

For the case of a number of mobile robots in a route network environment, we have devised a specific *Plan-Merging Protocol* (PMO) based on spatial resource allocation (see [3]). It is an instance of the general protocol described in [2], but in this context, *Plan-Merging Operation* is done for a limited list of required resources: a set of cells which will be traversed during the plan to merge. The robot broadcasts the set or required cells, receives back the set of coordination plans from other robots which have already planned to use some of the mentioned cells, and

then tries to perfrom a plan insertion which ensures that the union of the considered plans is a directed acyclic graph.

Due to space limitations, we will not describe in more detail this protocol. A full description may be found in [2].

One of the most interesting attributes of this protocol is that it allows several PMOs to be performed simultaneously if they involve disjunctive resource sets. This is particularly useful when there are several local conflicts at the same time.

Plan-Merging for cell occupation: In most situations, robot navigation and the associated Plan-merging procedure are performed by trying to maintain each cell of the environment occupied by at most one robot. This allows the robots to plan their trajectories independently, to compute the set of cells they will cross and to perform Plan-Merging at cell allocation level.

In order not to constrain unnecessarily the other robots, the allocation strategy makes a robot allocate one cell ahead when it moves along lanes, while it allocates all the cells necessary to traverse and leave crossings.

When reasoning about cells is not sufficient While, most of the time, the robots may restrict their cooperation to cells allocation, there are situations where this is not enough. This happens when they have to cross non-structured areas (§3.2) or when an unexpected obstacle, encountered in a lane or in a crossing, forces a set of robots to maneuver simultaneously in a set of cells. In such situations, a more detailed cooperation (using the same protocol but a different planner: the motion planner) takes place allowing robots to coordinate their actions at trajectory level. Thus, we have a hierarchy of PMOs

- first, at the cell level, based on resource (cells) allocation
- then, depending on the context, at trajectory level: motion planning in a set of common cells determined by the first level

This hierarchy authorizes a "light" cooperation, when possible, and a more detailed one, when necessary.

4 The Robot Control System

The architecture of the Robot Control System (RCS) is directly derived from the generic control architecture for autonomous robots developed at LAAS [5, 1, 9]. It is composed of a Decisional Level and a Functional Level.

4.1 The Decisional Level

The Decisional Level (also called the Robot Supervisor (RS)) consists of three layers, corresponding to a hierarchical decomposition of planning and control activities (Figure 3): the Mission layer, the Coordination layer and the Execution layer.

The first two layers are themselves composed of a planning and a a supervision process. All processes (five) run in parallel and satisfy different response time constraints [1, 13]

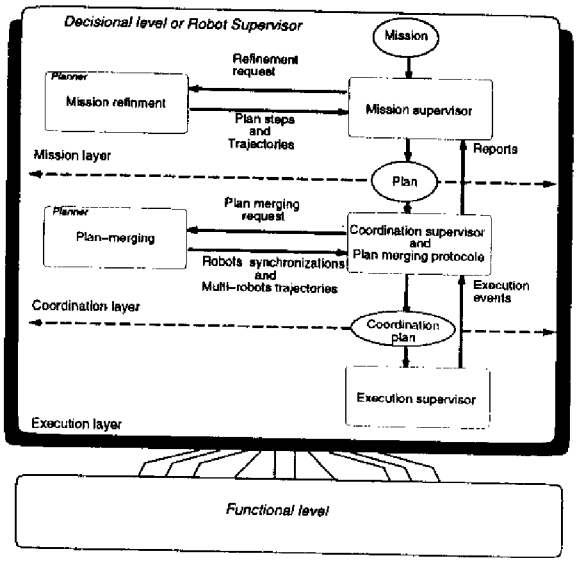


Figure 3: The robot supervisor architecture

1. The Mission layer: From time to time, the Central Station sends a new mission to the robot (Figure 4 illustrates a typical mission). The Mission layer deals with mission refinement and control. Mission refinement is performed through the use and context dependent instantiation of a library of "predefined plans" or "plans skeletons". A mission is first refined as if the robot was alone. The resulting plan is a sequence of actions (including planned trajectories) annotated with cell entry and exit monitoring operations which will be used to maintain the robot execution state and to synchronize its actions with other robots. The Mission Supervisor is in charge of controlling the execution of such plans. If a plan fails to achieve a particular goal, alternative plans are refined and attempted.

```
(mission (.
    (action 1 (goto (station 1)) (using (lane 10)))
    (action 2 (dock))
    (action 3 (put-down))
    (action 4 (un-dock))
    (action 5 (goto (station 3)) (using (lane 12) (lane 8)))
    (action 6 (dock))
    (action 7 (pick-up (container 5)))
    (action 8 (un-dock))
    (action 5 (goto (end-lane 0)) (using (lane 9) (lane 0))) .))
```

Figure 4: A mission example

2. The Coordination layer: produces and controls "coordination plans". It performs Plan-merging operations and manages the interactions with the other robots (exchanging coordination plans and events). Indeed, the plans produced by the Mission layer are incrementally validated in a multi-robot context by the Coordination layer through the use of a Plan-merging protocol. The result is a "coordination plan" which

specifies all trajectories and actions to be executed, together with all events to be monitored and sent to another robot or to be awaited from another robot (Figure 5).

Note that an execution failure reported by a robot from which an event is awaited will cause a new Plan-merging operation to be performed.

```
(coordination-plan (.
        (exec-plan 1 (report (begin-action 1)))
        (exec-plan 2 (wait-exec-event r3 9))
        (exec-plan 4 (monitor (entry (cell 4))))
        (exec-plan 8 (exec-traj 0))
        (exec-plan 3 (wait-exec-event r7 48))
        (exec-plan 5 (monitor (entry (cell 5))))
        (exec-plan 6 (monitor (exit (cell 14))
                              (signal-exec-event r4 17)))
        (exec-plan 7 (monitor (exit (cell 4))))
        (exec-plan 9 (exec-traj 1))
        (exec-plan 10 (exec-traj 2)) .))
```

Figure 5: Example of coordination plan

3. The Execution layer is in charge of the interpretation and the execution of coordination plan. As a result, it is responsible of most interactions with the functional level. Besides all actions and monitors included in the plan, it also monitors and reacts to a number of critical events, such as unexpected obstacles in its path, or its own status (battery or fuel level), failure reports from the different modules, as well messages sent by the other robots (infos about synchro events or plan failures).

4.2 The Functional Level

The functional level implements all the robot basic capabilities in sensing, acting and computing. These functionalities are grouped according to data or resource sharing, and integrated into *Modules*.

Beside real time capabilities to insure closed-loop control and reactivity, this level fulfills several conditions towards the decisional level: bounded response time to requests, observability and programmability.

All modules have the same structure (see [9]). A module is composed of a module controller and an execution level.

At the execution level, the implemented functions (i.e. embedded algorithms) are interruptible by the module controller. They can also abort by themselves if the execution cannot be achieved (internal failure, failure of a server at a lower level...). A specific report is then returned to the client.

A formal language is used to describe a module including its behavior, its interfaces and its connections with others modules. We have built an automatic modules compiler that uses a formal module description and user supplied functions to automatically produce the modules.

Figure 6 shows the functional level, for the presented application, including 7 modules, their client/-server relations and 3 exported data (posters). Each module will be described later on.

The Robot Supervisor is a client of all the modules of the functional level. It manages itself the poster

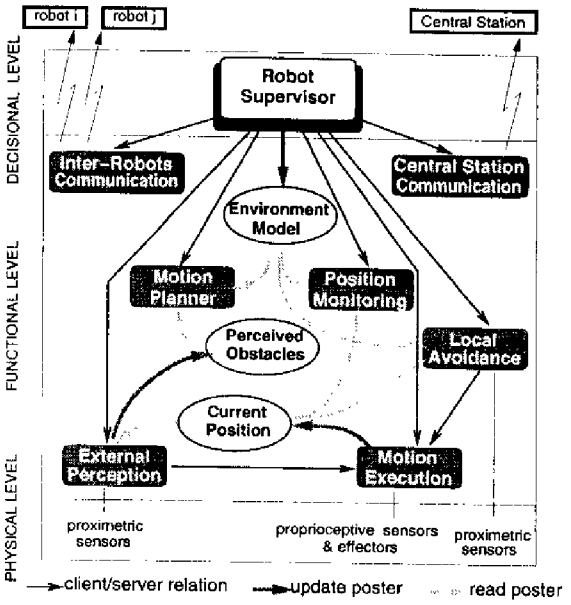


Figure 6: Architecture and interactions of the functional level

named `ENVIRONMENT` which exports the topological and geometrical model of the environment (cf §3.2).

4.2.1 The Motion Planner Module

Each robot is equipped with an independent Motion Planner composed of a Topological Planner, a Geometrical Planner and Multi-robot Scheduler. It is used in order to compute not only feasible trajectories but also synchronization events between different robot trajectories.

The Topological Planner: performs a search in the graph of cells in order to determine the set of cells to be used for a given motion task. The selection of cells may be done in two modes: a *Local obstacle Avoidance* mode that selects only the cells which correspond to the nominal way of traversing lanes or crossings, and an *Extended obstacle Avoidance* mode which is invoked, for example, when a major obstacle forces a robot to leave its current lane and to use cells belonging to a "parallel" lane.

The Geometrical Planner: computes a non-holonomic path between an initial position and a goal position. When used in a "multi-robot" mode, it produces a path which avoids the last positions of the other robots. It is based on techniques similar to those described in [6].

The Multi-robot Scheduler: determines, along each trajectory, the positions where a robot should update its set of occupied resources, the positions where it should signal a *trajectory synchronization event* to another robot, and the positions where it should stop and wait for synchronization. Figure 7 illustrates trajectory synchronizations: *Ri-Wj* stands for a position

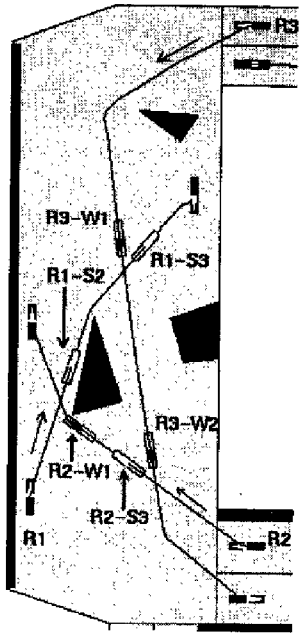


Figure 7: The result of a PMO at trajectory level (in an area)

where robot *ri* should stop and wait that robot *rj* has passed position *Rj-Si*.

4.2.2 The Motion Execution Module

This module has a permanent activity which consists in **position computation** from proprioceptive sensors (odometers, gyro) and in the feedback control on the robot position (x,y). The current position is exported in the `POSITION` poster Besides, it executes trajectories composed of a sequence of segments and arcs of circle, and a polygonal bounding area in which the robot should remain (cell, lane). The trajectories are smoothed with clothoids.

It is also able, for multi-robot coordination purposes, to monitor the curvilinear abscissa while executing a motion (cf §4.2.1).

4.2.3 The Local Obstacle Avoidance Module

The Local Avoidance module monitors obstacles with range sensors (ultra-sonic sonars or laser range finder) and filters the trajectories before transmitting them to the Motion Execution module.

According to the urgency, the robot can be stopped or the trajectory can be slightly modified in order to avoid an obstacle. However, the trajectory should remain in a bounding area specified by the Robot Supervisor. If the robot is blocked, the motion requests are ended. The Robot Supervisor will then produce a new planned trajectory taking into account the new obstacles.

4.2.4 The External Perception Module

The role of this module is twofold[16]: (1) updating the robot position using exteroceptive data (range sensors) and performing landmarks based re-localization; and (2) building and maintaining a local obstacles map which may be provided, upon request, to the Motion Planner, through the `PERCEIVED OBSTACLES` poster.

4.2.5 The Position Monitoring Module

This module allows to maintain the set of resources (cells, lanes, areas) occupied by the robot and to monitor the entry and the exit of these resources. This information is necessary for the Plan-Merging activity performed by the Robot Supervisor,

4.2.6 The External Communication Modules

Tne communications with the Central Station and with the others robots are achieved by two distinct modules called Inter-Robot Communication (IRC) and Central Station Communication (CSC).

A message between robots can be dedicated to one specific robot or broadcasted to all robots in its vicinity (the IRC is assumed to have a limited range).

5 Implementation of a Realistic Testbed

We have developed a complete robot control system which includes all the features described above.

The robot supervisor is coded using a procedural reasoning system: C-PRS [12, 13]. The target implementation runs on a multi-processor VME rack, under the VxWorks real-time system.

For testing and demonstration purposes we have built an emulation of the communication and multitasking primitives of the real-time system, that allows us to run all elements of a robot (the supervisor and all functional modules) on a Unix workstation as a set of Unix processes.

The motion execution is simulated at the lowest level by sending the elementary motion controls into the perception sub-system. Radio communications between robots and with the central station are simulated on the Ethernet network (Figure 8).

A 3-D graphic server has been built to visualize the motions and the load operations performed by the robots in their environment (Figures 1,9-12). It receives positions updates from the motion execution modules of all the robots, each running on its own workstation.

Experiments have been run successfully on a dozen of workstations (each workstation running a complete robot simulator, Figure 8). Some results are presented in the next sections.

We have also tested the implementation on two real mobile robots, and a third robot is under construction to run real experiments in a real environment.

6 Results

We shall now illustrate the plan-merging paradigm and its capabilities with some sequences from our experimentation with simulated robots in a route network with open areas. The first example presents a

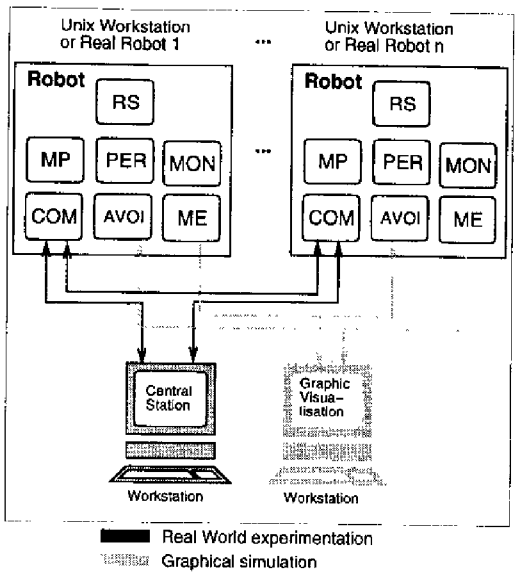


Figure 8: n Unix workstations implementing n robots

PMO at a crossing, the second, a PMO in an open area.

An example of PMOs at a crossing

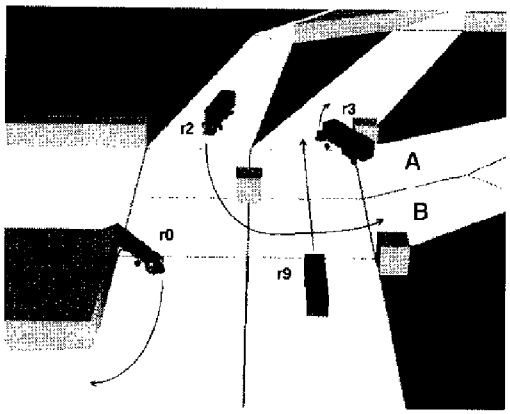


Figure 9: Crossing (Step 1)

Note that as presented in section 3.3, PMO performed at crossing allocates all the cells necessary to traverse and leave the crossing.

Crossing, Step 1 (Figure 9):

This snapshot has been edited to exhibit the routes that the robots must follow.

• Robot $r3$, coming from position A, and $r0$ have disjunctive list of resources. Therefore, they can perform PMOs in parallel and traverse the crossing without any synchronization.

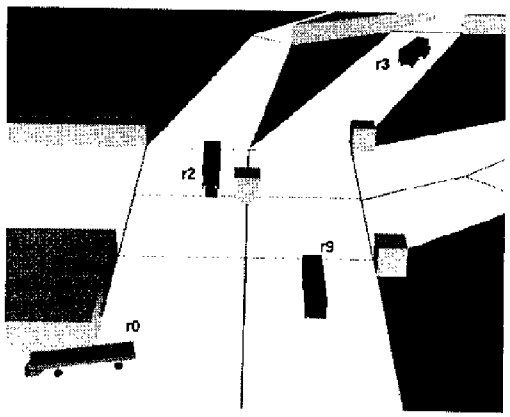


Figure 10: Crossing (Step 2)

• Robot $r9$ follows $r3$, but its PMO fails (because $r3$ has not yet planned an action to free the cell that $r9$ must allocate to exit the crossing). As a result $r9$ must wait a *planning event* from $r3$ (i.e. a new PMO).

• Robot $r2$, which wants to go to position B, finds itself in a situation where it can merge its plan with the other robots plans.

Crossing, Step 2 (Figure 10):

• Robot $r2$ traverses the crossing, after an execution synchronization with $r0$ (it must wait until $r0$ leaves the lower left cell of the crossing before entering it).

• Robot $r9$ has received the awaited *planning event* from $r3$ (which has now planned an action to exit its current cell) and its PMO succeeds, but it must synchronize with $r2$ and $r3$.

Crossing, Step 3:

• Robot $r2$ frees the crossing cells and sends the corresponding *execution event* to $r9$.

• Robot $r9$ can now traverse the crossing.

An example of PMOs in an Open Area

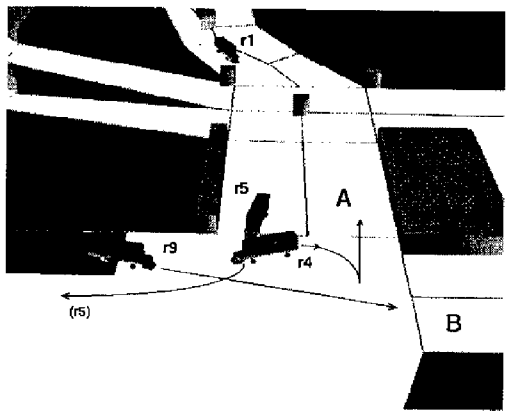


Figure 11: Open Area (Step 1)

14.6

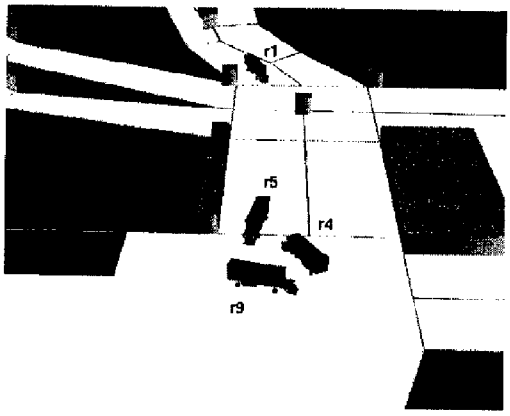


Figure 12: Open Area (Step 2)

As mentioned earlier, open areas are not exclusive resources. In our example, $r9$ goes to position B, $r4$ moves backward from a station in the area to position A and $r5$ enters the area to dock at a station.

Open Area, Step 1 (Figure 11):
The three robots have perform theirs PMOs, first at "cell level", then at "trajectory level", in the following order: $r4$, $r9$ and $r5$. $r9$ and $r5$ (which has entered the area) are both waiting until $r4$ passes a computed curvilinear abscissa. Each of them waits at a particular position (*R9-W4* and *R5-W4*) that $r4$ passes respectively *R4-S9* and *R4-S5*.

Open Area, Step 2 (Figure 12):
Robot $r5$ goes from *R5-W4* to *R5-W9* and waits for an *execution event* from $r9$ (which will be sent as soon as $r9$ passes *R9-W5*).

Open Area, Step 3:
$r9$ has sent the execution even and all the robot can now continue their mission.

The two examples, presented above, exhibit the following properties of the PMO:

- Planning and execution is done in parallel.
- Several robots may use the crossing simultaneously (r0 and r3).
- The example exhibits the two types of synchronization: *execution events* ($r4$, $r5$ and $r9$ in the area example), and *planning events* ($r9$ and $r3$ in the crossing example).
- Each robot produces and merges its plans iteratively, and the global plan for the use of the crossing is incrementally built through several PMOs performed by various robots.
- It is not a first arrived first served execution. In the crossing example $r9$ arrived second, was blocked by $r3$, but did not block the crossing, letting $r2$ enter the crossing before itself.

6.1 A Complete Mission

The simulated robots were able to achieve navigation missions, performing hundreds of PMOs and solving local conflicts. Motion planning and PMOs were sufficiently efficient to allow the robots to elaborate and merge their plans without stopping unless necessary.

From a set of experiments ran on the environment presented on Figure 1 where ten emulated robots execute each a representative mission (such as the one presented in section 4), we collected the following data: 379 messages were exchanged:

155	broadcasts for plan merging
56	plans exchanged
65	messages for execution synchronization.

During all these missions several types of conflicts have been encountered and solved:

5 conflicts for simultaneous PMO for common resources
11 conflicts for PMO queue management
16 messages to release the PMO token ($16 = 11 + 5$)
36 messages to update the deadlock detection graph.

Depending on the length and difficulty of their missions, individual robots have produced from 20 to 90 messages. The average message length is 100 bytes (without any optimization). The experiments lasted around 15 minutes for 41 k-bytes exchanged. A very simple improvement (shortening the keywords included in the messages) would result in a 20 k-bytes exchanged for the whole mission. These results show that the bandwidth required by the PMO protocol is compatible with the low baud rate provided by the communication interface.

Moreover, the robots were able to run at an average speed of 3m/s without ever stopping, unless required by a synchronization.

7 Conclusion and future work

The system described in this paper presents many original contributions to the field of research on autonomous mobile robots. To our knowledge, it is the first time such a large fleet of autonomous robots is put together to execute high level missions given by a central station.

Our experimentation using large number of emulated robots and with real robots (from the Hilare family) has shown the feasibility and the embarkability of our solution.

The Plan-Merging Paradigm we propose has the following properties;

1. It "fills the gap" between centralized, very high level planning and distributed execution by a set of autonomous robots in a dynamic environment.

2. It makes possible for each robot to produce a coordination plan which is compatible with all plans executed by other robots.

3. No system is required to maintain the global state and the global plan permanently. Instead, each robot updates it from time to time by executing a PMO.

4. The PMO is safe, because it is robust to plan execution failures and allows to detect deadlocks.

The current implementation in simulation has shown that the protocol works and allows for far more than ten robots to cooperate. In fact, considering the locality of the conflict resolution, i.e. the ability of robots

in a group to coordinate their plans and actions without disturbing the rest of the fleet, one can easily see that this protocol can scale to a much larger number of robots (hundreds). This protocol allowed us to make a large number of autonomous robots behave coherently and efficiently without creating a burden on the central system activity.

Our future work will concentrate on developing new cooperation schemes by embedding a multi-robot planning activity inside a PMO particularly in the case of motion planning.

References

[1] R. Alami, R. Chatila, B. Espiau. Designing an intellingent control architecture for autonomous robots. *ICAR'93*, Tokyo (Japan), Nov. 1993.

[2] R. Alami, F. Robert, F. Ingrand, S. Suzuki. A paradigm for plan-merging and its use for multi-robot cooperation. *IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Antonio (USA), 1994.

[3] R. Alami, F. Robert, F. Ingrand, S. Suzuki. Multi-robot cooperation through incremental plan-merging. *IEEE ICRA '95*, Nagoya (Japan), May 1995.

[4] H. Asama, K. Ozaki, et al. Negotiation between multiple mobile robots and an environment manager. *ICAR'91*, Pisa (Italy), June 1991.

[5] R. Chatila, R. Alami, B. Degallaix, H. Laruelle. Integrated planning and execution control of autonomous robot actions. *IEEE ICRA '92*, Nice (France), May 1992.

[6] J. C. Latombe. A Fast Path Planner for a Car-Like Indoor Mobile Robot. In *Ninth National Conf. on Artificial Intelligence*, 1991.

[7] H. Chu, H.A. EiMaraghy. Real-time multi-robot path planner based on a heuristic approach. *IEEE ICRA '92*, Nice (France), May 1992.

[8] M. Erdmann, T. Lozano-Perez. On multiple moving objects. *IEEE ICRA '86*, San Francisco (USA), April 1986.

[9] S. Fleury, M. Herrb, R. Chatila. Design of a modular architecture for autonomous robot. *IEEE ICRA '94*, San Diego (USA), 1994.

[10] T. Fraichard, C. Laugier. Path-velocity decomposition revisited and applied to dynamic trajectory planning. *IEEE Transactions on Robotics and Automation*, pages 40–45, 1993.

[11] D.D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4(5):491–497, Oct. 1988.

[12] F. Ingrand, M. Georgeff, A. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, Dec. 1992.

[13] F. Ingrand, R. Chatila, R. Alami, F. Robert Embedded Control of Autonomous Robots using Procedural Reasoning. *ICAR 95*, Sant Feliu de Guixols, Spain, Oct. 95.

[14] S. Kato, S. Nishiyama, J. Takeno. Coordinating mobile robots by applying traffic rules. *IEEE IROS'92*, Raleigh (USA), July 1992.

[15] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. *IEEE ICRA '90*, Cincinnati (USA), May 1990.

[16] P. Moutarlier, R. Chatila. Incremental free-space modelling from uncertain data by an autonomous mobile robot. *IEEE IROS'91*, Osaka (Japan), Nov. 1991.

[17] F.R. Noreils. Integrating multi-robot coodination in a mobile-robot control system. *IEEE IROS'90*, Tsuchiura (Japan), July 1990.

[18] P.A. O'Donnell, T. Lozano Perez. Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators. *IEEE Transactions on Robotics and Automation*, 1989.

[19] P. Svestka, M.H. Overmars. Coordinated Motion Planning for Multiple Car-Like Robots using Probabilistic Roadmaps. *IEEE Transactions on Robotics and Automation*, 1995.

[20] Pierre Tournassoud. A strategy for obstacle avoidance and its application to multi-robot systems. *IEEE ICRA '86*, San Francisco (USA), April 1986.

[21] T.Tsubouchi, S.Arimoto. Behavior of a mobile robot navigated by an "iterated forecast and planning" scheme in the presence of multiple moving obstacles. *IEEE ICRA '94*, San Diego (USA), 1994.

[22] J. Wang. On sign-board based inter-robot communication in distributed robotic systems. *IEEE ICRA '94*, San Diego (USA), 1994.

[23] S. Yuta, S. Premvuti. Coordination autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. *IEEE IROS'92*, Raleigh (USA), July 1992.

Acknowledgments: This work was partially supported by the MARTHA (ESPRIT III) Project, the CNRS, the Région Midi-Pyrénées and the ECLA ROCOMI Project.

This project is the fruit of a very intensive collaboration between numerous researchers. We would like to acknowledge the help and the effective involvement of M. Khatib, H. Bullata, S. Suzuki, J. Perret, T. Siméon, B. Dacre-Wright, C. Dousson, M. Devy, P. Gaborit.

A Multi-Robot Cooperation Scheme Based on Incremental Plan-Merging

Rachid ALAMI
LAAS / CNRS
7, Avenue du Colonel Roche
31077 Toulouse - France
e-mail: rachid@laas.fr

1 Introduction

We present and discuss a generic cooperative scheme for multi-robot cooperation. It is based on an incremental and distributed plan-merging process.

Each robot, autonomously and incrementally builds and executes its own plans taking into account the multi-robot context.

The robots are assumed to be able to collect the other robots current plans and goals and to produce plans which satisfy a set of constraints that will be discussed.

We discuss the properties of this cooperative scheme (coherence, detection of dead-lock situations) as well as the class of applications for which it is well suited.

We show how this paradigm can be used in a hierarchical manner, and in contexts where planning is performed in parallel with plan execution.

We also discuss how this paradigm "fills the gap" between (centralized or distributed) goal / task allocation and distributed task achievement[1]

We finally illustrate this scheme through an implemented system which allows a fleet of more than ten autonomous mobile robots to perform load transfer tasks in a route network environment with a very limited centralized activity, and important gains in system flexibility and robustness to execution contingencies.

[1] We use here the term "task achievement" instead of "task execution" in order to emphasize the fact that task achievement involves further (context-dependent) task refinement and sensor-based plan execution.

2 Two multi-agent cooperation issues

In the field of multi-agent cooperation we distinguish two main issues:

- **C1:** the first issue involves goal/task decomposition and allocation to various agents
- **C2:** the second issue involves the simultaneous operation of several autonomous agents, each one seeking to achieve its own task or goal.

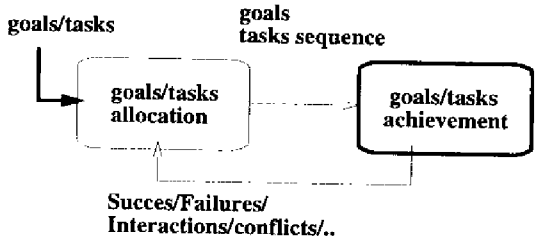


Figure 1: The two main issues in multi-agent cooperation

While several contributions have concentrated more particularly on one issue or the other, we claim that in numerous multi-agent applications, both issues appear and even "invoke" one another "recursively".

This is particularly true for autonomous multi-robot applications and, more generally, when the allocated tasks or goals cannot be directly "executed" but require further refinement.

The case of multi-robot applications is indeed directly concerned because the robots act in a same physical environment and because of the multiplicity of uncertainties.

Let us assume a set of autonomous robots, which have been given, through a centralized system or after a distributed process, a set of (partially ordered) tasks or goals. One can reasonably assume that this process is finished when the obtained tasks or goals have a sufficient range and are sufficiently independent to cause a substantial "selfish" robot activity. However, each robot, while seeking to achieve its task will have to compete for resources, to comply with other robots activities. Hence, several robots may find themselves in situations where they need to solve a new goal/task interaction leading to a new goal/task allocation scheme.

Indeed, these two issues are somewhat different in nature, and should call for different resolution schemes. While the first (**C1**) is more oriented towards the collective search for a solution to a problem and calls for a purely deliberative activity, the second (**C2**) involves a more "compliant" behavior of the agents and integrates a closer interaction between deliberation and action.

While several generic approaches have been proposed in the literature concerning task or goal decomposition and allocation (Contract Nets[25], Partial Global Planning[9], distributed search[10], negotiation [14, 5, 12, 23, 6], motivational behaviors [21, 11]), cooperation for achieving independent goals have been mostly treated using task-specific or application-specific techniques [16, 20, 22, 27]

We argue that there is also a need for generic approaches to **C2**. One can of course make the agents respect a set of rules (e.g. traffic rules), or more generally "social behaviors"[24], which are specially devised to avoid as much as possible conflicts and to provide pre-defined solutions to various situations. However, this cannot be a general answer applicable to various domains.

We would like to devise a scheme which guarantees a coherent behavior of the agents in all situations (including the avalanche of situations which may occur after an execution failure) and a reliable detection of situations which call for a new task distribution process.

In the following, we propose a paradigm[3, 4] which, we believe, provides a generic framework for **C2** issues and clearly establishes a link with **C1** issues.

3 The *Plan-Merging* Paradigm

Let us assume that we have a set of autonomous robots equipped with a reliable inter-robot communication device which allows to broadcast a message to all robots or to send a message to a given robot.

Let us assume that each robot processes sequentially the goals it receives, taking as initial state the final state of its current plan. Doing so, it incrementally appends new sequences of actions to its current plan.

However, before executing any action, a robot has to ensure that it is valid in the current multi-robot context, i.e. that it is compatible with all the plans currently under execution by the other robots. This will be done by collecting all the other robot plans and by "merging" its own plan with them. This operation is "protected" by a mutual exclusion mechanism and is performed *without modifying* the other robots plans or inserting an action which may render one them invalid in order to allow the other robots to continue execution.

We call this operation, a *Plan-Merging Operation* (PMO) and its result a *Coordination Plan* (i.e. a plan valid in the current multi-robot context). Besides new actions for the robot such a plan will specify the necessary synchronization between these new actions and the other robots coordination plans.

Note that such an operation involves only communication and computation and concerns future (near term) robot actions. It can run in parallel with the execution of the current coordination plan.

3.1 The "global plan" and its properties

Everything works as if there was a *global plan* produced and maintained by the set of robots. In fact, no robot elaborates, stores or maintains a complete representation of such a *global plan*.

At any moment, a robot has its own *coordination plan* under execution. Such a *coordination plan* consists of a sequence of actions and events to be signaled to other robots as well as events which are planned to be signaled by other robots. Such events correspond to state changes in the multi-robot context and represent temporal constraints (precedence) between actions involved in different individual coordination plans.

At any moment, the "global plan" is the graph

representing the union of all current robot coordination plans. Such a global plan is valid (i.e. it does not contain inconsistent temporal constraint) if it can be represented by a *directed acyclic graph* (*dag*).

The key point here is how to devise a system composed of a set of robots which should, as much as possible, plan independently to achieve their tasks while maintaining such property of the global plan.

3.2 The *Plan-Merging* Protocol

Let us assume here that:

1. there exists a mean which allows a robot to get the right to perform a PMO while having the guarantee that it is the only robot doing so. This right should be thought of as a resource allocation[2]

2. there is a mean allowing a robot (which has obtained the right to perform a PMO) to ask for and obtain all the other robots coordination plans.

3. there exists a mean allowing robots to ask or inform one another about the occurrence of an event.

Robots are assumed to plan from time to time (whenever it is necessary).

When a robot is not planning and even when it is waiting to obtain the right to perform a PMO, it must be able to send its current coordination plan to another robot (which currently has the right to perform a PMO).

When a robot has to plan, it uses the following protocol which we call the *Plan Merging Protocol* (see Figure 2):

1. It asks for the right to perform a PMO and waits until it obtains it together with the coordination plans of all the other robots.

2. It then builds the *dag* corresponding to the union of all coordination plans (including its own coordination plan)

3. It then tries to produce a new plan which can be inserted in the *dag*, *after* its current coordination plan. The new plan insertion may only add temporal constraints which impose that some of its actions must be executed after some time-points from other robots coordination plans.

   Besides, the insertion must maintain the fact that the obtained global plan is still a *dag*.

4. If it succeeds in producing the desired plan, the robot appends it to its current coordination plan.

5. And finally, it releases the right to perform a PMO.

[2] A simple way to do it is, for instance, to maintain a "token" through communication

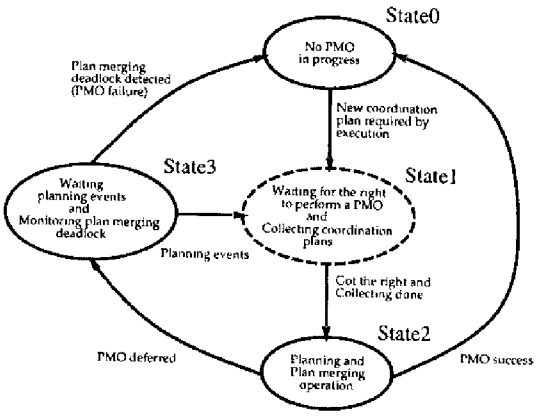


Figure 2: The general protocol state graph

When a robot executes its coordination plan, if it reaches a step with a temporal constraint linked to another robot time-point, it asks that robot if it has passed that time-point or not. Depending on the answer, the robot will wait until the other robot informs it or will immediately proceed.

3.3 Situations where PMO is deferred or where deadlock is detected

When a robot tries to perform a PMO, it may fail to produce a plan which satisfies the properties discussed earlier.

This may happen in two situations:

1. the goal can never be achieved. This can be detected if the robot cannot produce a plan even if it was alone in the environment.

2. the robot can generate a plan but this plan cannot be inserted in the global plan. This means that the final state of another robot forbids it to insert its own plan.

In such situation, the robot can simply abandon the PMO and decide to wait until the robots, that it has identified, have performed a new PMO which may possibly make them change the states preventing it to insert its plan.

Hence, we have introduced two types of events:

1. *execution events:* i.e. events which occur during plan execution and which allow robots to synchronize their execution.
2. *planning events:* i.e. events which occur whenever a robot performs a new PMO. These events can also be awaited for.

Note that, even when a robot fails in its PMO, it leaves the global plan in a correct state (it is still a *dag* and its execution can continue).

In order to detect deadlocks, a robot which finds itself in a situation where it has to wait for a *planning event* from a particular robot, must inform it. Then, it becomes possible for a robot to monitor and detect *deadlock situations* by propagating and updating, the list of robots waiting (directly or by transitivity) for *planning events* from itself. Indeed, a deadlock is detected when a robot finds itself in the list of robots waiting for itself.

When a deadlock occurs, it is necessary to take explicitly into account, in a *unique planning operation*, a conjunction of goals (which have been given separately to several robots).

This simply means that the global mission was too constrained to be solved using the Plan-Merging Paradigm. Here we must recall that we do not claim that the Plan-Merging paradigm can solve or help to solve multi-robot planning problems. The main point here is that the Plan-Merging paradigm is *safe* as it includes the detection of the deadlocks i.e. situations where a cooperation scheme of type **C1** should take place.

Note also that, in the case where only a small number of robots are involved in a deadlock, one can decide to allow the robot, which detected the deadlock, to plan for all the concerned robots . The Plan-Merging paradigm remains then applicable: the inserted plan will then concern several robots at a time.

A detailed discussion on the properties of the Plan-merging paradigm as well as on its ability to cope with execution failures can be found in [3].

The paradigm and the protocol presented so far is generic. We believe that it can be used in numerous applications. Several instances of the general paradigm can be derived, based on different planners: action planners in the stream of STRIPS, as well as more specific task planners or motion planners.

One class of applications which seems particularly well suited is the control of a large number of autonomous mobile robots.

We present in the sequel an application in the case of a fleet of autonomous mobile robots.

4 A fleet of autonomous mobile robots

We have applied the Plan-Merging Paradigm in the framework of the MARTHA project[3] which deals with the control of a large fleet of autonomous mobile robots for the transportation of containers in harbors, airports and railway environments.

In such context, the dynamics of the environment, the impossibility to correctly estimate the duration of actions (the robots may be slowed down due to obstacle avoidance, and delays in load and un-load operations, etc..) prevent a central system from elaborating efficient and reliable detailed robot plans.

The use of the Plan-Merging paradigm allowed us to deal with several types of conflicts in a general and systematic way, and to limit the role of the central system to the assignment of tasks and routes to the robots (without specifying any trajectory or any synchronization between robots) taking only into account global traffic constraints.

4.1 A Plan-Merging Protocol for Multi-Robot Navigation

For the case of a number of mobile robots in a route network environment, we have devised a specific *Plan-Merging Protocol* (PMO) based on spatial resource allocation (see [4]). It is an instance of the general protocol described above, but in this context, *Plan-Merging Operation* is done for a limited list of required spatial resources: a set of cells which will be traversed during the plan to merge. The robot broadcasts the set or required cells, receives back the set of coordination plans from other robots which have already planned to use some of the mentioned cells,

[3]MARTHA: European ESPRIT Project No 6668, "Multiple Autonomous Robots for Transport and Handling Applications"

and then tries to perform a plan insertion which ensures that the union of the considered plans is a directed acyclic graph.

One of the most interesting attributes of this protocol is that it allows several PMOs to be performed simultaneously if they involve disjunctive resource sets. This is particularly useful when there are several local conflicts at the same time.

4.1.1 *Plan-Merging for cell occupation:*

In most situations, robot navigation and the associated Plan-merging procedure are performed by trying to maintain each cell of the environment occupied by at most one robot. This allows the robots to plan their trajectories independently, to compute the set of cells they will cross and to perform Plan-Merging at cell allocation level.

In order not to constrain unnecessarily the other robots, the allocation strategy makes a robot allocate one cell ahead when it moves along lanes, while it allocates all the cells necessary to traverse and leave crossings.

4.1.2 *When reasoning about cells is not sufficient*

While, most of the time, the robots may restrict their cooperation to cells allocation, there are situations where this is not enough. This happens when they have to cross large (non-structured) areas or when an unexpected obstacle, encountered in a lane or in a crossing, forces a set of robots to maneuver simultaneously in a set of cells. In such situations, a more detailed cooperation (using the same protocol but a different planner: the motion planner) takes place allowing robots to coordinate their actions at trajectory level. Figure 3 illustrates the results of a *PMO* at trajectory level leading to trajectory synchronizations: R_i-W_j stands for a position where robot r_i should stop and wait that robot r_j has passed position R_j-S_i.

Thus, we have a hierarchy of PMOs

- first, at the cell level, based on resource (cells) allocation
- then, depending on the context, at trajectory level: motion planning in a set of common cells determined by the first level

This hierarchy authorizes a "light" cooperation, when possible, and a more detailed one, when necessary.

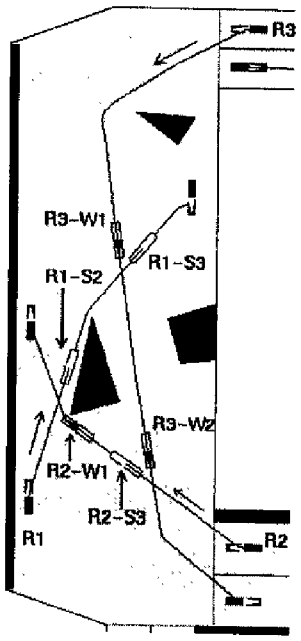


Figure 3: The result of a PMO at trajectory level (in an area)

4.2 Examples

We shall now illustrate the plan-merging paradigm and its capabilities with some sequences from our experimentation with simulated robots in a route network with open areas. The first example presents a PMO at a crossing, the second, a PMO in an open area.

An example of PMOs at a crossing

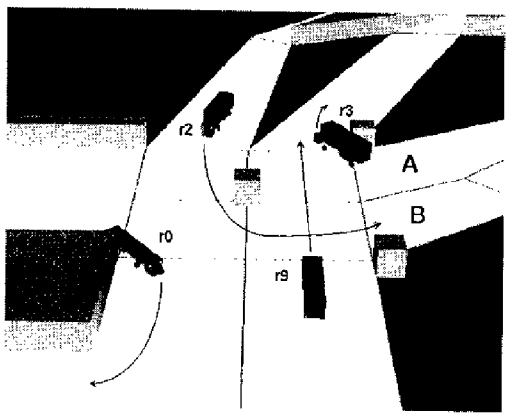


Figure 4: Crossing (Step 1)

15.5

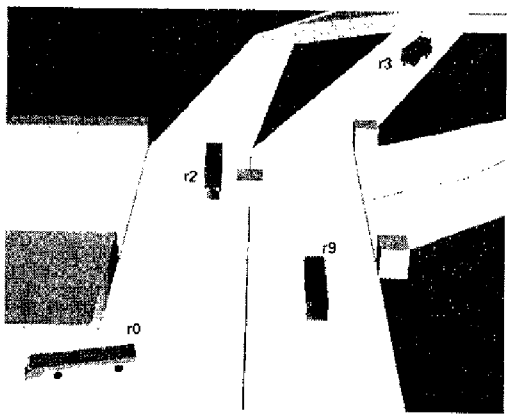


Figure 5: Crossing (Step 2)

Note that as presented in section 4.1.1, PMO performed at crossing allocates all the cells necessary to traverse and leave the crossing.

Crossing, Step 1 (Figure 4):
This snapshot has been edited to exhibit the routes that the robots must follow.

- Robot $r3$, coming from position A, and $r0$ have disjunctive list of resources. Therefore, they can perform PMOs in parallel and traverse the crossing without any synchronization.
- Robot $r9$ follows $r3$, but its PMO fails (because $r3$ has not yet planned an action to free the cell that $r9$ must allocate to exit the crossing). As a result $r9$ must wait a *planning event* from $r3$ (i.e. a new PMO).
- Robot $r2$, which wants to go to position B, finds itself in a situation where it can merge its plan with the other robots plans.

Crossing, Step 2 (Figure 5):

- Robot $r2$ traverses the crossing, after an execution synchronization with $r0$ (it must wait until $r0$ leaves the lower left cell of the crossing before entering it).
- Robot $r9$ has received the awaited *planning event* from $r3$ (which has now planned an action to exit its current cell) and its PMO succeeds, but it must synchronize with $r2$ and $r3$.

Crossing, Step 3:

- Robot $r2$ frees the crossing cells and sends the corresponding *execution event* to $r9$.
- Robot $r9$ can now traverse the crossing.

An example of PMOs in an Open Area

As mentioned earlier, open areas are not exclusive resources. In our example, $r9$ goes to position B, $r4$ moves backward from a station in the area to position A and $r5$ enters the area to dock at a station.

Open Area, Step 1 (Figure 6):
The three robots have perform theirs PMOs, first at "cell level", then at "trajectory level", in the following order: $r4$, $r9$ and $r5$. $r9$ and $r5$ (which has entered the area) are both waiting until $r4$ passes a computed curvilinear abscissa. Each of them waits at a particular position (*R9-W4* and *R5-W4*) that $r4$ passes respectively *R4-S9* and *R4-S5*.

Open Area, Step 2 (Figure 7):
Robot $r5$ goes from *R5-W4* to *R5-W9* and waits for an *execution event* from $r9$ (which will be sent as soon as $r9$ passes *R9-W5*).

Open Area, Step 3:
$r9$ has sent the execution even and all the robot can now continue their mission.

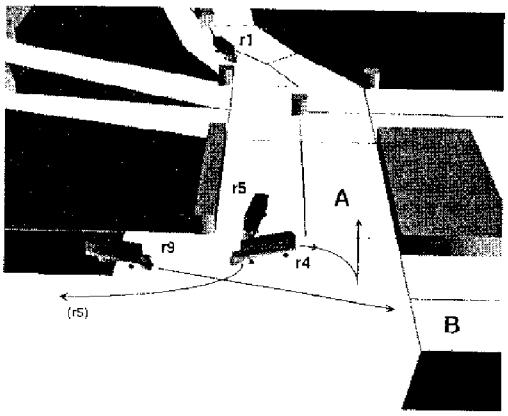


Figure 6: Open Area (Step 1)

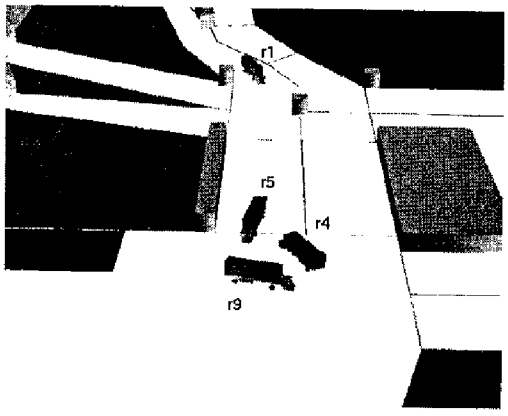


Figure 7: Open Area (Step 2)

The two examples, presented above, exhibit the

following properties of the PMO:

- Planning and execution is done in parallel.
- Several robots may use the crossing simultaneously (r0 and r3).
- The example exhibits the two types of synchronization: *execution events* ($r4$, $r5$ and $r9$ in the area example), and *planning events* ($r9$ and $r3$ in the crossing example).
- Each robot produces and merges its plans iteratively, and the global plan for the use of the crossing is incrementally built through several PMOs performed by various robots.
- It is not a first arrived first served execution. In the crossing example $r9$ arrived second, was blocked by $r3$, but did not block the crossing, letting $r2$ enter the crossing before itself.

4.3 Implementation and Results

We have developed a complete robot control system which includes all the features described. Its architecture is based on a generic control architecture for autonomous mobile robots developed at LAAS [7, 2]. It is instantiated in this case by adding an intermediate layer for performing Plan-Merging operations.

For testing and demonstration purposes, it has been linked to a robot simulator.

Experiments have been run successfully on a dozen of workstations (each workstation running a complete robot simulator) communicating through Ethernet. A 3-d graphic server has been built in order to visualize the motions and the load operations performed by all the robots in a route network (Figure 6) or in-door (Figure 8) environments. The simulated robots where able to achieve navigation missions, performing hundreds of PMOs and solving local conflicts. Motion planning and PMOs were sufficiently efficient to allow most often the robots to elaborate and merge their plans without stopping unless necessary.

Extensive experiments have also been performed using three lab robots (Fig 9).

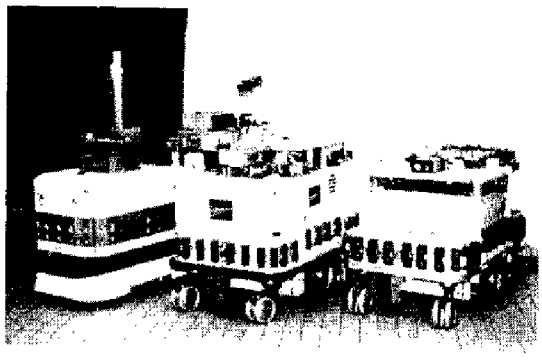


Figure 9: The Hilare family

5 Related work

There are numerous contributions dealing with multi-robot cooperation. However, the term "cooperation" has been used in several contexts with different meanings.

We will not consider here contributions to cooperation schemes at servo level (e.g. [17]) nor contributions which aim at building an "intelligent group" of simple robots (e.g. [18]). We will limit our analysis to contributions which involve an effective cooperation (at plan or program level) between several robots.

Several approaches have been proposed, such as generation of trajectories without collision (e.g. [8, 26]), traffic rules [13, 15], negotiation for dynamic task allocation [16, 5], and synchronization by programming [19, 22].

Inter-robot communication allows to exchange various information, positions, current status, future actions, etc [5, 22, 27] and to devise effective cooperation schemes.

Traffic rules have been proposed as a way to allow several robots to avoid collision and to synchronize their motion (with limited or even without communication). However, many aspects should be taken into account in order to build the set of traffic rules: the tasks, the environment, the robot features, and so on. This entails that the generated rules are valid only under the considered assumptions. If some of them are changed, the rules have to be modified or sometimes be regenerated completely. Besides, these systems are generally built heuristically and do not provide any guarantee such as deadlock detection.

Negotiation have been used for dynamic task [5] or resource [16] allocation to several robots on a situation-dependent basis.

Most contributions which make use of synchronization through communication are based on a pre-defined set of situations or on task dependent properties.

Indeed, most of the methods listed here, deal

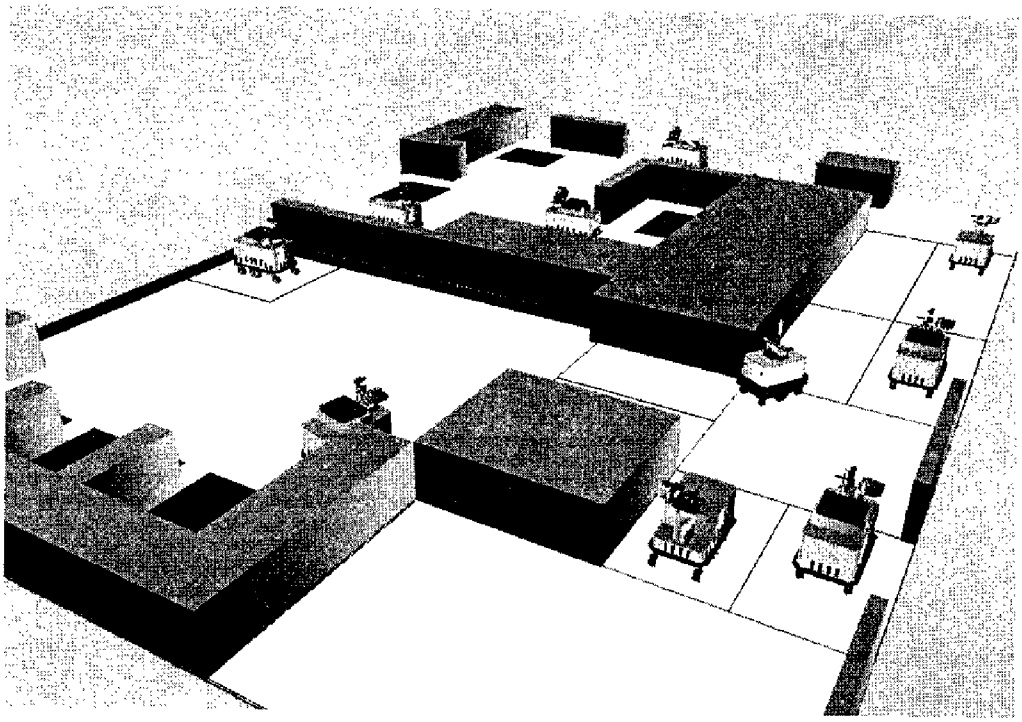


Figure 8: Several (10) Hilare using the Plan-Merging Paradigm (simulation)

essentially with collision avoidance or motion coordination and cannot be directly applied to other contexts or tasks.

We claim that our Plan-Merging paradigm is a generic framework which can be applied in different contexts, using different planners (action planners as well as motion planners). It has some clean properties (and clear limitations) which should allow, depending on the application context, to provide a coherent behavior of the global system without having to encode explicitly all situations that may encoded.

6 Conclusion

We have argued that, in the field of multi-robot (and more generally multi-agent) cooperation, it is useful to distinguish between two main issues: **C1** goal/task decomposition and allocation, and **C2** cooperation while seeking to achieve loosely coupled goals.

We have even claimed that in numerous multi-agent applications, both issues appear and even "invoke" one another "recursively".

We have then proposed a "generic" approach called *Plan-Merging Paradigm* which deals with **C2** issues and clearly establishes a link with **C1** issues.

The Plan-Merging paradigm has the following properties;

1. It makes possible for each robot to produce a coordination plan which is compatible with all plans executed by other robots.

2. No system is required to maintain the global state and the global plan permanently. Instead, each robot updates it from time to time by executing a PMO.

3. The PMO is safe, because it is robust to plan execution failures and allows to detect deadlocks.

We believe that it can be applied to a large variety of contexts and with different planners (from action planners to task or motion planners), and at different granularities.

Such a multi-robot cooperation scheme "fills the gap" between very high level planning (be it centralized or distributed) and distributed execution by a set of autonomous robots in a dynamic environment.

Indeed, it appears to be particularly well suited to the control of a large number of robots navigating in a route network. The application that we have implemented clearly exhibits its main features. It allowed us to make a large number of autonomous robots behave coherently and efficiently without creating a huge activity at the central system.

Besides the investigation of other classes of applications and the work on a more formal description of the proposed approach, our future work will concentrate on developing new cooperation schemes by embedding a multi-robot planning activity.

References

[1] L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert *Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment* In, *IEEE International Conference on Intelligent Robots and Systems*, IROS'95, Pittsburgh (USA), July 1995.

[2] R. Alami, R. Chatila, and B. Espiau. Designing an Intelligent Control Architecture for Autonomous Robots. in *International Conference on Advanced Robotics*, Tokyo (Japan), October 1993.

[3] R. Alami, F. Robert, F. Ingrand, and S. Suzuki. A paradigm for plan-merging and its use for multi-robot cooperation. In *IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, Texas (USA), October 1994.

[4] R. Alami, F. Robert, F. Ingrand, S. Suzuki. Multi-robot Cooperation through Incremental Plan-Merging In *IEEE International Conference on on Robotics and Automation*, Nagoya (Japan), May 1995.

[5] H. Asama, K. Ozaki, et al. Negotiation between multiple mobile robots and an environment manager. In *International Conference on Advanced Robotic*, Pisa (Italy), June 1991.

[6] Ronen I. Brafman, Yoav Shoham. Knowledge considerations in robotics and distribution of robotics tasks. *IJCAI 95*, Montréal (Canada), August 1995.

[7] R. Chatila, R. Alami, B. Degallaix, and H. Laruelle. Integrated planning and execution control of autonomous robot actions. In *IEEE International Conference on Robotics and Automation*, Nice, (France), May 1992.

[8] H. Chu and H.A. EiMaraghy. Real-time multi-robot path planner based on a heuristic approach. In *IEEE International Conference on Robotics and Automation*, Nice, (France), May 1992.

[9] E. H. Durfee, V. Lesser Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation In *IEEE Transactions on Systems, Man and Cybernetics*, Vol 21 (5), October 1991.

[10] E. H. Durfee, T. A. Montgomery Coordination as Distributed Search in a Hierarchical Behavior Space In *IEEE Transactions on Systems, Man and Cybernetics*, Vol 21 (6), December 1991.

[11] Eithan Ephrati, Motty Perry, Jeffrey S. Rosenschein. Plan execution motivation in multi-agent systems. *AIPS'94*, Chicago, June 1994.

[12] George Ferguson, James F. Allen. Arguing about plans: plan representation and reasoning for mixed-initiative planning. *AIPS'94*, Chicago, June 1994.

[13] D.D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4(5):491–497, October 1988.

[14] N.R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions *Artificial Intelligence*, Vol 73 (1995) 195-240.

[15] S. Kato, S. Nishiyama, and J. Takeno. Coordinating mobile robots by applying traffic rules. In *IEEE International Conference on Intelligent Robots and Systems*, IROS'92, Raleigh, (USA), July 1992.

[16] C. Le Pape. A combination of centralized and distributed methods for multi-agent planning and scheduling. *IEEE International Conference on Robotics and Automation*, Cincinnati (USA), May 1990.

[17] Z.W. Luo, K. Ito, and M Ito. Multiple robot manipulators cooperative compliant

manipulation on dynamical environments. In *IEEE International Conference on Intelligent Robots and Systems*, IROS'93, Yokohama, (Japan), July 1993.

[18] M. Mataric. Minimizing complexity in controlling a mobile robot population. In *IEEE International Conference on Robotics and Automation*, Nice, (France), May 1992.

[19] F.R. Noreils. Integrating multi-robot coordination in a mobile-robot control system. In *IEEE International Conference on Intelligent Robots and Systems*, IROS'90, Tsuchiura (Japan), July 1990.

[20] K. Ozaki, H. Asama, et al. Synchronized motion by multiple mobile robots using communication. *IEEE International Conference on Intelligent Robots and Systems*, IROS'93, Yokohama (Japan), July 1993.

[21] Lynne E. Parker. Heterogeneous multi-robot cooperation. *MIT Technical Report AITR-1465* , February 1994.

[22] S. Yuta, S. Premvuti. Coordination autonomous and centralized decision making to achieve cooperative behaviors between multiple mobile robots. *IEEE International Conference on Intelligent Robots and Systems*, IROS'92, Raleigh (USA), July 1992.

[23] Jeffrey S. Rosenschein, Gilad Zlotkin. Designing conventions for automated negotiation. *AI MAGAZINE, Volume 15, (1994)*

[24] Yoav Shoham, Moshe Tennenholtz. On social laws for artificial societies: Off-Line design. *Artificial Intelligence, 73 (1995) 231-252*

[25] Reid G. Smith. The Contract Net Protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, Vol C-29 (12), December 1980.

[26] T. Tsubouchi and S.Arimoto. Behavior of a mobile robot navigated by an "iterated forecast and planning" scheme in the presence of multiple moving obstacles. In *IEEE International Conference on Robotics and Automation*, San Diego, (USA), May 1994.

[27] J. Wang. On sign-board based inter-robot communication in distributed robotic systems. In *IEEE International Conference on Robotics and Automation*, San Diego, (USA), May 1994.

Acknowledgments: This work was partially supported by the MARTHA (ESPRIT III) Project, the CNRS, the Région Midi-Pyrénées and the ECLA ROCOMI Project.

It is the fruit of a very intensive collaboration between numerous researchers: F. Robert, F. Ingrand, S. Fleury, M. Herrb, L. Aguilar. I would like also to acknowledge the help and the effective involvement of M. Khatib, H. Bullata, S. Suzuki, J. Perret, T. Siméon, B. Dacre-Wright, M. Devy.

15.10

Robots autonomes: du concept au robot. Architectures, représentations et algorithmes

Le travail présenté procède de l'ambition de doter le robot d'un haut niveau de flexibilité et d'adaptation à la tâche en présence d'imprécisions et d'incertitudes liées à celle-ci et à son état interne. Ceci se traduit par le développement de concepts et d'outils visant à permettre au robot de planifier sa tâche et d'en contrôler l'exécution.
Une première partie porte sur l'élaboration d'architectures permettant d'intégrer les composantes décisionnelle et fonctionnelle et de mettre en œuvre des processus bouclés sur la tâche et sur l'environnement à différents niveaux d'abstraction. Elle présente notamment une architecture de contrôle générique permettant à la fois l'élaboration d'un plan d'actions (processus généralement coûteux en temps calcul et non borné dans le temps), et la disponibilité permanente dans un environnement évolutif (réactivité).
Un deuxième aspect concerne le développement de représentations et d'algorithmiques pour la planification et l'interprétation de plans: planification logique et temporelle (au niveau de la mission) mais aussi planification géométrique (plus proche de la tâche). Les contributions portent sur la planification de mission avec prise en compte de contraintes temporelles et du non-déterminisme, la coopération multi-robots, la planification des tâches de manipulation, ainsi que la planification de stratégies de déplacement pour un robot mobile en présence d'incertitudes.
La dernière partie présente la réalisation effective de *systèmes robotiques complets* démontrant les capacités développées et servant de support de validation et d'aiguillons exigeants à l'extension de ces mêmes capacités.

Mots clefs: Robots autonomes, Systèmes décisionnels, Architecture de contrôle, Planification de tâches, Planification de trajectoires, Manipulation, coopération multi-robots, Cellules Flexibles d'assemblage.

Autonomous Robots: from the concept to the robot. Architectures, representations and algorithms

This work aims at endowing the robot with a high level of flexibility and adaptability in presence of inaccuracies and uncertainties on its current state as well as on the task state. This has entailed the development of concepts and tools which allow the robot to plan its task and to monitor its execution.
The first part concerns the elaboration of architectures in order to integrate decisional and functional components and to run deliberative processes at different levels of abstractions. We present a generic control architecture which provides a framework for implementing planning, time-bounded decision and reaction at different levels, from task-oriented closed loops to situation-driven decision and goal-driven plan generation.
The second part involves the development of representations and algorithms for planning and plan interpretation for mission planning as well as for task planning. The presented contributions deal with mission planning taking into account temporal constraints and non-determinism, a novel approach to multi-robot cooperation, a new formulation of the manipulation planning problem as well as a motion planner for a mobile robot in presence of uncertainties.
The last part reports on the effective integration of most of the presented capabilities into complete autonomous robots which serve as a highly demanding validation testbed as well as a source for future extensions.

Key words : Autonomous robots, Real-time Decisional systems, Motion and task planning, manipulation, Multi-robot cooperation, Flexible assembly cells.