Navigation sous contraintes : planification et contrôle d'exécution pour un robot mobile autonome

Olivier Causse

THESE

présentée par

—

Olivier CAUSSE

—

à l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir le grade de

DOCTEUR spécialité INFORMATIQUE

Arrêté ministériel du 30 mars 1990

—

NAVIGATION SOUS CONTRAINTES : PLANIFICATION ET CONTRÔLE D'EXÉCUTION POUR UN ROBOT MOBILE AUTONOME

—

Thèse soutenue le 22 novembre 1994

Composition du jury :

Président :	M. Bernard Espiau
Rapporteurs :	M. Raja Chatila
	M. Philippe Coiffet
Examinateurs :	M. Luc-Henri Pampagnin
	M. James L. Crowley (Directeur de thèse)

Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle
46, av. Félix Viallet. 38031 Grenoble Cedex

Remerciements

Écrire des remerciements apporte des satisfactions multiples. Placée en premier mais écrite en dernier, cette page est le signe d'un achèvement. Elle est également le signe d'un travail qui, bien que personnel, est inspiré et enrichi par tous les membres d'une équipe. C'est enfin la seule page où l'expression peut s'affranchir des limites de la rigueur scientifique et où l'auteur peut exposer ses sentiments.

Je remercie

Monsieur Bernard Espiau, Directeur de recherches INRIA, pour l'honneur qu'il me fait de présider le jury,

Messieurs Raja Chatila, Directeur de recherches CNRS, et Philippe Coiffet, Directeur de recherches CNRS, d'avoir accepté la lourde tâche d'être rapporteurs.

Monsieur Luc-Henri Panpagnin, chef de projet au département recherche à ITMI (Industrie et Technologie de la Machine Intelligente), d'avoir accepté d'être examinateur.

Monsieur James L. Crowley, Professeur à l'ENSIMAG, examinateur, mais avant tout directeur de cette thèse. Il m'a permis de réaliser ce travail dans les meilleures conditions ; sa patience à mon égard et son dynamisme m'ont été précieux.

Monsieur Philippe Jorrand, Directeur du LIFIA, pour m'avoir accepté au sein de son laboratoire,

Monsieur Augustin Lux, avec qui le débat d'idées a été un plaisir constructif. J'ai eu la chance de bénéficier de ses lumières à des moments décisifs pour moi,

Madame Joelle Coutaz, son énergie et son enthousiasme explique la part importante qu'a pris le domaine de l'interface homme-machine dans ce manuscrit.

Mes amis de l'équipe PRIMA me sont chers. Partager tant d'années (...) à conjuger nos efforts pour faire bouger une masse de 250 kg peut sembler dérisoire à un lecteur peu averti, mais crée des liens indélébiles entre les protagonistes. C'est avec Philippe Bobet et Patrick Reignier que j'ai eu le plaisir de commencer ce travail et je suis heureux que nous soyons de concert à la présente étape. Ils ont souffert trop longtemps mes blagues, jeux de mots et calembours lourds mais j'ai ouï dire que cela leur a manqué pendant mon escapade nordique. Toute équipe a un Gourou, le notre est grand, c'est Chistophe Discours. Certains membres, s'ils sont déjà partis, ne sont pas oubliés : Olivier Boissier, Mouafak Mesrabi, d'autres sont arrivés en cours de route : Patrice de Marconnay, Cordelia Schmid, Bruno Zoppis, Claude Poizat et Bernt Schiele. Je n'oublie pas tous les membres du LIFIA qui font que ce laboratoire est un lieu de travail agréable. Merci enfin à Anne et Nina pour leurs tentatives de me réconcilier avec l'orthographe.

Résumé

Le travail présenté dans ce mémoire propose une contribution à la planification et au contrôle d'exécution de missions pour un robot mobile autonome, en environnement structuré, dynamique et partiellement connu : il porte sur l'étude et la réalisation d'un système de supervision pour robot mobile. Ce superviseur, qui se trouve au sommet d'une hiérarchie de processus de perception et de commande du robot, a pour rôle de planifier la mission spécifiée par un opérateur et d'en contrôler l'exécution tout en assurant le dialogue entre l'opérateur et le robot.

Nous proposons d'abord un langage permettant, à un operateur non informaticien, de spécifier des mission avec des contraintes pouvant porter soit sur la mission elle même (coûts en durée, énergie, risque, distance, localisation) soit sur l'environnement (accessibilité variable dans le temps, possibilités de recharge ou de relocalisation). La planification s'énonce alors comme un problème de recherche d'un chemin optimal contraint dans un graphe. Pour la résolution de ce problème nous introduisons la notion de chemin efficace appliquée à un nouvel algorithme de recherche heuristique pour résoudre le problème des contraintes multiples. La détermination des chemins efficaces repose sur une représentation d'état adaptée aux contraintes sur l'environnement. Les caractéristiques de complexité de cet algorithme de planification sont comparées à celles de l'algorithme A*.

L'exécution de la mission doit assurer l'activation et la supervision des processus chargés d'accomplir les tâches planifiées. L'architecture de contrôle du robot défini une décomposition hiérarchique du contrôle entre le niveau réactif qui réalise les actions et le superviseur qui évalue si le comportement produit satisfait les contraintes. L'évaluation mise en œuvre permet une réponse échelonnée au problème d'échec d'exécution. L'organisation du superviseur en modules asynchrones tient compte de la non prédictibilité des temps de planification, de sorte que le suivi d'exécution et la recherche de plans alternatifs s'exécutent sans pénaliser la réactivité du système.

L'ensemble de ces problèmes nous a amené à réaliser d'une part, une interface homme-machine évoluée et d'autre part, un système de supervision en OPS5. Le fonctionnement cyclique, en chaînage avant, de ce système de production assure une bonne réactivité pour le contrôle des processus temps-réel et pour le dialogue avec un opérateur.

Abstract

The work presented here deals with planning and execution control problems for an autonomous mobile robot, in a structured, dynamic and partially known environment. It concerns the study and implementation of a supervision system for mobile robots.

This supervisor is on top of a hierarchy of perception and action processes. It has to plan the missions specified by an operator and to control their execution while managing the dialog between the operator and the embedded system.

The planning algorithm uses a description of the environment containing both operational and geometric information. An interactive program permits an operator to create the description of the environment composed of structural elements such as walls and furniture. This description is completed by the addition of operational information in the form of a network of places, routes, landmarks and recharge stations. Places and routes include constraints on the actions of the robot as well as other information needed for planning and plan execution. A separate interactive program permits an operator to compose a surveillance mission as parallel sequences of navigation tasks and surveillance tasks, subject to constraints on energy, time, risk and position uncertainty.

The planning algorithm computes an optimal path for each navigation task according to the optimisation criterion and constraints. We introduce the notion of *efficient path* applied to a new best first search algorithm solving a multiple constraints problem. The paths determination relies on a state representation adapted to deal with environment constraints. The complexity characteristics of our algorithm are compared to those of the A* algorithm.

The control architecture enable to evaluate the current execution and to compute alternate plans concurrently. The error recovery strategy is adapted according to the evaluated failure level in order to avoid complete replanning procedures.

The planning and execution control system described in this thesis have been implemented in OPS5 on a laboratory workstation and tested using radio-modem communication with a mobile platform equipped with ultra-sonic range sensors and an active stereo vision system.

Table des matières

Chapitre 1

Introduction

L'objet des recherches menées en robotique est de construire des systèmes capables d'agir dans le monde physique. Ils doivent effectuer des tâches concrètes, prédéfinies, dans un environnement incertain et dynamique. Dans ce domaine, la robotique mobile est un sujet d'étude à part entière à propos duquel se posent des problèmes spécifiques. Elle suscite d'autre part un intérêt croissant dans la sphère industrielle, attentive au fait que de plus en plus d'applications vont voir le jour pour la robotique non manufacturière. Mais pour cela, des progrès doivent être réalisés pour accroître l'autonomie de ces robots mobiles.

Nous nous intéresserons particulièrement au problème de l'autonomie décisionnelle, caractéristique des robots de troisième génération. Notre contribution portera sur les aspects algorithmiques et architecturaux d'un système de supervision capable de planifier et contrôler l'exécution de missions définies en termes de tâches et contraintes associées. Nous présentons dans ce chapitre le problème auquel nous nous sommes confronté, notre approche pour le résoudre et les résultats que nous avons obtenus.

1.1 Cadre de l'étude et objectifs

Notre étude s'est inscrite dans le projet européen EUREKA EU 110 : MITHRA. L'objectif de ce programme est le développement de robots mobiles autonomes de télésurveillance et de première intervention, comportant un système décisionnel embarqué. Ces robots doivent évoluer dans un univers incomplètement structuré et partiellement connu où ils effectueront des missions programmées. Ils doivent pouvoir être affectés à des tâches de surveillance, des patrouilles, des interventions télé-opérées en milieu hostile.

Les objectifs de ce projet visent à dépasser les limitations des systèmes existant actuellement en milieu industriel. Ces derniers sont limités par les facteurs suivants :

1. Ils prennent en compte une description rigide de la tâche à réaliser, avec une faible interaction avec l'environnement.

2. Ils nécessitent un aménagement important de l'environnement. Tous les chemins possibles doivent y être inscrits, par un marquage magnétique ou visuel.

3. Ils ont peu ou pas d'autonomie de décision : en cas d'échec d'exécution, peu de solutions de reprise sont possibles et souvent elles nécessitent une intervention humaine.

4. Le système décisionnel est souvent distant, ce qui implique que la communication doit toujours être établie entre le robot et le poste de contrôle.

Ces différents facteurs restreignent le domaine d'application de tels systèmes à des environnements très structurés et statiques. Le projet a donc pour ambition de s'affranchir de ces limites.

Le champ d'application envisagé pour le démonstrateur est la surveillance. On souhaite pouvoir décrire simplement des missions de surveillance comme celle ci :

" Le robot doit se trouver à l'entrée de 20h à 20h15 en y allant le plus vite possible, puis il doit patrouiller dans le laboratoire 15 minutes après, pendant trois heures en s'arrêtant 1 minute à chaque poste. Pendant ce temps, s'il détecte une source de chaleur anormale alors il donne l'alarme. De 21h à 22h, lorsque le robot se trouve au poste 2, il communique les images qu'il enregistre. Pendant cette mission, le robot ne doit pas sortir du bâtiment B et doit maintenir une réserve d'énergie d'au moins 10 % de la capacité totale. "

Le degré d'abstraction élevé d'une telle description est adapté au fait que l'opérateur n'est pas nécessairement informaticien. Une *mission* sera donc décrite en termes de *tâches* de navigation et *tâches* spécifiques à l'application (dans notre cas des tâches de surveillance) auxquelles sont associées des *contraintes opérationnelles.* Ces contraintes concernent, entre autres, le temps d'exécution ou la consommation d'énergie.

Le système doit pouvoir planifier cette mission. Cela consiste à vérifier si elle est réalisable, et par là même déterminer les actions nécessaires à sa réalisation. Enfin il doit l'exécuter en essayant de respecter au mieux les contraintes définies par l'opérateur.

Le cadre de notre étude se situe donc dans le mouvement actuel qui tente de faire passer les systèmes robotiques du statut de machines dépendantes à celui de collaborateurs partiellement autonomes. Cela nécessite que le système soit apte à gérer les relations entre les objectifs qui lui sont assignés et les actions qu'il est capable d'effectuer selon sa situation dans l'environnement. Pour cela il doit le percevoir, le modéliser, planifier ses actions et en contrôler l'exécution pour réaliser des tâches décrites dans un formalisme de plus en plus proche du langage naturel.

1.2 Définition du problème

Nous intéressant aux robots mobiles, nous concentrerons notre étude sur l'aspect navigation. Vouloir effectuer une mission ayant la forme de celle donnée en exemple pose en premier le problème de la recherche d'un itinéraire passant, en temps voulu, par les localisations associées aux tâches composant la mission. Il est apparu également nécessaire de pouvoir prendre en compte d'autres contraintes portant non seulement sur la mission (paramètres de consommation d'énergie, sûreté de l'itinéraire), mais aussi sur l'environnement (accessibilité variable dans le temps, positions de points de recharge, d'amers, etc.).

La prise en compte des contraintes définies sur la mission suppose que l'on peut caractériser les différents itinéraires possibles selon leur coût pour chaque paramètre. Il nous faudra donc choisir une représentation adéquate de l'environnement pour pouvoir associer ces coûts aux itinéraires et représenter les contraintes définies sur l'environnement lui même. Notre problème est alors de construire un plan satisfaisant les différentes contraintes. Il sera ensuite de contrôler l'exécution du plan en évaluant l'écart entre les prédictions et les valeurs effectives d'exécution pour chaque paramètre que l'on souhaite contrôler.

Diverses méthodes de planification d'itinéraire ont été développées, mais il n'y en a pas, à notre connaissance, qui permette de déterminer un chemin optimal, dans un espace discret représenté par un graphe, qui prennent en compte à la fois diverses contraintes sur les tâches et des contraintes sur l'environnement telles que nous les avons définies. Nous avons restreint notre problématique sur l'aspect itinéraire, en supposant que la détermination de la trajectoire effective est réalisée par des modules hiérarchiquement inférieurs au module de planification d'itinéraire.

1.3 Notre apport

Notre apport concerne la possibilité de définir, de planifier et d'exécuter des missions satisfaisant des contraintes opérationnelles. Nous avons défini notre problème de planification comme un problème d'optimisation multi-contraintes. Les fonctions de coût, telles que nous les avons définies, conduisent à un algorithme de recherche d'itinéraire de complexité acceptable. Nous avons considéré deux types de fonctions de coût : des fonctions monotones croissantes (pour la distance, le temps d'exécution) et des fonctions croissantes par morceaux (pour l'énergie, la localisation). L'exemple du coût énergétique correspond au deuxième type. Si le robot peut se recharger à certains emplacements, la consommation d'énergie peut être considérée comme un coût strictement croissant entre deux opérations de recharge. Cela concerne donc plus généralement tous les coûts qui peuvent être réinitialisés en exécutant des actions spécifiques.

Chaque tâche est décomposée en actions correspondant à des sous-objectifs. Cette décomposition vise à permettre un comportement le plus proche possible du comportement optimal désiré. Un plan est une structure contenant l'évaluation des coûts et les contraintes à respecter pour chaque tâche et chaque action.

Le contrôleur d'exécution a pour rôle d'évaluer si le sous-système réactif produit un comportement satisfaisant. Cette évaluation doit permettre une réponse échelonnée au problème de non satisfaction des objectifs de la mission.

Nous avons donc réalisé un système, appelé *superviseur*, ayant les trois fonctions suivantes :

- dialoguer avec l'opérateur (non informaticien) pour la définition des missions affectées au système. Une mission est décrite par un ensemble de tâches qui définissent les objectifs à atteindre ;
- planifier ces missions, c'est-à-dire définir le cadre de réalisation des tâches qui constituent la mission en précisant les buts intermédiaires à atteindre et les contraintes à satisfaire ;
- contrôler l'exécution de la mission par adaptation du plan à la situation courante.

Pour réaliser ces fonctions nous avons étés amenés à proposer :

Un environnement de spécification de mission et de suivi d'exécution reposant sur la définition d'un langage de spécification de mission plus simple qu'un langage de programmation classique.

Il n'y a encore actuellement que peu de contributions concernant le développement d'interfaces homme-machine permettant la spécification de mission par définition d'objectifs (terminologie proposée dans [Gaspart, 1987]). Pour la spécification d'environnement, nous avons adopté une représentation de l'environnement en réseau d'emplacements et de routes qui permet une description aisée des missions de surveillance. Ces missions sont elles-mêmes décrites à l'aide d'un langage de spécification de mission que nous avons défini. Une mission sera décrite par une liste de tâches de navigation et une liste de tâches qui doivent s'exécuter en parallèle : surveillances, transmissions de données ou télé-opérations. L'opérateur peut associer des contraintes à chaque tâche ou à la mission dans son ensemble. Ce langage est conçu pour un opérateur non spécialiste en informatique, il ne contient pas les structures de contrôle auxquelles sont habitués les informaticiens. Néanmoins il permet de décrire des missions de surveillance suffisamment complexes.

Une méthode de planification qui prend en compte des contraintes définies sur la mission et celles définies sur l'environnement.

L'approche choisie pour planifier une mission repose sur une décomposition de la mission en un ensemble de *tâches*. Une tâche est une opération abstraite que doit effectuer le robot. Elle est abstraite dans la mesure où elle donne la nature générale de l'opération devant être accomplie sans avoir à en spécifier précisément sa réalisation. Planifier peut consister alors à remplacer une tâche abstraite par une ou plusieurs tâches plus spécifiques. Ce processus de raffinement du niveau d'abstraction est appelé *réduction de tâche.*

Pour le planificateur symbolique, une tâche irréductible est appelée *action*, alors qu'au niveau de contrôle procédural, une action peut encore être décomposée en séquences de commandes de déplacement élémentaires.

Cette planification procède en deux phases correspondant aux deux types de tâches définies par le langage de spécification. La planification des tâches de navigation repose sur une recherche dans un graphe d'état guidée par une heuristique. Cette recherche tient compte des contraintes imposées sur les tâches et des contraintes sur l'environnement : accessibilité, localisation des emplacements de recharge. Une analyse de la complexité permet d'évaluer les performances de notre algorithme. La planification des tâches parallèles consiste à déterminer les intervalles d'activité des sous-systèmes chargés de réaliser ces tâches. Le résultat global de la planification produit un arbre de couples (coûts, contraintes) d'une part et d'autre part un graphe d'exécution dont les nœuds sont les intervalles de temps prévus pour la réalisation des tâches et des actions.

Une architecture de contrôle qui permet la co-existence d'un niveau symbolique pour le contrôle d'exécution des tâches et d'un niveau réactif pour la réalisation des actions de navigation de telle sorte que le processus de planification, étant essentiellement non déterministe, ne pénalise pas les performances du système complet.

Nous nous sommes principalement intéressés à l'évaluation et à la récupération des erreurs concernant la navigation. Nous proposons une méthode de replanification hiérarchique dont l'effet est d'adapter le coût de la planification à l'importance de l'erreur estimée par rapport au plan. Nous détaillons également une stratégie d'évaluation de l'exécution des actions de navigation afin de prédire l'échec d'exécution par rapport aux contraintes imposées sur la tâche. Le contrôleur est organisé en différents modules fonctionnant de manière asynchrone, ainsi la replanification peut se faire indépendamment de l'évaluation périodique de l'état du véhicule.

1.4 Mise en œuvre et résultats

Le superviseur a été réalisé avec un système à base de règles asynchrone. Cette approche permet d'une part une description déclarative du contrôle et d'autre part le fonctionnement cyclique du mécanisme d'inférence donne la possibilité d'interrompre et de reprendre toute séquence de calcul et assure au superviseur une bonne réactivité aux événements provenant des sous-systèmes ou de l'interface homme-machine.

Nos expérimentations simulées ou réelles ont toujours eu pour cadre un environnement intérieur à un bâtiment. Cette restriction provient du fait que nous ne pouvons faire d'expérience réelle avec notre robot que dans ce type d'environnement.

1.5 Organisation du rapport

Une étude bibliographie des méthodes de planification et de contrôle d'exécution est présentée dans le chapitre 2. Elle nous permettra de situer notre approche face aux

autres systèmes existants, traitant également de plans de haut niveau, et de dégager les travaux sur lesquels nous nous sommes appuyés.

Le chapitre 3 situe l'architecture de contrôle dans laquelle s'inscrit notre système par rapport aux divers types d'architecture existants. Cette description nous permettra de préciser les capacités d'action et de perception des sous-systèmes dont nous voulons superviser le fonctionnement.

La conception de l'interface de notre système avec un opérateur humain est présentée dans le chapitre 4. L'état de l'art dans le domaine de la télé-robotique [Sheridan, 1992] montre que les systèmes de navigation autonomes nécessitent la présence d'un opérateur humain dans les trois phases d'exploitation suivante : spécification d'environnement, spécification de mission et suivi d'exécution. Notre interface est constituée de modules dédiés à chacune de ces phases.

Les chapitres 3 et 4 nous ont permi de définir les interfaces logicielles entre le superviseur et les sous-systèmes d'action et de perception d'une part et l'interface homme-machine d'autre part. Le chapitre 5 expose la partie du superviseur qui réalise la planification des missions.

Le chapitre 6 présente le contrôle d'exécution basé sur les structures produites par la planification et sur les informations d'exécution fournies par les sous-systèmes.

Chapitre 2

Le problème de planification et de contrôle d'exécution

Cette étude bibliographique se compose de deux parties, une centrée sur la planification, l'autre sur l'aspect contrôle d'exécution. La planification sera exposée dans la perspective traditionnelle des recherches en Intelligence Artificielle (section 2.1). Le problème de la planification générale est d'abord présenté, suivi d'un bref historique des principales méthodes de planification existantes. Un exposé plus complet de ces méthodes pourra être trouvé dans [Georgeff, 1987]. La complexité du problème de planification nous amènera ensuite à étudier des approches spécifiques au problème de recherche d'itinéraire.

Dans le cadre défini pour la planification, le contrôle d'exécution consiste, pour l'état courant du système, à choisir une action exécutable parmi un ensemble défini à partir du plan. La section 2.2 présente le problème de contrôle d'exécution d'un plan et décrit les approches principales, à partir d'exemples de systèmes.

Les domaines de la planification et du contrôle d'exécution ont fait l'objet de recherches propres, en intelligence artificielle et en automatique. Ces domaines s'ignoraient mutuellement ou supposaient un fonctionnement fortement découplé entre planification et contrôle d'exécution. Cette séparation est apparue de plus en plus arbitraire et inefficace, et nous verrons plusieurs approches différentes qui cherchent à les intégrer.

2.1 Planification

La planification a pour objet de produire un guide opératoire permettant de passer d'une situation donnée dans un domaine (ou environnement) réel ou abstrait, à une situation désirée. Dans le cas d'un robot, la situation désirée pourra être définie comme une liste d'objectifs à réaliser.

Ce problème de planification à été envisagé selon deux approches :

- une approche indépendante du domaine ayant pour ambition de dégager des méthodes de planification générales (*planification générale, domain independent*

planning),

- une approche qui, au contraire, tire parti de la structure particulière d'un domaine et qui utilise des connaissances spécifiques à celui ci, pour résoudre le problème de planification (*planification spécifique, domain dependant planning*). C'est le cas des système de planification de mouvements pour la navigation [Latombe, 1990; Laumond *et al.*, 1989] ou les robots manipulateurs [Lozano-Pérez, 1987].

Si les planificateurs spécifiques sont souvent plus efficaces que les planificateurs généraux, ils sont cependant limités quand à l'expressivité des objectifs que l'on peut définir et des interactions possibles avec l'environnement. Pour cette raison, certains systèmes combinent au sein d'une architecture de contrôle adaptée plusieurs types de planificateurs, dans une hiérarchie de niveaux d'abstraction allant du plus général au plus spécialisé (cet aspect particulier est développé au chapitre 3). Il apparaît donc comme nécessaire, pour étudier le problème de la planification de missions pour un robot mobile, d'étudier dans un premier temps ce problème sous sa forme la plus générale. Ceci permet de définir le cadre théorique dans lequel des solutions plus liées aux applications pourront être envisagées et permet de mieux préciser le niveau d'abstraction auquel nous nous situerons par la suite.

2.1.1 L'approche symbolique

La théorie du contrôle fournit des méthodes pour analyser et synthétiser des systèmes de contrôle pour une grande variété de problèmes modélisés en termes d'équations aux différences finies, systèmes d'équations différentielles, et d'autres formalismes pour modéliser des systèmes dynamiques. Cependant, pour des domaines fortement non linéaires, il a semblé plus adéquat de représenter le système dynamique sous-jacent en utilisant les techniques reposant sur la logique du premier ordre, la représentation d'événements, les processus et les relations causales. Par exemple, la représentation basée sur la notion d'événement peut faciliter l'expression de l'information incomplète; elle accepte la construction incrémentale de modèle, et souvent correspond mieux que les équations différentielles à la description intuitive de relations causales.

Dans le cadre général de la planification, on établit un modèle causal d'événements, avec un sous-ensemble particulier d'événements appelés *actions*, contrôlés par le robot. Le robot peut agir directement et connaître les effets associés aux actions mais il ne peut influer sur les autres événements que de manière indirecte, via les relations causales que ces événements ont avec les actions. Le robot a également des objectifs décrivant les propriétés voulues de son comportement, donnés sous la forme d'événements. La planification est donc le processus d'assemblage de ces actions de base dans un objet composite appelé *plan* conçu pour remplir les objectifs.

Le problème central de la planification est celui du raisonnement à propos des effets ou des conséquences potentielles des actions. Une tâche importante de raisonnement

consiste à déterminer si une certaine propriété doit être vérifiée à un instant donné, après, ou pendant l'exécution du plan. En résumé, la planification cherche à :

1. prédire l'évolution du monde sur la base de modèle d'événements,

2. utiliser ces prédictions pour choisir et ordonner les actions qui doivent réaliser les objectifs.

Pour cela, on caractérisera le problème à résoudre par la description d'un *état initial* et d'un *état but.* Un état est une représentation abstraite du monde réel. Il peut être décrit par un ensemble de formules logiques atomiques. La solution du problème consiste alors à construire un plan permettant de passer de l'état initial à l'état but. Pour cela le planificateur utilise des *opérateurs de changement d'état.*

Un *opérateur de changement d'état* désigne un type d'action. C'est une description générique d'une action et l'instanciation d'un opérateur est alors une action. Les opérateurs de changement d'état sont fournis au planificateur, ils sont spécifiques à un domaine d'application.

Les actions planifiées peuvent être totalement ou partiellement ordonnées. Un plan *linéaire* défini un ordre total sur les actions alors que dans un plan *non linéaire* l'ordre d'exécution des actions n'est pas totalement spécifié, les actions sont seulement partiellement ordonnées et peuvent éventuellement s'exécuter en parallèle si le système robotique le permet. La section 2.1.2 présente les premiers planificateurs linéaires et dans la section 2.1.3 sont décrits les planificateurs non linéaires.

2.1.2 Planificateurs linéaires

Les deux premiers planificateurs GPS [Newell et Simon, 1963] et STRIPS [Fikes et Nilsson., 1971] ont introduit les techniques qui sont la base de tous les planificateurs généraux qui ont suivi.

GPS a introduit la notion d'*opérateur* et d'analyse *moyens-buts* (means-ends) par définition de sous-buts. La méthode de planification repose sur le principe consistant à identifier la différence entre l'état initial et l'état final, et ainsi de tenter réduire cette différence par l'application d'un opérateur qui satisfait certains buts. Les préconditions de cet opérateur définissent de nouveaux sous-buts. Quand un sous-but ne peut être atteint, un mécanisme de cheminement arrière (backtracking) est mis en œuvre. Des tables de différence, adaptées à chaque application, sont utilisées pour guider le planificateur dans le choix d'un opérateur.

STRIPS, tout en reprenant l'analyse moyens buts, a contribué à la modélisation des actions. Chaque action ou opérateur, est modélisé par une précondition, une liste d'ajouts et une liste de retraits. Un état est décrit par une liste de formules logiques et la précondition d'un opérateur doit être vérifiée pour pouvoir exécuter l'action (l'opérateur instancié) dans cet état. Le nouvel état atteint est alors décrit en supprimant de la description de l'état courant les formules contenues dans la liste de retraits et en ajoutant celles de la liste d'ajouts.

Un des problèmes de STRIPS est qu'il repose sur l'hypothèse de linéarité supposant que des sous-buts peuvent être atteints indépendamment, ce qui n'est pas toujours le cas.

Sussman met en évidence le problème des buts interdépendants connu sous le nom de "anomalie de Sussman". Il introduit pour résourde cela, dans le système HACKER [Sussman, 1973], la notion d'*intervalle de protection* (*protection interval*). Cette structure est destinée d'une part à empêcher qu'un sous-but réalisé soit détruit et d'autre part à détecter les interactions. Cependant, la seule manière dont HACKER peut tenter de résoudre les interactions détectées est de revenir à un point ou un autre ordonnancement des buts peut être tenté.

D'autres systèmes sont apparus avec l'ambition de résoudre de manière élégante ce problème. Dans WARPLAN [Warren, 1974], une action qui viole une protection est placée, pas à pas, de plus en plus tôt dans le plan, jusqu'à ce qu'il n'y ait plus d'interaction. Cette technique connue sous le nom de *régression* est applicable non seulement aux actions mais également aux sous-buts. Tate introduit dans INTERPLAN [Tate, 1974] une structure appelée *tick list* utilisée pour mémoriser le lien entre l'effet d'une action et les préconditions d'une action suivante et faciliter ainsi la détection des interactions.

2.1.3 Planificateurs non linéaires

Les planificateurs linéaires, en développant un plan de manière strictement séquentielle, sont sur-contraints. Le principe de base des planificateurs non linéaires[1] est de retarder la décision d'ordonnancement entre des actions jusqu'à ce que cette décision soit nécessaire pour résoudre un conflit (*stratégie d'engagement minimum*).

Le premier planificateur non linéaire est NOAH [Sacerdoti, 1975]. Dans ce système, un plan partiel est représenté par un réseau procédural (*procedural net*) et une table des effets multiples (*TOME: Table Of Multiple Effects*). Cette table est utilisée pour découvrir les interactions, elle est similaire à la structure utilisée dans INTERPLAN. Quand une décision d'ordonnancement doit être prise entre deux actions, NOAH en choisi une à l'aide d'heuristiques. Ne pouvant remettre ce choix en cause, NOAH peut échouer même si une solution existe.

Basé sur NOAH, Tate propose avec NONLIN [Tate, 1976] une structure de contrôle de la planification permettant d'évaluer plusieurs alternatives, utilisant le retour-arrière en cas d'échec. Il peut ainsi trouver des plans pour lesquels NOAH restait bloqué, mais sa complétude n'est pas prouvée.

Il faut attendre MOLGEN [Stefik, 1981] pour voir de nouveaux résultats en planification. Ce système fait pour la première fois de la définition de contraintes la technique centrale de résolution des interactions. Les interactions sont décrites par des spécifications partielles de propriétés que les sous-problèmes doivent satisfaire. Poser des contraintes sur ces propriétés permet d'assurer que des propriétés ne seront pas détruites pendant la résolution.

Chapman présente avec TWEAK [Chapman, 1987], un planificateur avec contraintes possédant une procédure complète, de complexité polynomiale pour ré-

1. Le terme *partial order planner* est également employé.

soudre le problème de génération de plan. TWEAK étant basé sur un modèle formel, il est possible d'analyser ses propriétés. Chapman a ainsi prouvé que la planification est semi-décidable, i.e. si un plan existe TWEAK le trouvera mais si il n'existe pas TWEAK peut éventuellement ne jamais s'arrêter. Il s'avère d'autre part que l'efficacité de l'algorithme repose sur un formalisme trop pauvre pour qu'un tel planificateur soit mis en œuvre sur des applications concrètes. Idéalement il faudrait qu'un tel formalisme puisse prendre en compte simultanément les éléments suivants :

- la qualification de l'action : décrire complètement les préconditions nécessaires à l'exécution de l'action. Dans TWEAK les préconditions et post-conditions doivent être atomiques.
- la ramification de l'action : décrire complètement les effets de l'exécution d'une action. Ces effets peuvent être implicites ou varier selon le contexte, c'est le cas des actions conditionnelles [Christensen et Grove, 1991]. Ils peuvent êtres combinés entre plusieurs actions simultanées.
- la persistance : tenir compte d'une manière efficace de tout ce qui ne change pas.
- le temps : pour décrire la durée des actions, les contraintes temporelles sur les effets des actions, sur l'environnement ou sur les buts.
- les ressources : elles peuvent être partagées ou consommées.
- un critère d'optimalité dans l'exécution du plan, par exemple en temps d'exécution ou en utilisation des ressources.

À l'heure actuelle aucun planificateur prouvé correct ne repose sur un formalisme suffisement expressif pour prendre en compte ces éléments. Des résultats théoriques récents expliquent ces limitations.

2.1.4 Complexité des algorithmes de planification généraux

Bylander dans [Bylander, 1991] précise les limites théoriques indiquant pourquoi aucun planificateur actuel en logique propositionnelle du type de STRIPS ne possède en même temps la plupart de toutes ces extensions. Il étudie quelles restrictions appliquer pour que des problèmes de planification en logique propositionnelle soient solubles formellement. Ces restrictions concernent le nombre de préconditions et de post-conditions. Bylander montre que la planification en logique propositionnelle est un problème P-ESPACE complet même si chaque opérateur est limité à une post-condition, avec un nombre quelconque de préconditions. C'est un problème P-ESPACE complet même si chaque opérateur est limité à deux préconditions positives et deux post-conditions. Ce problème est NP-dur si chaque opérateur est restreint à une précondition positive et deux post-conditions (voir aussi [Gupta et Nau, 1991]). Il est NP-complet si les opérateurs sont restreints à des post-conditions positives, même si les opérateurs sont restreints à une précondition et une post-condition positive. Il

est polynomial si chaque opérateur est restreint à des préconditions positives et une post-condition. Mais la plupart des problèmes de planification imposent de violer ces restrictions.

Ainsi, on peut considérer de manière pessimiste la perspective d'un unique algorithme de planification indépendant du domaine. Il est donc plus fructueux d'adopter des algorithmes différents pour des types de problèmes de planification différents. Il y a un équilibre difficile à trouver entre l'expressivité d'un langage de planification et la difficulté de résolution d'un plan décrit dans ce langage. L'expressivité autorisée par un langage entraîne un accroissement du nombre de choix possibles à chaque étape de planification. De plus l'existence d'un critère de validité tel que l'a défini Chapman (*truth criterion*) efficace et prouvé correct est moins probable. Cependant, une meilleure expressivité permet une description plus fine et la "sur-spécification" est plus facilement évitée, et ainsi le nombre de retours en cas d'échec peut être réduit. De même, une plus grande expressivité permet d'encoder plus de connaissance pour chaque étape de planification ainsi les choix effectués peuvent-ils être plus informés. Mais relâcher les exigences de correction produit des planificateurs basés sur des heuristiques qui peuvent produire des plans non exécutables. Beaucoup de systèmes sont dans cette catégorie, par exemple [Dean *et al.*, 1988].

Comme nous l'avons exposé dans le chapitre introductif, nous avons choisi de restreindre notre problème de planification en un problème de recherche d'itinéraire. Si l'on voulait formuler ce problème selon l'approche générale, il serait possible d'introduire un opérateur *Aller* qui serait dépendant d'une position de départ et d'une position d'arrivée. Un tel opérateur serait défini en fonction de ces deux positions et de la position du robot. Pour conserver une complexité algorithmique polynomiale, il faudrait définir autant d'opérateurs *Aller* qu'il y a de chemins élémentaires possibles, et alors il n'y aurait aucun gain réel en complexité.

Dans la section suivante nous abordons les méthodes spécifiques à la planification d'itinéraires. Ces méthodes traitent implicitement les problèmes de qualification, les préconditions et les post-conditions d'actions de navigation étant triviales. Par contre elles permettent de prendre en compte des critères d'optimalité.

2.1.5 Planification d'itinéraires

On s'intéresse ici à la génération automatique d'un *itinéraire* (ou *chemin*) pour un robot mobile autonome. Cet itinéraire doit être optimal selon un critère à minimiser. Ce critère est défini par l'utilisateur; ce peut être par exemple la distance, le temps, la consommation d'énergie ou toute autre contrainte opérationnelle.

Les différentes méthodes de planification d'itinéraire qu'on trouve dans la littérature diffèrent surtout par le type d'environnement qu'elles envisagent et leurs façons de le modéliser. Mais la recherche du chemin optimal se fait presque toujours de la même manière: l'espace est divisé en différentes régions, qui sont connectées entre elles par des liens. On établit alors un graphe dont les nœuds sont des régions et les arcs les liens entre ces régions. Nous nous intéressons dans un premier temps aux méthodes de planification prenant en compte des environnements 2D intra-muros, c'est à dire

constitués typiquement de salles, de couloirs, de portes, et d'obstacles schématisés par des surfaces polygonales.

Nous ferons l'hypothèse que le planificateur ne possède pas d'information géométrique sur l'environnement mais un graphe défini par un réseau de routes et d'emplacements. Nous le distinguerons ainsi des planificateurs de *trajectoires* conçus pour déterminer une trajectoire évitant les obstacles.

Recherche d'un chemin optimal

Pour trouver le chemin optimal, deux méthodes sont principalement utilisées. La première part d'une représentation matricielle du graphe et permet une résolution algébrique du problème du plus court chemin, en se plaçant dans une algèbre des chemins ; la seconde méthode est fondée sur un parcours du graphe guidé par une heuristique.

Les algèbres de chemins : Gondran et Minoux ont montré dans [Gondran et Minoux, 1985] que les problèmes de cheminement dans un graphe peuvent être reformulés et résolus dans les structures algébriques appelées *algèbres de chemins.* Ces structures ont un rôle unificateur intéressant puisqu'elles permettent d'appliquer un même algorithme à des problèmes de cheminement de natures différentes.

Soit un graphe de N sommets, $G = \{X, U\}$ avec $X = \{1, 2, \ldots, N\}$. Soit l_{ij} la longueur de l'arc (i, j) si $(i, j) \in U$. Définissons $\pi^*(i)$ comme la longueur minimum des chemins de 1 à i ; en particulier $\pi^*(1) = 0$.

L'équation d'optimalité s'écrit :

$$\pi(i) = \min_j(\pi(j) + l_{ji})$$

en posant $l_{ij} = +\infty$ pour les arcs manquants du graphe. La valeur des plus courts chemins sera alors solution d'un système d'équations dont l'expression matricielle est

$$\pi = \pi * L \oplus B$$

où L est la matrice des longueurs des arcs du graphe, B est le vecteur défini par

$$B = \begin{bmatrix} 0 \\ +\infty \\ \vdots \\ +\infty \end{bmatrix}$$

Les opérateurs $\oplus$ et $*$ définis par

$$a \oplus b = min(a, b) \text{ pour } a, b \in I\!R \cup \{+\infty\}$$

$$a * b = a + b \text{ pour } a, b \in I\!R \cup \{+\infty\}$$

sont étendus sur les matrices. L'équation matricielle peut être résolue par exemple par la méthode de Jacobi. Les auteurs montrent que plus généralement de nombreux problèmes de cheminement se ramènent à la résolution d'un système d'équations linéaires de type point fixe dans la classe des dioïdes.[2]

Parcours de graphe guidé par une heuristique: Étant donné un état initial x_0 et un état final $x_\star$, le problème de trouver une séquence d'actions passant de x_0 à $x_\star$ peut être décrite comme la recherche d'un chemin dans un graphe d'état. Le graphe d'état est un graphe dont les nœuds sont les états et les arcs correspondent aux transformations possibles entre états. Cette approche permet de trouver un plan minimal mais exclut de produire des plans contenant des actions à exécuter en parallèle.

L'algorithme de parcours de graphe A* [Hart *et al.*, 1968; Nilsson, 1980] est une variante de recherche du *meilleur d'abord.* Il suppose que l'on peut associer un *coût* pour atteindre l'état final à partir de l'état courant et permet alors de trouver un chemin de coût minimum entre un état de départ et un (ensemble) de but(s) sur un graphe décrivant un espace d'états donné. Cet algorithme a souvent été utilisé pour la planification de chemins [Lozano-Perez et Wesley, 1979; Arkin, 1987; Fennema, 1991; Fraichard, 1992]. Dans le cas le plus simple, le graphe d'état est identique au graphe décrivant la topologie de l'environnement ; les nœuds correspondant aux emplacements à atteindre et les arcs correspondant aux mouvements possibles à partir d'un emplacement.

Le graphe d'états est spécifié implicitement par l'état de départ, une procédure de génération des successeurs possibles d'un état et d'une fonction qui reconnaisse le ou les états buts. L'algorithme A* gère deux listes de nœuds OPEN et CLOSED. OPEN contient les nœuds dont les successeurs n'ont pas encore été générés et CLOSED les nœuds dont les successeurs ont étés générés. Pour un nœud n de OPEN, $g(n)$ est le coût du meilleur chemin courant du nœud de départ à n, $h(n)$ est une estimation heuristique du coût du plus court chemin entre n et un état but et $f(n) = g(n) + h(n)$ est le coût global du nœud n. A chaque itération, A* choisi le nœud de coût f minimum dans OPEN, génère ses successeurs, insère n dans CLOSED et ses successeurs dans OPEN. Quand un nœud but n est choisi, A* se termine avec n comme solution.

Si l'estimation du coût pour atteindre le but ne sur-estime jamais le coût réel d'un chemin vers le but, alors il est garanti que l'algorithme trouvera une solution optimale [Nilsson, 1980; Pearl, 1981]. La complexité d'un tel algorithme est exponentielle dans le pire des cas et polynomiale en moyenne pour les domaines bien conditionnés. Nous reviendrons sur ce point dans le chapitre concernant la planification.

Planification en espace de recherche borné L'algorithme RTA* (Real-Time A*) de Korf [Korf, 1990] est une variante de l'algorithme A* dans lequel la profondeur de recherche est bornée. Cet algorithme reprend une méthode similaire à l'élagage alpha-bêta. Il détermine l'heuristique associée à un nœud en fonction de la situation courante

2. Un dioïde est un ensemble muni des deux opérations $\oplus$ et $*$ le dotant de la structure de semi-anneau pour lequel $\oplus$ n'est pas simplifiable.

sans tenir compte du nœud de départ. L'algorithme permet des phases de planification et d'exécution entrelacées et dans cette situation prend des décisions localement optimales avec la garantie de trouver une solution si elle existe. La limitation de l'espace de recherche vise à permettre une planification en temps borné. Cette limitation entraîne d'une part que le chemin n'est pas globalement optimal et d'autre part que les situations d'échec (non satisfaction des contraintes) peuvent être découvertes trop tardivement ce qui ne permet pas de planifier efficacement des actions de réinitialisation (recharge, relocalisation, etc.).

Après avoir précisé la problématique de la planification et s'être focalisé sur la planification d'itinéraire, nous abordons dans la section suivante les techniques de contrôle d'exécution de plan.

2.2 Contrôle d'exécution

Tout système qui opère dans un environnement modérément complexe et imprévisible doit répondre dynamiquement aux changements de son environnement. Pour cela le système doit vérifier que ses actions ont l'effet désiré et dans le cas contraire modifier son plan. Pour des tâches simples dans des domaines soigneusement préparés, le comportement non-réactif est acceptable ; ce n'est plus le cas pour des robots autonomes dans un domaine non contraint.

Nous distinguerons trois approches principales. La première communément appelée raisonnement révisable qui repose sur une approche logique, déductive de la notion de connaissance. Dans le cas du système FIGARO [Barrouil et Mampey, 1990], cette révision agit sur l'arborescence du plan. La seconde approche appelée ordonnancement réactif (reactive scheduling) se base sur la détection de conflits introduits dans l'ordonnancement des actions, à partir des changements dans l'environnement. La troisième approche, appelée planification réactive (reactive planning) se base plutôt sur le fait qu'un module d'exécution dispose de schémas d'actions à appliquer suivant sa connaissance de l'environnement. Ces schémas peuvent être prédéfinis ou produits par un planificateur.

2.2.1 Raisonnement révisable

Les systèmes de planification de haut niveau (planification d'itinéraire ou de mission) sont généralement des systèmes lourds, coûteux en temps de calcul. C'est pourquoi, en cas de replanification, on a tout intérêt à disposer de systèmes capables de réviser les raisonnements de façon minimale, c'est-à-dire en ne modifiant que les parties affectées par les changements de données. La conception de systèmes de planification intégrant des principes liés à la théorie des raisonnements révisables est l'un des axes de recherche du programme FIGARO du CERT. Considérons la figure 2.1 qui illustre un plan généré par ce système [Barrouil et Mampey, 1990]:

Un plan est un arbre dont chaque nœud est un but, ou un sous-but, exécutable directement par le robot s'il s'agit d'une action élémentaire (ex. stopper), ou bien dé-

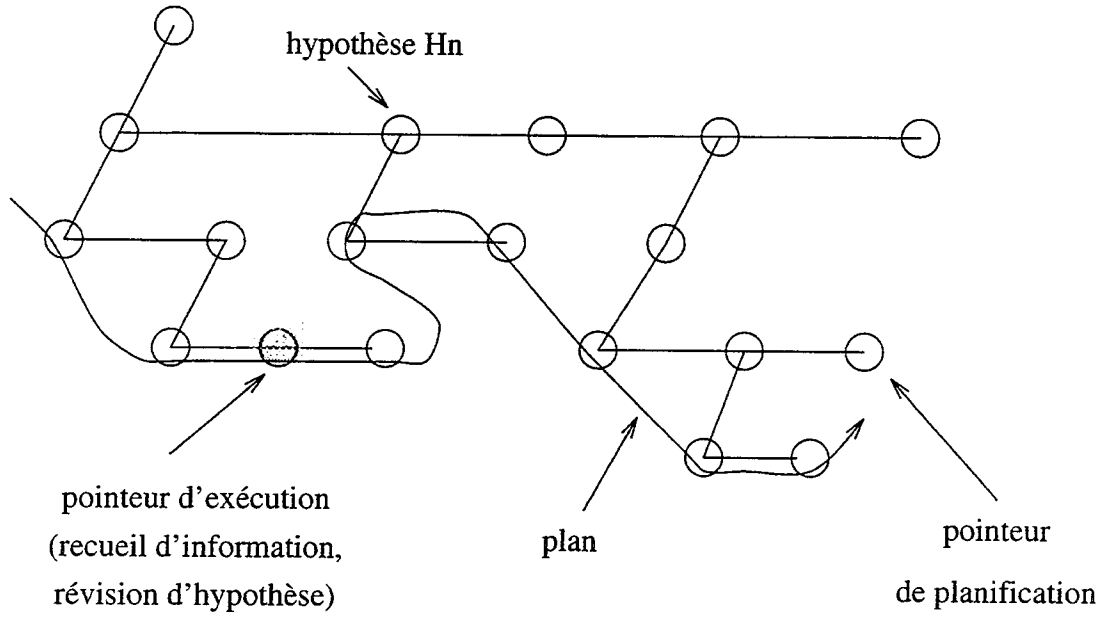


FIG. 2.1 - *Allure des plans générés par FIGARO*

composable en d'autres sous-buts. A chaque nœud est associé un ensemble d'hypothèses ou de conditions nécessaires à la réalisation du sous-but correspondant. Si l'action à réaliser consiste à ouvrir une porte, cela suppose qu'elle ne soit pas fermée à clef.

Ainsi, pour qu'un nœud soit sélectionné dans le plan, il faut que les hypothèses qui lui sont associées soient vérifiées relativement à l'ensemble des connaissances que possède le robot sur son environnement. La résolution d'un objectif (c'est-à-dire sa décomposition en sous-buts) dépend donc de la vérification de certaines hypothèses.

Il s'agit, par exemple, de sortir d'une pièce possédant deux portes A et B, le robot choisira la porte B s'il sait que la porte A est fermée à clef et vice versa. Si maintenant les deux portes ne sont pas verrouillées, il devra disposer d'une heuristique (la distance à parcourir...) qui lui permette de faire son choix. L'état de l'environnement évoluant, le plan n'est pas généré en une seule fois, mais au fur et à mesure de la progression du robot, et avec une légère avance afin de se référer à un ensemble de connaissances le plus actualisé possible.

Grâce à son système de perception, le robot est susceptible de se rendre compte que l'une des hypothèses n'est pas vérifiée. Il s'agit alors pour le système de contrôle de lancer une phase de révision du plan, en tenant compte du nouvel ensemble de connaissances (ou état de connaissances) que le robot a de son environnement. La première étape du mécanisme consiste d'abord à identifier le point de reprise minimum, c'est le nœud du plan à partir duquel on pourra relancer la planification à moindres frais. Dans le cadre du problème lié à la navigation d'un robot mobile, deux déplacements distincts sont toujours indépendants. En effet, si le robot doit aller d'un point A à un point B puis ensuite à un point C, la façon dont il ira du point B au point C ne dépend pas de la façon dont il est allé de A à B. Dans ce cas, il est possible à ce jour de réaliser des mécanismes de révision permettant de concevoir un système de replanification efficace.

En revanche, si deux sous-buts sont liés, c'est-à-dire que la résolution de l'un influe sur la résolution de l'autre, alors toute la planification réalisée en aval du point de reprise est perdue car on suppose une dépendance des objectifs résolus dans l'ordre chronologique de leur traitement. Il faudrait, pour obtenir un système de planification capable de faire des retours économes et pertinents, pouvoir reconnaître au fur et à mesure de leur création la dépendance entre sous-objectifs et préparer à l'avance les mécanismes de reprise adéquats.

2.2.2 Ordonnancement réactif

Pour la plupart des planificateurs, comme celui envisagé dans [Fikes et Nilsson., 1971], les règles de planification génèrent la liste complète des actions avant de commencer leur exécution. Le plan généré est décomposé en actions directement exécutables par les sous-systèmes concernés. L'exécution de ces actions est contrôlée pour vérifier qu'elles produisent l'effet désiré. Dans le cas contraire, le contrôle est redonné à l'ensemble de règles qui peuvent modifier le plan de façon appropriée.

Parmi les techniques développées pour suivre l'exécution de plans et replanifier en cas d'échec, la plus courante est de maintenir une description explicite des conditions que doit respecter l'exécution du plan pour assurer que le but sera correctement atteint. Pendant l'exécution, ces conditions sont testées régulièrement.

Le planificateur STRIPS n'est pas réactif: le système PLANEX [Fikes *et al.*, 1972] lui a été adjoint pour exécuter le plan fourni et pour en contrôler l'exécution. Au lieu de prendre en compte simplement une liste d'actions, il utilise une structure de données appelée table triangulaire qui code les actions avec leurs préconditions et leurs effets. Le plan linéaire construit par STRIPS est transformé en un graphe complet non-orienté dont chaque nœud est une action et que l'on parcour grâce à la table triangulaire. Ainsi, au lieu de recalculer un plan après chaque action, PLANEX utilise cette table pour relancer l'exécution à partir d'un opérateur antérieur ou postérieur au point courant. Si aucune liste de clauses d'une case de la table ne correspond à la situation, alors le contrôle est redonné à STRIPS qui recalcule un nouveau plan.

Les techniques de réordonnancement des actions ou de replanification sont précisées dans [Ow *et al.*, 1988]. Elles sont fondées sur les principes suivants:

- la réactivité doit être focalisée sur la reconnaissance et l'analyse de conflits sur les contraintes introduites dans l'ordonnancement courant.

- les deux sources majeures de conflits viennent d'un ordonnancement inadéquat ou d'un dépassement de ressources.

- le contrôle du réordonnancement doit être opportuniste: il analyse en permanence les conflits possibles pour déterminer quelle action de réordonnancement opérer.

- la nature fortement couplée des décisions d'ordonnancement fait qu'il est difficile de prédire les effets de bord d'une action d'ordonnancement.

Dans le système OPIS [Ow *et al.*, 1988] le schéma standard de réordonnancement est :

1. détection des conflits introduits dans l'ordonnancement des actions, à partir des changements dans l'environnement
2. choix d'une méthode d'ordonnancement pour résoudre le conflit
3. application de cette méthode.

Chaque stratégie de résolution de conflit peut être caractérisée par :

- le type de conflit traité
- la qualité de la révision
- le taux de perturbation sur le plan
- l'efficacité de la méthode, ou rapidité de la méthode de révision.

Dans OPIS ces critères sont arbitrairement synthétisés en un facteur de pondération de chaque méthode.

Les méthodes les plus coûteuses cherchent parmi un ensemble d'affectations possibles de ressources celle qui sera optimale ; cette recherche peut être limitée dans le temps et focalisée sur un sous ensemble de ressources. Les méthodes les plus simples effectuent un décalage dans le futur de certaines actions et des affectations de ressources associées. A partir de l'analyse des conflits, on obtient un arbre de décision qui permet en peu d'étapes de déterminer quelle méthode appliquer.

2.2.3 Planification réactive

Un des problèmes de la rigidité des méthodes précédentes provient du fait que des informations qui permettraient de mieux atteindre les buts peuvent n'être disponibles que lors de l'exécution. On peut résoudre ce problème de plusieurs façons :

Produire des plans extrêmement conditionnels dont la plupart des branches ne seront jamais utilisées. Laisser réaliser les actions de bas niveau par des primitives fixes qui sont elles-mêmes très conditionnelles [Nilsson, 1984]. En fait cette approche repousse le problème vers un niveau inférieur.

Une autre méthode consiste à entrelacer planification et exécution ce qui permet de différer des décisions jusqu'au moment où il faut réellement choisir. On a pu acquérir ainsi plus d'informations et les choix sont donc a priori meilleurs.

Le fait que la planification cherche une solution dans un espace de taille éventuellement très importante est en contradiction avec la contrainte de réactivité temporelle. Le système doit être capable de déceler les changements critiques de son environnement dans un intervalle de temps bref. M. P. Georgeff [Georgeff et Lansky, 1987] remarque que la plupart des systèmes existants sont trop contraints par la phase de planification, sans se soucier de l'urgence des informations d'origine externe (événements). Ces

systèmes passent autant de temps qu'il est nécessaire pour produire un plan avant d'effectuer des actions d'urgence. Il leur manque la possibilité de décider entre prendre le temps de planifier et agir immédiatement. Il faut non seulement pouvoir décider de différer la planification de buts et l'exécution des plans associés mais aussi de les annuler et de replanifier en fonction de nouveaux buts complètement différents. Pour cela, il faut définir une priorité dans les buts en différenciant les buts spécifiques des contraintes de comportement à respecter invariablement.

Le système PRS

Les systèmes de planification traditionnels ne construisent leurs plans qu'à partir de la définition d'un ensemble de primitives d'actions exécutables par le robot. Il faut aussi pouvoir intégrer des plans obtenus par des descriptions explicites, ou appris en téléopération. Cela suppose que le système possède un ensemble très grand de connaissances procédurales [Georgeff et Lansky, 1986] et que l'on puisse les décrire d'une façon cohérente avec le reste des connaissances. C'est l'ambition du système Procédural Reasoning System (PRS) de Georgeff. Ce système se compose de quatre parties :

- une base de faits contenant les connaissances que le système a du monde et de son propre état,
- une bibliothèque de plans appelés *aires de connaissances* (Knowledge Area - KA),
- un graphe d'*intentions*,
- un interpréteur.

Une KA est une méthode pour réaliser un but et une intention est un couple formé d'un but et d'une KA instanciée qui doit le réaliser. L'interpréteur sélectionne la première intention du graphe d'intentions et passe d'un état de sa KA à un autre. Quand il arrive à un état qui le fait passer à une autre KA, une nouvelle intention est créée et elle est ajoutée au graphe. Si à une étape, une KA est en état d'échec, l'interpréteur tente d'en instancier une autre, si aucune n'est applicable l'intention échoue. Une KA donnée est essayée une seule fois pour une intention donnée. PRS utilise des *méta KA* pour contrôler chaque étape d'exécution d'une intention.

Le principe de fonctionnement de PRS consiste à exécuter une partie des sous-buts avant d'aller plus en avant dans l'élaboration du plan. Pour représenter ceci, PRS dispose de plusieurs opérateurs temporels[3]. Si p est une description d'état, alors :

- !p est vrai sur une séquence d'états si p est vrai sur le dernier état de la séquence. Résoudre !p revient à connaître la séquence de comportements qui aboutira à p,
- ?p est vrai sur une séquence si p est vrai sur le premier état de la séquence. Résoudre ?p revient à tester p, quelle que soit la séquence qui va suivre,
- #p est vrai sur une séquence si p reste vrai durant le déroulement de la séquence.

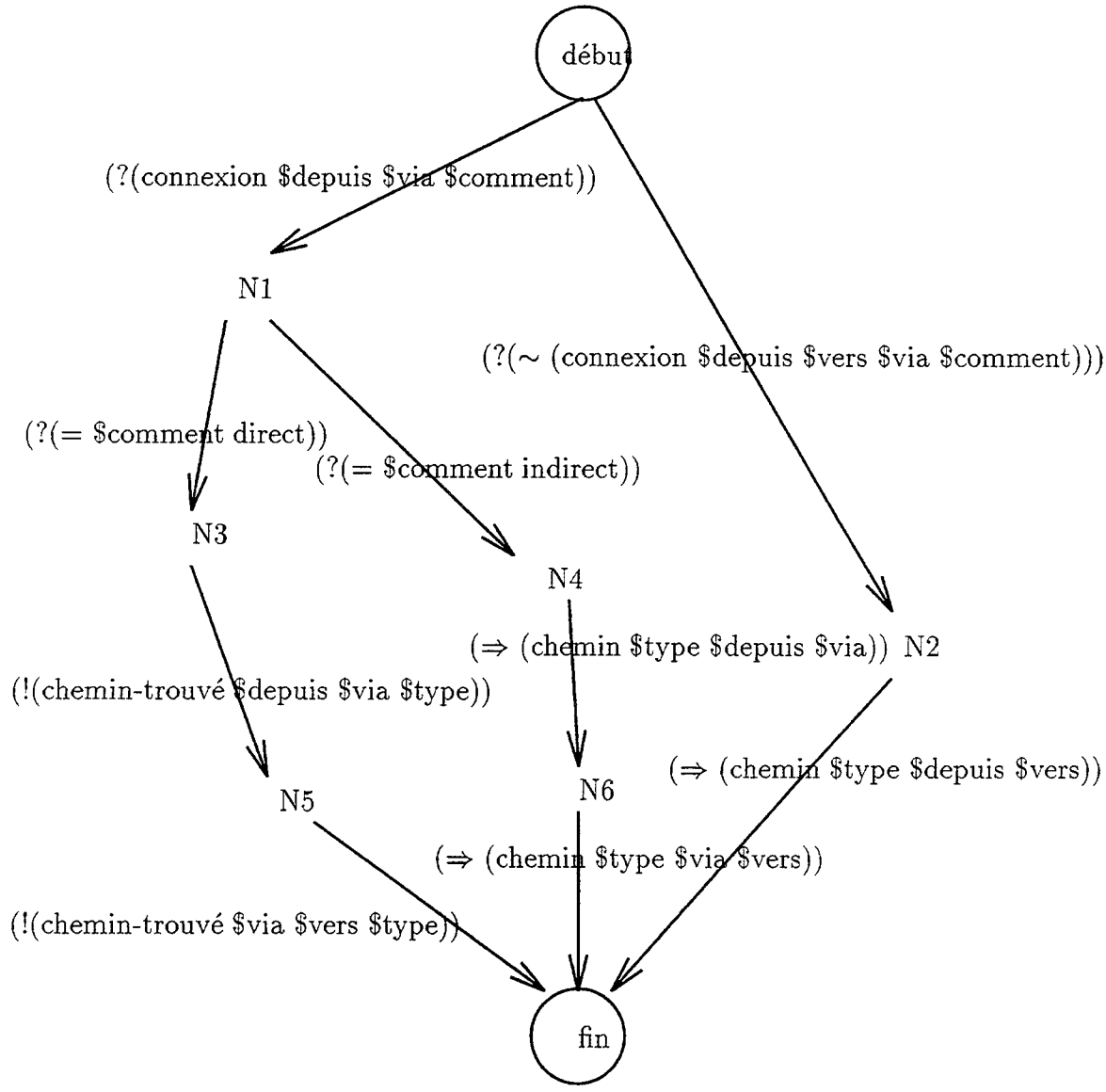


FIG. 2.2 - *Exemple de plan partiel dans le système PRS*

PRS permet également un méta-niveau de connaissances :

- $\Rightarrow p$ signifie que l'on doit ajouter p à la base de connaissances ;
- BUT p est vrai si p est un but.

Cette technique de planification partielle consiste à réaliser les actions dès qu'elles sont choisies. En observant ce qu'il se produit dans le monde réel, on peut alors planifier depuis la situation courante. Cette approche a l'avantage d'autoriser une prise en compte rapide des informations nouvelles car le cycle de planification est court. Elle est particulièrement adaptée lorsque le système doit choisir parmi un large éventail d'actions dont l'espace des paramètres de chacune d'elles est réduit. L'inconvénient de cette approche est que lorsqu'une mauvaise action est choisie, il est impossible de revenir en arrière. Il est donc important de bien contrôler les choix. C'est l'objectif de l'utilisation d'une représentation procédurale. Un autre inconvénient souligné par Schoppers [Schoppers, 1987] est que, dans PRS, la décomposition des comportements en séquence de sous-buts doit être faite à la main et n'est pas réalisée par un planificateur.

Le système RAP

J. Firby [Firby, 1987] propose un autre modèle d'exécution dans lequel les schémas d'actions peuvent être produits par un planificateur. Il a développé un langage de programmation et un interpréteur pour ce langage fondé sur un modèle d'exécution de plan dans lequel les plans sont dynamiquement interprétés selon la situation en cours. Un planificateur de haut niveau soumet le plan au module d'exécution sous la forme d'un ensemble de tâches partiellement ordonnées. Le module d'exécution décompose chacune de ces tâches grâce à une bibliothèque de méthodes appelées *reactive action packages* ou RAP. Le choix de la méthode dépend de la situation courante telle qu'elle est décrite par les faits contenus dans la mémoire de travail du module d'exécution. Les actions primitives sont exécutées en respectant les contraintes imposées sur la tâche et les contraintes additionnelles survenant lors de la décomposition de la tâche. Certaines actions agissent sur les capteurs pour mettre à jour la mémoire de travail du module d'exécution de manière à ce que le système puisse déterminer si les actions ont été correctement exécutées. Le module d'exécution se compose de :

- une base de données qui, à partir des données capteur, met à jour les connaissances sur l'environnement ;
- un interpréteur ;
- un agenda de tâches qui permet de contrôler les tâches spécifiées par le planificateur et les tâches additionnelles introduites lors de l'application de la RAP.

3. Exemple : !(aller A B) inter #(tenir tournevis) : ce but signifie qu'il faut aller de A vers B tout en tenant le tournevis.

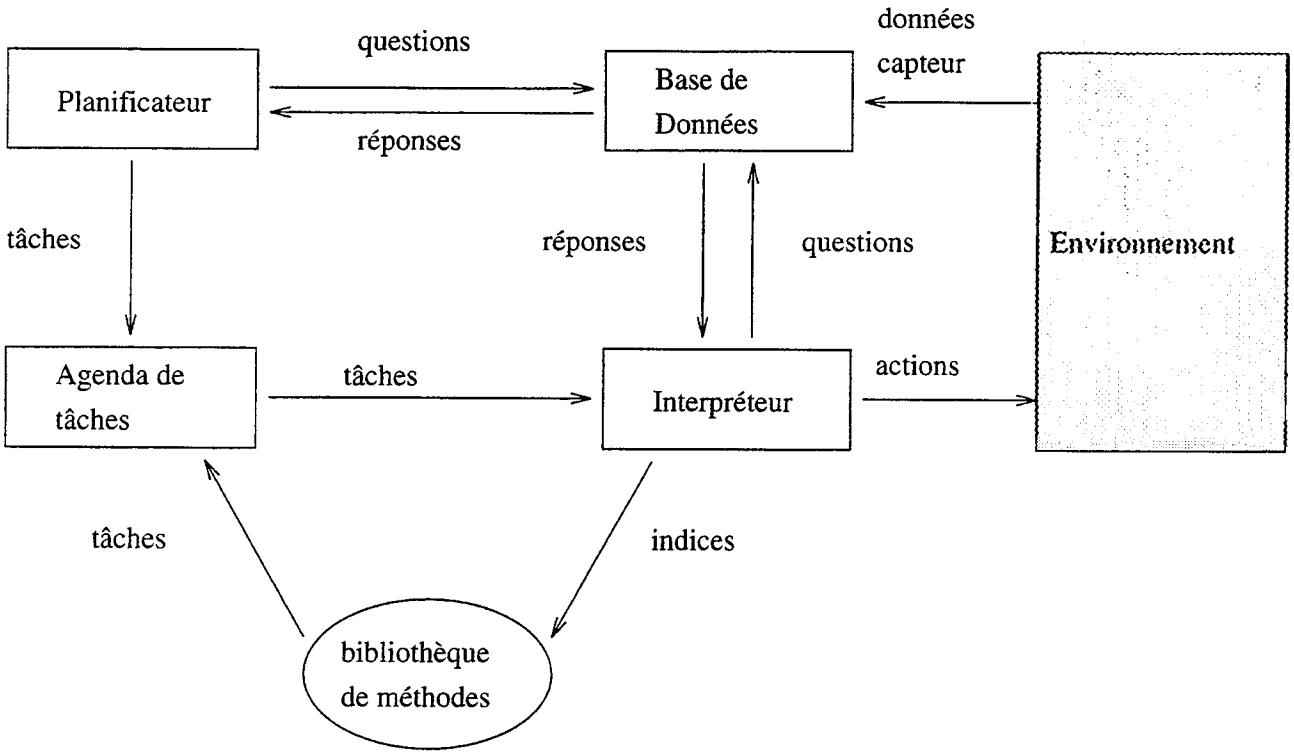


FIG. 2.3 - *Le module d'exécution de RAP.*

Une RAP est décrite par un index qui indique le type de tâche à laquelle elle est associée, un critère de succès qui permet au module d'exécution de déterminer si la tâche est réalisée, un ensemble de méthodes, chacune d'elles spécifiant un contexte dans lequel elle peut être choisie. Un réseau de sous-tâches est associé à chaque méthode. Chaque sous-tâche décrit ses buts et la manière dont les autres sous-tâches en dépendent. Ces dépendances sont exprimées comme des contraintes d'ordonnancement.

Le cycle de base de l'interpréteur peut être décrit comme suit :

1. choisir une tâche dans l'agenda;
2. vérifier dans la base de données que la tâche est à réaliser ;
3. si c'est le cas, choisir la méthode appropriée à la situation courante parmi toutes celles associées à la tâche ;
4. si la méthode est primitive, l'exécuter ;
5. en cas d'échec , marquer la tâche comme ayant échoué;
6. si elle réussit, remettre la tâche dans l'agenda ;
7. si la méthode est décrite par un réseau de sous-tâches, celles-ci sont insérées dans l'agenda, y compris la tâche principale.

Ainsi, tant qu'une tâche n'a pas complètement échoué, elle est remise dans l'agenda. Les tâches sont éliminées si leur critère de succès est vérifié. Si une tâche primitive échoue, alors la tâche qui l'a engendrée est encore dans l'agenda et le module d'exécution va éventuellement tenter de la redécomposer dans un contexte d'exécution nouveau. Ainsi la méthode sélectionnée pour réaliser la tâche pourra être différente. L'interpréteur est suffisamment simple pour qu'un cycle d'exécution nécessite peu de calculs. Cette simplicité assure un temps de réponse court et donc une bonne réactivité du système.

Cependant, les problèmes de coordination ou d'interactions possibles entre les tâches ne sont pas traités. Ceci peut entraîner un bouclage sur une action qui échoue continuellement. Cette situation peut être évitée si le système décide d'essayer une autre action après un certain nombre d'échecs ou si l'échec est remonté au système de planification de haut niveau. Le module d'exécution peut échouer dans la réalisation d'une tâche et cependant rester réactif ; les problèmes nécessitant plus de capacité de raisonnement pour les résoudre sont remontés au planificateur.

2.3 Synthèse

La complexité des méthodes gén'erales de planification nous impose de chercher une solution adaptée au type d'application envisagée.

L'approche générale traite le problème de planification dans le cas où il peut y avoir beaucoup d'opérateurs différents. Pour ce qui concerne la navigation, nous n'en considérons que trois : *aller, rester,* et *réinitialiser.* Ces opérateurs ont des préconditions et des post-conditions triviales mais sont paramétrés par des coûts. Le critère de choix des opérateurs par le planificateur est basé sur l'évaluation de ces coûts et des contraintes à satisfaire.

D'autre part, l'interaction du robot avec son environnement, dans le cas d'un robot de surveillance, est limitée au fait que l'environnement évolue indépendamment des

actions du robot. En effet ce dernier ne modifie pas intentionnellement son environnement. Dans la situation contraire, deux alternatives sont possibles. Soit il est possible de faire l'hypothèse que ces modifications ne peuvent pas remettre en cause l'exécution du plan dans le futur et la faisabilité des actions dépend uniquement de l'évolution propre de l'environnement. Dans ce cas le problème de ramification ne se pose pas et le problème de persistance est géré à l'exécution. Soit il faut envisager un planificateur général capable de faire une gestion globale des coûts, dont la recherche d'itinéraire soit un des modules.

Notre objectif est de concevoir un système de planification suffisamment simple pour que la planification soit assez rapide et réalisable pendant l'exécution de la mission. Cela nous amène à poser des hypothèses simplificatrices :

- L'ordonnancement des tâches de navigation est donnée par l'opérateur via la spécification de la mission.

- Pour simplifier cette planification nous avons pris l'option de séparer la planification des tâches de navigation de celle des tâches opérationnelles.

Ces tâches sont planifiées dans l'ordre dans lequel elles sont spécifiées. La suite ordonnée des tâches de navigation n'est pas remise en cause. Les buts des tâches de navigation devront donc être atteints dans l'ordre où ils ont été spécifiés.

L'activité de surveillance fait appel à l'exécution de procédures prédéfinies qui ne nécessitent pas de planification spécifique. La planification des tâches de navigation fournit des contraintes d'ordonnancement pour ces tâches opérationnelles. Il faut donc que les tâches de navigation soient correctement spécifiées pour que les tâches opérationnelles puissent s'exécuter en parallèle.

Cette conception est réaliste pour la spécification de missions d'une flottille de robot. A partir d'un ensemble non ordonné d'objectifs, un système central peut attribuer des objectifs à chaque robot en utilisant des techniques de planification plus générales mais plus coûteuses en temps de calcul. Chaque robot dispose alors d'une mission définie par des objectifs ordonnés. Le superviseur embarqué sur chaque robot a des capacités de raisonnement suffisantes pour modifier son propre plan en cours d'exécution afin d'atteindre ses objectifs, les problèmes locaux d'exécution pouvant dans la plupart des cas être résolus localement.

Chapitre 3

Architectures de contrôle pour un robot mobile

Les systèmes de contrôle intelligent doivent pouvoir accepter des commandes assez générales de la part d'un opérateur humain. Ces systèmes doivent être capables d'interpréter ces commandes et d'accomplir des actions qui les réalisent. Du fait des contraintes d'embarquabilité (limite en mémoire, en puissance de calcul) la nomenclature de ces commandes est elle-même limitée et l'ensemble des actions élémentaires que le système peut accomplir est a priori connu par l'opérateur. Comme nous l'avons montré dans le chapitre précédent, le mécanisme de choix des actions possibles n'est pas unique et il n'est pas simple de déterminer les lois de sélection de ces actions. La caractéristique principale de ces systèmes est qu'ils doivent interagir avec un environnement réel : incertain et dynamique.

Cela suppose une organisation adaptée des transferts d'information entre les différentes composantes pour construire ces systèmes qui deviennent de plus en plus complexes. Le principe d'organisation est généralement appelé **architecture de contrôle**.

L'imprévisibilité de l'environnement introduit des contraintes temps réel pour la réalisation logicielle et matérielle de tels systèmes. C'est la raison pour laquelle l'aspect **temps de réponse** apparaît clairement dans la conception de beaucoup de ces architectures.

De nombreuses architectures de contrôle ont été proposées pour construire ces systèmes. La décomposition planification - contrôle d'exécution qui a présidé au début des recherches en robotique partait de l'hypothèse que l'on ne commence l'exécution que lorsque tout le plan d'action a été calculé et que cette planification détermine à l'avance toutes les actions, même les plus élémentaires. Depuis il est apparu que cette conception ne pouvait être mise en pratique que par une décomposition plus précise en sous-problèmes de planification et sous-problèmes de contrôle.

Pour décrire ces diverses architectures plusieurs classifications ont été proposées [Crowley, 1987; Hörmann, 1991], il en ressort que toute architecture de contrôle se structure en hiérarchie dès qu'il faut traiter une grande quantité de données ou qu'une grande quantité de calculs doit être effectuée. La conception repose sur un concept hiérarchique indépendamment du fait que le contrôle soit centralisé, comme c'est le cas

pour l'architecture TCA [Hebert *et al.*, 1989; Simmons, 1990], ou distribué [Elfes et Talukdar, 1983; Kœnig et Crochon, 1988].

3.1 Principes de hiérarchisation

Nous allons présenter dans cette section les concepts qui ont guidé la réalisation de la plupart des systèmes de contrôle intelligents actuels. Cet ensemble de systèmes suit globalement la même approche regroupée sous le terme : systèmes hiérarchiques à niveaux multiples.

3.1.1 Décomposition hiérarchique en sous-systèmes

Dans la théorie des systèmes hiérarchiques exposée par Mesarovic [Mesarovic *et al.*, 1970] un système global peut être considéré comme une famille de sous-systèmes en interaction, selon un arrangement vertical. Chaque sous-système possède des canaux d'entrées / sorties pour les échanges avec les unités de niveau supérieur, les unités de niveau inférieur et l'environnement. Remarquons qu'en toute généralité, n'importe quelle unité peut être en relation directe avec l'environnement et agir sur lui à partir de ses propres données d'entrée.

Dans ces systèmes, la priorité est accordée à l'action ou au droit d'intervention des sous-systèmes de niveau le plus élevé. Le contrôle exercé par une unité sur celles de niveau inférieur n'est plus seulement l'expression d'une commande mais il se généralise à la possibilité de redéfinir la loi de commande, qui traduit les objectifs du système en un ensemble de commandes, ou même de redéfinir les objectifs du système. Le succès d'une unité supérieure dépend de la performance des unités inférieures. La performance est alors considérée comme une rétroaction.

Cette interdépendance est encore plus forte dans les systèmes où les échanges avec l'environnement se font exclusivement sur les niveaux inférieurs du système, comme c'est presque toujours le cas en robotique. Les caractères communs liés à ce type de structuration sont les suivants :

1. une unité de niveau supérieur est concernée par une partie plus large ou des aspects plus généraux du comportement du système global ;

2. la période de décision d'une unité de niveau supérieur est plus longue que celle d'une unité inférieure. En effet, pour évaluer les effets de la coordination, l'unité supérieure ne peut agir plus souvent que les unités dont le comportement est conditionné par la coordination.

3. une unité de niveau supérieur est concernée par les aspects plus lents du comportement du système global.

3.1.2 Les différents types de coordination

L'unité supérieure a la priorité de l'action : elle donne les instructions aux unités inférieures sur la façon de procéder et elle les influence pour modifier leur fonctionnement. La notion de priorité de l'action correspond au problème du choix de la structure des relations entre unités supérieures et inférieures : c'est le choix d'un *mode de coordination.* La notion d'influence correspond au choix de l'intervention réelle, ou de la variable de coordination. Les différents degrés d'influence que peut exercer une unité sur une unité immédiatement inférieure vont de la coordination par prévision à la coordination par coalition, dans un ordre de complexité croissante. Dans le cas de la coordination par prévision, l'unité supérieure spécifie complètement le comportement des unités inférieures qu'elle contrôle. Le type de coordination le plus complexe est la coordination de type coalition : les unités inférieures admettent l'existence des autres unités de décision de même niveau ; l'unité supérieure spécifie quels types de communication sont autorisés entre elles : coalition ou compétition. On s'approche ainsi des concepts étudiés dans les approches multi-agents [Boissier, 1993].

3.1.3 Justification des structures hiérarchisées

Un des inconvénients évident d'un système à niveaux multiples est la complexité de son fonctionnement. Le système ne peut pas être commandé aussi facilement qu'on le voudrait. Néanmoins certaines justifications pratiques et théoriques [Mesarovic *et al.*, 1970] ont été proposées en faveur de cette approche. Elles concernent tout d'abord les performances de tels systèmes [Maximov et Meystel, 1992]. Les ressources sont mieux utilisées lorsque l'on adopte une approche à niveaux multiples pour résoudre les problèmes de grande dimension. Cela dépend néanmoins fortement du mode de coordination.

La détermination de la structure hiérarchique optimale pour un système donné est réalisable lorsque les traitements peuvent être exprimés de manière homogène sur tous les niveaux de la hiérarchie. Mais ce choix ne dépend pas uniquement de considérations sur la complexité d'un contrôle hiérarchique mais aussi de considérations sur la méthode de conception, notamment sur la facilité d'intégration. Un système hiérarchique se développe souvent à partir de composants déjà existants. Pour construire un système de commande intégré, il est fréquent de prendre comme point de départ un système donné qui contient le processus et le niveau de commande inférieur. On ajoute ensuite un niveau de commande supérieur de manière à intégrer le fonctionnement total du système pour coordonner les interactions des sous-systèmes. Une autre justification repose sur un critère de flexibilité : dans un système décentralisé à niveaux multiples, les changements dans la procédure de décision nécessités par la modification du fonctionnement d'un sous-processus peuvent être localisés, et en conséquence, réalisés au moindre coût et dans un délai bref, le système s'adapte plus vite.

3.1.4 Exemples de hiérarchies

Ce qui permet de distinguer les différentes architectures étudiées c'est la manière dont elles combinent certains des principes de hiérarchisation énoncés ci-dessous, identifiés par Schoppers [Schoppers, 1992].

Hiérarchie fréquentielle [Albus *et al.*, 1987; Payton, 1986; Crowley, 1987] Un niveau est défini par la fréquence à laquelle sont effectués les traitements. Les traitements sont effectués d'autant plus fréquemment que le niveau de traitement est plus bas.

Hiérarchie d'abstraction de données ou fonctionnelle [Chochon, 1991; Camargo, 1991] Ce principe de hiérarchisation est celui prôné par la méthodologie orientée objet. Le concept de base étant de fournir un ensemble de fonctionnalités encapsulées dans un module et un protocole client serveur entre ces modules de tel manière qu'un module n'a pas à connaître le fonctionnement des autres pour interagir avec eux.

Hiérarchies d'abstraction des représentations [Sacerdoti, 1973] Le principe est de supprimer ou d'ignorer de l'information en remontant dans la hiérarchie de représentation de l'information. Cette simplification des représentations est réalisée en fonction du problème à résoudre.

Hiérarchies de résolution [Crowley, 1981; Goto et Stentz, 1987] A la différence d'une hiérarchie d'abstraction où un niveau peut ignorer des informations du niveau inférieur, dans une hiérarchie de résolution, les informations d'un niveau sont obtenues par un opérateur qui effectue un traitement global (moyenne, Laplacien), sur les informations du niveau inférieur. Les informations étant alors en quantité moindre, les traitements peuvent s'effectuer plus rapidement. On parle alors de *pyramide de résolution.*

Hiérarchies d'agrégation [Erman, 1980; Crowley et Christensen, 1994] Ce type de hiérarchie concerne les systèmes de perception. Ici, chaque niveau est défini par une méthode de groupement des informations perceptuelles du niveau inférieur. Ceci permet de remonter du niveau signal, le plus bas, au niveau symbolique, le plus haut. Au contraire de la hiérarchisation par abstraction des représentations, il n'y a pas dans le cas présent d'élimination d'une partie de l'information.

Hiérarchies de comportements [Brooks, 1985; Kaebling, 1986] Les niveaux sont organisés en "comportements" de plus en plus complexes. Traditionnellement on appelle comportement d'un système l'ensemble des réponses observables qu'il produit en réaction aux stimuli extérieurs. Chacun des niveaux de compétence étant capable de fonctionner individuellement, par abus de langage, on appelle comportement chaque niveau de cette architecture. Chaque niveau donne une compétence propre au système, par exemple errer ou explorer. Cette décomposition hiérarchique est orthogonale aux hiérarchies de niveaux d'abstraction ou de résolution.

3.2 Exemples d'architectures de contrôle

Les architectures présentées ici caractérisent les différences radicales qui peuvent exister dans la conception des architectures de contrôle les plus récentes. Le premier exemple est celui d'un système basé sur une hiérarchisation fonctionnelle flexible, le deuxième sur une hiérarchie frequentielle et le troisième sur une hiérarchie de comportements.

3.2.1 Hiérarchie fonctionnelle flexible

Le robot HILARE II, en service au CNRS-LAAS à Toulouse depuis fin 1990, possède une structure de contrôle à deux composantes [Camargo, 1991] (cf figure 3.1) :

1. un ensemble de modules fonctionnels,

2. un contrôleur central, implanté au dessus de cet ensemble, chargé de la supervision globale de l'exécution du plan d'actions.

Les modules fonctionnels : Les modules coopèrent entre eux pour réaliser les tâches fournies par le contrôleur central. Cet ensemble de modules représente la composante distribuée du système de contrôle, laquelle permet au robot d'accomplir une mission avec un degré suffisant d'autonomie et de réactivité. Cet ensemble de modules fonctionnels est extensible.

Les modules sont des entités logicielles capables de réaliser un ensemble de services spécifiques appelés "fonctionnalités", au moyen d'actions sur le robot virtuel (actionneurs et capteurs) et/ou sur les autres modules du système. Par exemple, le module locomotion comprend les fonctionnalités :

- calculer la position odométrique,

- lire la position odométrique,

- initialiser la position odométrique,

- asservir le robot en position,

- recevoir une position de consigne de position.

Le regroupement des fonctionnalités en modules est dicté par des raisons d'efficacité : elles partagent les mêmes ressources matérielles et/ou logicielles. Un module est conçu pour pouvoir être utilisé par n'importe quel autre module du système. Il n'y a pas de concept de couches hiérarchiques pré-définies. Les modules communiquent entre eux par des mécanismes de client / serveur, sans connaissance préalable des interlocuteurs possibles, de manière à garantir l'évolutivité du système.

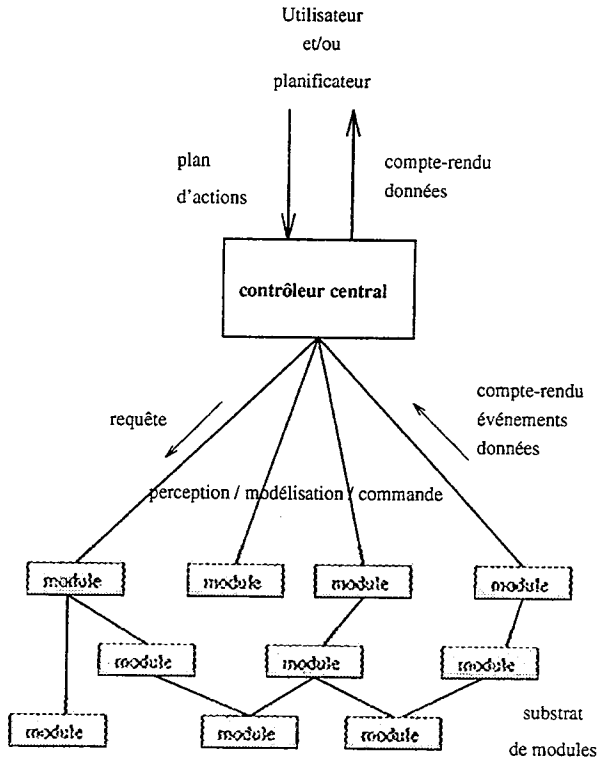


FIG. 3.1 - *Architecture de contrôle du robot Hilare II*

Le contrôleur : Dans cette approche, le contrôleur central possède des connaissances sur la finalité des actions à accomplir, il est donc capable de :

1. faire exécuter ces tâches par les modules, au moyen de l'envoi de requêtes de service, tout en assurant la gestion des attitudes à prendre face à des événements asynchrones,

2. superviser le bon déroulement du plan d'actions et, dans le cas d'échec détecté pendant l'exécution d'une action, il doit soit solliciter l'aide de l'opérateur et/ou du planificateur, soit essayer de résoudre de façon interne le problème, au moyen de l'accès à d'éventuels sous-systèmes de diagnostic , reprise, etc.

L'exécution d'actions est représentée par des *activités*. Une activité simple est l'exécution d'une fonction par un module. Cette activité peut déclencher des requêtes vers d'autres modules déclenchant des *activités filles*. A un instant donné, l'ensemble des activités représente les fonctions en cours d'exécution. Les activités du système se structurent en un arbre qui évolue dynamiquement selon les tâches que le système doit accomplir.

3.2.2 Architecture basée sur la coordination hiérarchique Action-Perception

J. L. Crowley a proposé en 1987 une architecture de contrôle pour un robot mobile basée sur la coordination hiérarchique action - perception. Notre superviseur s'intégrant dans cette architecture, nous allons la décrire de façon plus détaillée que la précédente.

Cette hiérarchie coordonnant action et perception est à la fois fréquentielle et fonctionnelle (cf fig 3.2). La hiérarchie des processus de perception combine des principes d'agrégation et d'abstraction :

Le niveau sensori-moteur : au plus bas niveau les contrôleurs individuels des moteurs fonctionnent de manière asynchrone ainsi que les processus qui acquièrent les signaux des capteurs toutes les 4 milisecondes.

Le niveau commande du véhicule et intégration sensorielle : à ce niveau les multiples informations perceptuelles sont intégrées dans un modèle géométrique exprimé dans un système de coordonnées unique. La commande du véhicule assure l'exécution des trajectoires de celui-ci. Les processus opèrent à ce niveau toutes les 80 milisecondes.

Le niveau actions de perception et de navigation : à ce troisième niveau sont définies, de façon algorithmique, les capacités d'action et de perception du système. Le cycle est environ de 500 millisecondes pour une planification locale de chemin.

Le niveau symbolique : le superviseur se trouve au plus haut niveau de la hiérarchie. Là est définie la connaissance symbolique de l'environnement, sont définis les buts, les moyens de les planifier et le contrôle de l'exécution des plans. Le superviseur assure également la communication avec l'opérateur.

Nous ne décrirons dans les sections suivantes que les principes généraux mis en œuvre. Une description plus est donnée dans [Crowley et Coutaz, 1986; Crowley, 1989; Crowley *et al.*, 1991].

Commande du véhicule et intégration sensorielle

Le contrôleur de véhicule Le contrôleur de véhicule est organisé en trois couches. La couche de plus haut niveau est l'interface entre le contrôleur et les autres composantes du système. Elle est constituée de procédures gérant les déplacements du robot et l' état cinématique du véhicule. La deuxième couche est chargée de maintenir les données cinématiques du robot (position, vitesse) et de l'asservir par rapport aux consignes de déplacement. On peut l'asservir séparément pour les translations et les rotations et ainsi combiner les deux types de mouvements. Pour chacun de ces deux types de mouvements le profil temps - vitesse est de type trapezoïdal (figure 3.3).

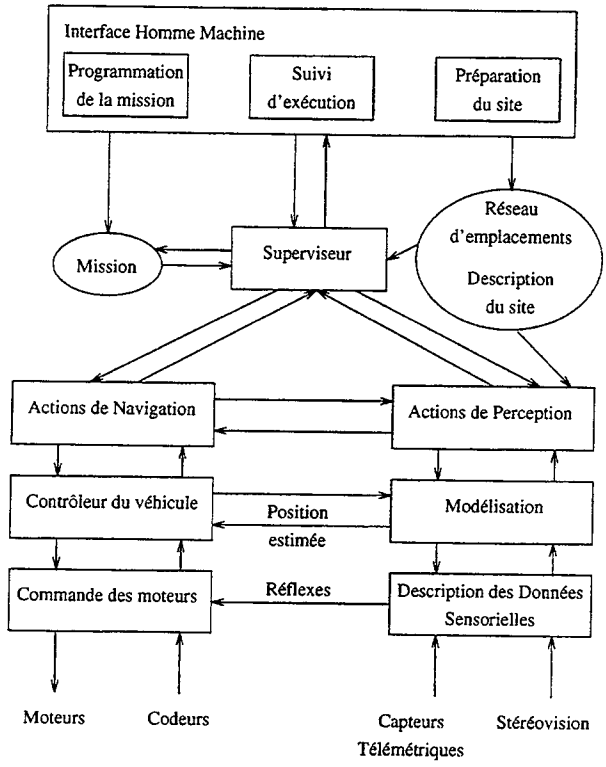


FIG. 3.2 - *Architecture Logicielle et matérielle du robot mobile MITHRA.*

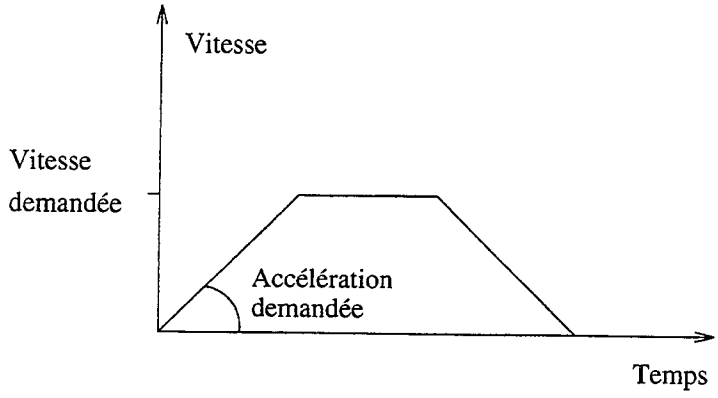


FIG. 3.3 - *Diagramme des vitesses lors d'un déplacement.*

Les couches que nous venons de décrire constituent la partie indépendante de la géométrie du véhicule et fournissent donc un contrôleur de véhicule standard. La couche de plus bas niveau comprend les données géométriques propres au robot. Elle transforme des déplacements linéaires et angulaires en déplacement pour chacune des deux roues motrices, et inversement.

Intégration sensorielle Les capteurs utilisés dans le cadre de notre étude sont des télémètres ultrasoniques. Ces capteurs fournissent des profondeurs relativement peu précises (la précision dépend des matériaux constituant les obstacles, ainsi que de la température de l'air). De manière à fiabiliser les résultats, il est nécessaire de rechercher une corrélation entre les données. Pour ce faire, nous transformons toutes les mesures en points dans le repère absolu (grâce à la position estimée du véhicule) et nous recherchons lesquels sont susceptibles d'être alignés pour former un segment (il en faut trois au minimum pour qu'un segment soit créé).

L'intégration sensorielle a pour but de modéliser l'environnement. Le rôle de la modélisation pour un véhicule autonome est double :

- il s'agit tout d'abord de déterminer les régions libres et les régions occupées de l'espace. La connaissance de l'espace libre permet au véhicule de se déplacer sans entrer en collision avec les différents objets constituant l'environnement. Les modules de perception et de navigation devront être conçus à partir des primitives géométriques que la modélisation sera en mesure de fournir. Leurs performances respectives dépendront de deux critères : le temps de réponse du modèle (temps écoulé entre l'apparition d'un objet dans la scène et son apparition dans le modèle) et l'adéquation entre ce modèle et la réalité (est-ce que les primitives géométriques du modèle correspondent à un objet de la scène et inversement).

- le second rôle est de permettre de maintenir la position estimée du robot au cours de ses déplacements. Ceci lui permet de se localiser par rapport aux différents lieux où il doit se rendre. De la précision de cette localisation dépendra la précision avec laquelle le véhicule pourra atteindre un point.

Les données sensorielles brutes sont dans un premier temps traitées de manière à produire une description de l'environnement immédiat en termes de primitives géométriques. Les primitives géométriques sont les segments et les points (ceux n'appartenant pas à un segment). Ces primitives sont ensuite envoyées au fur et à mesure de leur création vers le processus de modélisation proprement dit, dont la tâche est de maintenir le *modèle local composite.*

Ce modèle est dit local car il n'intègre que les données situées dans un environnement proche du véhicule. Il est dit composite car il est formé d'observations réalisées en différents point de l'espace, et pouvant éventuellement provenir de plusieurs sources différentes (dans notre cas, il y a une seule source, les capteurs ultrasons).

Les primitives géométriques provenant des couches basses sont mises en correspondance avec celles déjà présentes dans le modèle. Les nouvelles sont ajoutées, celles déjà existantes sont mises à jour (par utilisation du filtre de Kalman). Cette mise à jour

a pour effet de bord de produire une correction de la position estimée du véhicule (seulement lorsque l'élément observé est un segment).

Le temps de calcul du cycle de modélisation dépend du nombre de segments présents dans le modèle (complexité linéaire dans le cadre de la mise en correspondance). De manière à garder des temps de réponse corrects, toute primitive du modèle n'ayant pas été observée depuis trop longtemps est jugée obsolète et supprimée. Ceci permet aussi de garder la propriété de localité. Enfin, le processus peut avoir recours s'il le décide à une carte pré-apprise de l'environnement contenant les obstacles principaux.

Le véhicule est donc maintenant capable de maintenir sa position estimée et a une connaissance de l'espace libre. Cette connaissance est sous forme "brute" (liste de points et de segments). Il s'agit maintenant de fournir à la navigation des outils permettant de l'exploiter. C'est le rôle du module de perception.

Actions de perception et de navigation

Le module de perception a pour rôle d'extraire pour la navigation les données pertinentes contenues dans le modèle local. Il doit donc être conçu en tenant compte des primitives géométriques élémentaires fournies par la modélisation. Il doit également s'appuyer sur les caractéristiques des trajectoires que doit pouvoir accomplir le robot. Il utilise pour cela :

- La connaissance pré-établie du monde réel extérieur.
- Les informations reçues par les capteurs ultrasoniques.
- Les connaissances acquises depuis le début de la session.

Ce module fournit également un ensemble de primitives d'analyse de cet environnement pour la navigation. Le véhicule est en mesure de percevoir son environnement et de l'interpréter en termes de chemins possibles.

La fonction principale du module de navigation est l'exécution de l'action *Aller à*. Son paramètre est un point de l'espace d'évolution en coordonnées cartésiennes. La planification de trajectoire consiste à décomposer l'action en un ensemble de commandes *move* et *turn* interprétables par le niveau commande du véhicule.

- La fonction *Aller à* dirige le robot le long d'une ligne droite jusqu'à son but, ceci à une vitesse déterminée. Si le chemin n'est pas libre, cette fonction identifie le premier obstacle rencontré, et ne déclenche aucun mouvement.

  Si le but n'est pas face au robot, celui-ci effectue d'abord une rotation afin de s'aligner. Au cours du déplacement, le robot vérifie constamment que le chemin reste libre (dans l'éventualité, par exemple, d'un obstacle mobile) et effectue des corrections d'orientation pour atteindre effectivement son but.

  Remarquons que le point d'arrivée ne se situe pas nécessairement au moment de la commande dans le champ de visibilité du système de perception.

- La fonction *Eviter* détermine un chemin pour contourner un obstacle et lance l'exécution des différents segments le composant. Si le but n'est pas atteignable, une erreur est remontée au superviseur. Le but n'appartient pas nécessairement à l'horizon visible au départ de l'exécution. Si au cours de l'exécution d'un des segments du chemin résultat le robot détecte un nouvel obstacle, il essaie alors de déterminer un nouveau chemin pour le contourner.

Cette décomposition qui traite de manière séparée la trajectoire nominale de la trajectoire d'évitement d'obstacles se retrouve dans la description de l'architecture d'Hilare II [Camargo, 1991].

3.2.3 Approches basées sur les comportements

C'est à partir d'une perspective totalement orthogonale aux approches précédentes que les architectures réactives ont été définies. La puissance de calcul embarquée n'est plus focalisée sur la réalisation d'une tâche particulière jusqu'à son succès ou son échec. Elle est répartie entre plusieurs comportements qui se partagent le contrôle du système.

D'après Brooks [Brooks, 1991], les principes suivants devraient être à la base de la conception de toute architecture de robot mobile :

La mise en situation : Dans la plupart des approches classiques, le robot ne traite pas le monde réel se trouvant autour de lui, mais une représentation abstraite. Un des problèmes résultant directement de cette approche est : quelle représentation utiliser pour modéliser le monde?

Le principe de la *mise en situation* suggère que non seulement il est très difficile de modéliser le monde, mais bien plus, que cette modélisation est inutile. Le meilleur modèle du monde est le monde lui-même. Le problème de la fusion multi-sensorielle disparaît du même coup : les capteurs utilisent le monde comme point de référence.

La nécessité de l'incarnation :

On appelle incarnation le fait que les systèmes intelligents soient dans un corps. Seule l'incarnation d'un agent peut le valider, c'est à dire attester qu'il est capable de prendre en compte et de réagir dans le monde réel. Les relations physiques et sensorielles du robot avec le monde ne sont pas connues a-priori et doivent être acquises par le système lui-même.

Pour cela, il propose un modèle d'architecture qu'il appelle *subsomption* qui se résume dans les principes suivants :

1. les traitements sont distribués dans un réseau asynchrone à topologie fixe dont les éléments sont des machines d'états finis,

2. les messages de communication entre éléments n'ont pas de sémantique figée ; chaque élément interprète les messages suivant le contexte.

3. les capteurs et effecteurs sont connectés directement sur ce réseau.

Les comportements sont combinés en niveaux de compétence correspondant aux capacités du robot. Chaque niveau de compétence couple la perception à l'action.

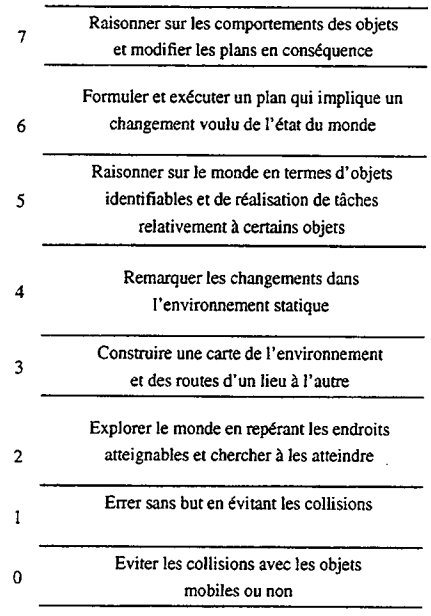


FIG. 3.4 - *Décomposition d'un système de contrôle en comportements*

Les niveaux de compétence

Dans [Brooks, 1985], Brooks décompose le problème du système de contrôle du robot sur la base de ses capacités externes. Il distingue huit niveaux de compétence. Un niveau de compétence représente une classe de comportements désirés pour le robot, dans tous les environnements qu'il rencontrera. La figure 3.4 représente une telle décomposition. Elle n'est pas unique. Chaque nouveau niveau de compétence inclut le niveau immédiatement inférieur.

Les niveaux bas (0, 1 et 2) sont des niveaux à haute priorité; ce sont des actions de type réflexe qui permettent au robot d'être très réactif lorsqu'il est confronté à des situations difficiles. Les tâches mises en jeu dans ces niveaux se doivent surtout d'être rapides, efficaces et robustes.

Les tâches à faible priorité concernent les couches supérieures avec la partie décisionnelle du système de navigation. La reconnaissance de formes, l'identification d'objets, la cartographie, le choix des buts doivent se situer dans ces niveaux. En maîtrisant le système d'alerte, ces tâches confèrent au robot un comportement intelligent, sans pour autant remettre en cause l'efficacité des tâches de haute priorité.

L'idée maîtresse est que chaque niveau de compétence peut être construit au-dessus des niveaux déjà élaborés et testés sans remettre en cause l'ensemble du système. Lorsqu'un niveau a été testé, il n'est plus retouché lors de l'évolution du système. On

commence donc par construire les niveaux bas en testant et validant les actions réflexes. Le niveau n a le droit d'examiner les données circulant au niveau $n-1$. Il peut aussi les supprimer ou les inhiber : le niveau n *subsume* le niveau $n-1$.

Un niveau est constitué d'un ensemble de modules communiquant de manière asynchrone à l'aide de messages. L'architecture -une machine d'état fini étendue (AFSM)- est constituée de registres, d'horloges, d'un réseau et d'une machine d'état fini classique.

Mise en œuvre de l'architecture "subsomption"

Nous donnons ci-dessous un exemple d'architecture testée en simulation puis sur un robot réel. Le schéma (figure 3.5) illustre la conception du système de contrôle couche par couche. La couche inferieure représente le niveau 0 du système de contrôle, c'est-à-dire l'évitement de collisions. La couche intermédiaire crée un comportement d'errance utilisant la couche de niveau 0 et la couche superieure met en œuvre un comportement exploratoire.

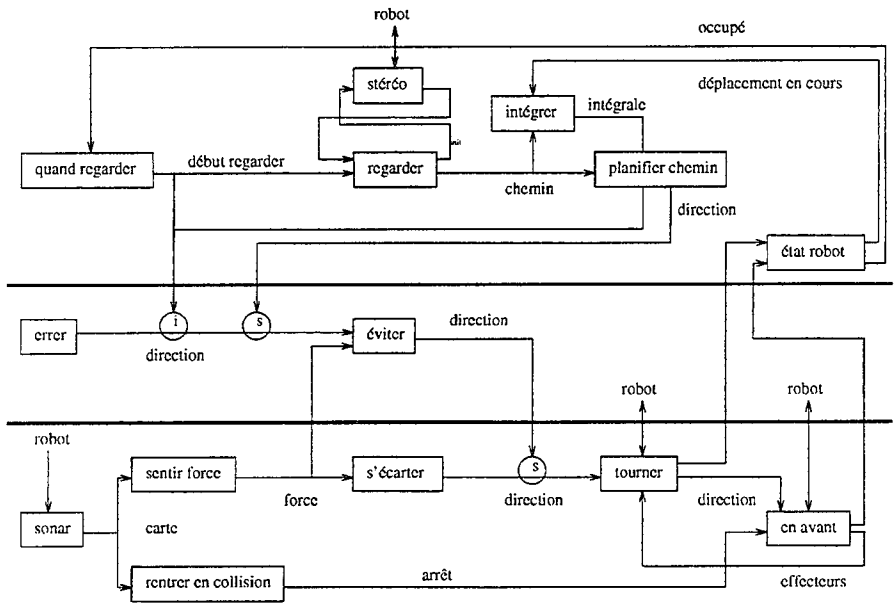


FIG. 3.5 - *Niveaux 0:évitement, 1:errance et 2:exploration*

Expérimentation

Cette architecture a été testée d'abord en simulation puis en grandeur réelle sur divers robots : robot mobile autonome, insecte artificiel (robot à six pattes). Des démonstrations ont permis de montrer expérimentalement une des assertions de départ de Brooks : l'intelligence peut se trouver dans l'œil de l'observateur. En effet ces robots sont constitués de modules extrêmement simples mais eventuellement nombreux. En particulier la fourmi décrite dans [Brooks, 1989] qui apprend à marcher puis à suivre des êtres vivants, ne comporte pas moins de 150 capteurs qui rendent sa conception assez délicate.

Bien qu'en principe modulaire, l'architecture proposée par Brooks est difficile à mettre en œuvre dans le cadre d'un système très complexe : les communications et leur câblage deviennent très difficiles à maîtriser. L'ajout d'un nouveau module, théoriquement aisé, devient rapidement inextricable.

L'ajout d'un nouveau niveau ne touche théoriquement que peu le fonctionnement des niveaux inférieurs. En particulier, il ne doit surtout pas remettre en cause les capacités d'actions réflexes, ni les résultats des tests des niveaux prioritaires. Or l'ajout de fils de suppression et d'inhibition peut perturber complètement le fonctionnement des couches inférieures. Il semble exister donc une certaine incohérence entre les principes à la base de l'architecture et les méthodes effectivement mises en œuvre.

3.2.4 Coordination par concurrence ou compétition

L'un des problèmes majeurs dans les approches comportementales réside dans la manière dont les comportements coopèrent vers un résultat cohérent et constructif ; les deux principales approches sont la concurrence et la compétition.

Concurrence : Lorsque les divers comportements sont activés de manière concurrente, ils ne possèdent en général aucune connaissance les uns des autres et leurs produits sont intégrés sans discernement dans la commande envoyée aux effecteurs.

On retrouve cette solution dans les agents autonomes de Anderson : après sélection des comportements qui doivent être actifs, leurs résultats sont combinés avant d'être envoyés au robot mobile [Anderson et Donath, 1990]. Le problème a légèrement été faussé dans ce cas : Anderson réalise une sélection a priori des comportements qu'il veut utiliser pour émuler un comportement complexe.

Dans les travaux de Beer [Beer *et al.*, 1990], le problème se retrouve et les conflits sont partiellement résolus par des liens d'inhibition permettant d'assurer la compatibilité de comportements. Ces liens sont appris sous forme de réseaux de neurones.

Compétition : C'est l'autre approche, préconisant une compétition et l'élection du meilleur comportement à l'instant d'exécution, qui est utilisée dans les travaux de Pattie Maes sur les modules de compétence [Maes, 1989]. La compétition est régie par une énergie d'activation provenant à la fois des buts (comportement motivé) et des événements (comportement opportuniste).

On peut dire que Brooks et Anderson présentent des approches mixtes, dans la mesure où l'un comme l'autre propose des mécanismes de combinaison et d'inhibition.

Il est assez intéressant de noter que, quelle que soit l'approche, il est impossible de se fier uniquement à une exécution strictement compétitive ou concurrente. La plupart des auteurs utilisant cette voie se trouvent obligés d'ajouter des mécanismes d'inhibition pour éviter les conflits entre les actions générées par les comportements. Ces conflits peuvent être physiques (accès aux ressources matérielles du robot), mais ils peuvent aussi être induits par des états d'attraction trop distincts.

3.2.5 Comportement motivé

Comment, à partir d'un robot basé sur des comportements imités du monde animal, obtenir un robot ayant un comportement motivé, c'est-à-dire exécutant une ou des tâches pré-définies?

Il faut bien reconnaître qu'aujourd'hui la quasi totalité des robots comportementaux n'ont jamais été développés jusqu'à des niveaux de raisonnement suffisants pour pouvoir réellement juger du problème. En effet, si l'architecture comportementale se prête bien à la fabrication de robots très réactifs guidés sur les événements, il est moins évident de leur faire prendre en compte un comportement de plus haut niveau du type "aller-à-tel-point".

Un certain nombre d'auteurs proposent néanmoins des solutions. Il s'agit, pour la plupart du temps, d'approches où des aspects hiérarchiques ont été surajoutés à l'architecture comportementale ou au contraire où les couches basses d'une architecture hiérarchique ont été "comportementalisées" [Payton, 1986; Arkin, 1990; Connell, 1992]. Il est clair qu'en suivant une approche réellement comportementaliste, le meilleur moyen pour qu'un robot effectue une tâche serait d'introduire une motivation liée à cette tâche. Reste à déterminer l'expression de cette motivation.

3.3 Vers des architectures hybrides

Dans les systèmes purement réactifs, la possibilité de pouvoir spécifier un but en tant que position précise à atteindre dans l'espace et dans le temps n'est pas réalisée. Concernant les niveaux hauts d'une architecture de contrôle, il ne semble pas aisé de raisonner avec des automates d'états finis.

Les systèmes purement réactifs n'ont pas un comportement prévisible et n'ayant pas de but précis ils n'ont pas de notion d'échec. Il faut donc définir précisément quelle interface peut exister entre un niveau de raisonnement symbolique et un niveau réactif. C'est ce que Connell [Connell, 1992] se propose de faire dans l'architecture SSS. Le niveau symbolique influe sur le niveau réactif par la paramétrisation des processus réactifs qui sont eux-même capables de retourner des événements vers le niveau symbolique. L'expérience montre que le robot est capable de construire une carte relativement précise de son environnement et ensuite de naviguer rapidement à l'intérieur de celui-ci.

Pour le problème spécifique de la navigation, Reignier pose dans [Reignier, 1993] les principes d'une navigation réactive s'intégrant dans l'architecture hiérarchique, après une analyse des faiblesses de l'approche purement géométrique de la génération de trajectoires. La justification essentielle d'une approche réactive est qu'il faut une adaptation rapide et permanente de la trajectoire à l'environnement et non pas que la détection d'un obstacle particulier déclenche le calcul d'une trajectoire d'évitement. L'architecture inititale a donc été adaptée pour permettre une navigation réactive en ajoutant un lien direct du module de description des données sensorielles vers le module qui gère les actions de navigation.

Chapitre 4

Interface Homme-Machine et représentation des connaissances

Comme nous l'avons vu dans le chapitre précédent, au sommet de l'architecture de contrôle deux modules dialoguent l'un avec l'autre: le superviseur et l'interface homme-machine.

Une interface homme-machine (IHM) est un dispositif logiciel et matériel qui permet la connexion entre la manifestation externe du système et les organes sensori-moteurs de l'utilisateur. L'interface doit effectuer une traduction entre les formalismes du système et ceux de l'utilisateur. Cette interface permet à un opérateur humain de décrire l'environnement d'évolution du robot, les missions que le robot aura à accomplir et elle permet de suivre l'exécution de ces missions. L'objet de ce chapitre est d'aborder les problèmes de représentation des connaissances du point de vue de l'opérateur en tentant de déterminer quelle information utile échanger entre le système et l'opérateur, sans préjuger de la forme ultérieure de cette information, adaptée à son exploitation.

Comme le montre Coutaz [Coutaz, 1987], une IHM bien conçue n'est pas seulement un habillage des fonctions logicielles du système, de même que le système ne doit pas être conçu indépendamment des facultés de l'utilisateur final. Il faut alors définir les propriétés nécessaires pour une bonne interaction homme-machine, où le système et l'opérateur coopèrent à la réalisation d'une tâche. Ici, le terme de tâche est très général et ne désigne pas uniquement la tâche robotique mais toutes les activités de l'utilisateur quand il interagit avec le système. L'interface que nous avons conçue est spécifique aux besoins de communication homme-robot. Sa conception est liée aux caractéristiques de l'architecture logicielle du robot, aux types de missions à effectuer et aux interactions souhaitées.

La prise en compte de l'opérateur dans la boucle de contrôle du plus haut niveau du système fournit une base de spécification pour la conception du superviseur. Ce dernier doit être conçu en tenant compte des propriétés requises pour cette interface.

4.1 Fonctions de l'interface

Notre interface est conçue pour deux classes d'utilisateurs. La première est responsable de la description de l'environnement, la deuxième, typiquement un surveillant, a un rôle de spécification et de suivi des missions. L'opérateur est a priori novice en informatique. Nous avons donc défini un langage de spécification de mission qui contient, dans un formalisme très simple, tous les concepts jugés nécessaires à la définition d'une mission de surveillance.

Pendant la phase de spécification de mission, l'opérateur est assisté pour corriger la mission et il peut en faire une simulation. Pendant son exécution, le superviseur informe l'IHM de l'état d'avancement de la mission. Il s'assure que les contraintes sont respectées et dans le cas contraire il peut décider de replanifier la mission. L'opérateur n'a pas nécessairement besoin d'intervenir mais l'interface lui permet d'arrêter l'exécution à tout moment pour ajouter ou enlever une tâche ou télé-opérer le robot. Une autre fonctionnalité de l'interface est de permettre de faire exécuter une simulation à partir des informations d'exécution d'une mission effectuée par le robot.

4.2 La spécification de l'environnement

Certains auteurs choisissent d'obtenir une description de l'environnement par spécification [de Saint Vincent, 1990] ou par apprentissage [Chatila et Laumond, 1985; Crowley, 1986; Kuipers et Byun, 1988]. Ces descriptions ne sont pas nécessairement équivalentes, la première étant plus guidée par le type de tâche à accomplir, elle peut être plus sommaire que les descriptions obtenues par apprentissage, le système ne sachant distinguer ce qui est utile à la réalisation de la tâche de ce qui ne l'est pas.

Pour s'affranchir des problèmes complexes d'apprentissage autonome de l'environnement nous préférons initialiser un modèle de celui-ci lors d'une phase de spécification quitte à l'affiner petit à petit, à partir des données perçues en exécution. Ce modèle initial donne une description nécessairement imparfaite de l'environnement réel. Il faut vérifier par l'expérimentation qu'elle est suffisante pour permettre une exécution correcte. D'autre part, le système doit, non seulement connaître les caractéristiques géométriques de l'environnement, mais également savoir comment se comporter dans celui-ci. Le système doit pouvoir fournir à l'opérateur une prédiction des caractéristiques d'exécution (durée d'exécution, énergie consommée, etc.). L'opérateur doit être capable d'imposer des comportements spécifiques selon la situation du robot.

4.2.1 Hiérarchies de représentation

Afin d'obtenir une représentation spatiale cohérente, Kuipers propose une description de l'environnement par une hiérarchie de quatre niveaux sémantiques, laquelle lui a permis de développer les systèmes Tour, NX et QUANLAV [Kuipers et Byun, 1988]. Voici succinctement pour chaque niveau quelles sont les informations traitées :

1. Sensori-moteur : les relations d'entrée/sortie qu'a le robot avec l'environnement.

2. Procédural ou tactique : des procédures apprises et stockées définies en termes de primitives sensori-motrices afin d'accomplir des instances particulières des actions de navigation.

3. Topologique : une description de l'environnement en termes d'entités fixes comme les emplacements, chemins, bornes, points de repère, amers ou régions, liées par des relations topologiques telles que la connectivité, la contenance et l'ordre.

4. Métrique : une description de l'environnement avec les mêmes entités que le niveau topologique, mais ces entités sont maintenant liées par des relations métriques telles que la distance et les angles relatifs ou absolus par rapport à un référentiel fixe.

D'autres travaux proposent une description similaire [Chatila et Laumond, 1985; Crowley et Coutaz, 1986]. Les connaissances sont classées en trois catégories : géométrique, cartographique (ou topologique) et tactique. Le type de données nécessaires à la description géométrique repose essentiellement sur le type de capteurs utilisés pour la navigation et la relocalisation alors que le type de données nécessaires à la description cartographique et tactique repose plus sur la nature des tâches à réaliser.

La description topologique de cet environnement peut reposer soit sur la définition de l'espace libre soit sur la définition d'emplacements correspondant à des points de changement de direction possibles permettant d'atteindre de nouvelles régions de l'espace. Ces emplacements et routes sont organisés en réseaux. Les deux descriptions sont presque duales. L'avantage de la description par réseau d'emplacements est de pouvoir décrire plus précisément des lois de circulation dans un environnement où plusieurs robots auraient à cohabiter; celle par description de l'espace libre permet de mieux définir les parties de l'espace autorisées et celles dont la traversée est interdite au robot. On aura par exemple une description en réseau d'emplacements pour des tâches de transport, et une description en espaces libres pour des tâches de nettoyage.

Le projet AMR fournit un exemple de spécification de l'environnement par description de l'espace libre. L'hypothèse du projet AMR [de Saint Vincent, 1990] est également qu'un modèle précis de l'environnement n'est pas toujours disponible a priori. Les informations a priori sur l'environnement sont donc issues d'une connaissance topologique des lieux (couloirs, pièces, portes).

Ces informations sont représentées sous forme d'un graphe de lieux et connecteurs (ou landmarks). Un lieu est un domaine spatial présentant des caractéristiques homogènes pour la navigation du robot : caractéristiques de terrain, modes d'asservissement possibles. Ce sont des pièces et des couloirs. Un connecteur est une entité localisée, reconnaissable par le robot, et qui sert d'étape pour la mission de déplacement. En particulier, le changement de lieu doit être associé à un connecteur. Ces connecteurs sont des portes, des escaliers ou des rampes.

A ce modèle topologique est associée une information géométrique imprécise (sous forme de bornes), concernant :

- les positions relatives des connecteurs,

- les dimensions des lieux,

- les caractéristiques géométriques des différentes entités.

La structure de données est construite en s'appuyant sur des outils graphiques simples utilisant des icônes.

Nous allons préciser dans les sections suivantes les connaissances géométriques, cartographiques et tactiques utilisées par notre système.

4.2.2 Connaissances géométriques

Le système de localisation de notre robot utilise un modèle local dynamique de l'environnement obtenu à partir des capteurs à ultra-sons et du système de vision active, il doit utiliser également un modèle global spécifié par l'opérateur. Le modèle local est utilisé pour détecter les limites de l'espace libre et pour estimer la position courante du robot [Reignier et Crowley, 1991]. Le processus de mise à jour du modèle permet de corriger l'erreur sur la position et l'orientation estimées par mise en correspondance du modèle perçu et du modèle global.

Les informations visuelles et télémétriques sont traitées de manière à produire une description de l'environnement en termes de primitives géométriques exprimées dans un système commun de coordonnées externes. Ainsi, le modèle global est décrit dans le même formalisme. Cependant il n'est pas nécessaire que le modèle global et le modèle local soient décrits par le même type de primitives. Schiele [Schiele et Crowley, 1994] propose une méthode pour faire la localisation à partir d'une grille d'occupation, ce type de représentation étant parfois préféré pour la navigation [Moravec, 1988]. Il montre que, pour la localisation, le processus de mise en correspondance d'une grille d'occupation locale avec une description paramétrique de l'environnement (modèle global) peut être réalisé en temps réel.

L'opérateur ne doit décrire que les objets physiques les plus importants et les moins sujets aux changements de position. Le système de localisation ne doit se repérer que sur les objets fixes mais il est utile de connaître a priori la position d'un objet mobile volumineux qu'il faudrait mieux qualifier, dans la plupart des cas, de déplaçable (n'étant pas autonome). Par contre il n'est pas souhaitable que la description de l'environnement contienne des objets dont la position varie trop. Cela surcharge inutilement la base de données, et la connaissance préalable de ces objets n'est pas utile à la navigation. Ils ne seront perçus et pris en compte que lors de l'exécution.

Les murs sont décrits par la position d'un segment, la hauteur du mur (supposé être une surface plane rectangulaire) et la nature du revêtement. Cette information peut être utilisée pour le choix du capteur servant à la localisation.

Les portes sont décrites afin de pouvoir s'en servir comme repère visuel lors de leur franchissement.

Les objets circulaires ou rectangulaires ont en plus de leurs caractéristiques géométriques un attribut supplémentaire pour indiquer si l'objet est a priori *fixe* ou *mobile*.

4.2.3 Connaissances cartographiques

Afin de simplifier la description de l'environnement et la description des missions, nous structurons l'espace à l'aide d'un réseau d'emplacements et de routes qui les relient; routes que devra emprunter le robot. L'opérateur peut désigner par un nom tous les éléments de description du réseau.

Un emplacement est défini par ses coordonnées cartésiennes 2D par rapport à un repère fixe. Comme il est nécessaire de décrire le passage d'un niveau à un autre dans un bâtiment, il faut définir des emplacements de connexion entre environnements distincts.

Une route relie deux emplacements.

Un lieu est un groupement d'emplacements connexes correspondant ainsi à une structuration en pièces, couloirs et bâtiments.

Une balise est un point de recharge ou de communication associé à un emplacement. Cet emplacement est supposé être aménagé spécifiquement pour ces opérations de recharge ou de communication d'un volume important d'informations.

Un amer est un objet spécifiquement installé pour permettre une localisation du robot à l'aide du capteur vision avec une précision meilleure que la localisation fournie par l'odométrie ou la télémétrie par ultra-sons. Cet objet a la propriété d'être facilement identifiable.

Chaque amer est associé à un lieu qui doit contenir l'ensemble des routes depuis lesquelles l'amer est a priori visible. Ceci afin de simplifier la sélection des amers lors de la planification des tâches de localisation par la vision.

L'accessibilité de chaque élément du réseau d'emplacements et de routes peut varier au cours du temps. On associera donc à chacun de ces éléments des intervalles de temps d'accessibilité. Cette information ne peut venir que de l'opérateur et le degré de précision est faible (la minute).

4.2.4 Connaissances tactiques

Le réseau d'emplacements et de routes fournit la structure de base pour organiser des connaissances dépendantes de la localisation. Les connaissances tactiques concernent les modes de navigation et les vitesses adaptées à chaque route. Elles concernent également la définition d'actions standards associées à des zones.

Une zone est un ensemble d'emplacements non nécessairement connexes. On lui associe un comportement prédéfini du robot.

Les modes de navigation sont les suivants :

- Aller à l'estime : il est soit impossible soit peu nécessaire d'avoir une précision importante en localisation.
- Suivre un amer : pour des tâches qui nécessitent un asservissement sur l'environnement.
- Franchir un passage : dans les passages étroits comme les portes, on doit avoir une bonne précision en position et orientation du véhicule. Ce mode de navigation impose l'utilisation du capteur vision et indique le type d'information à extraire de l'image; les montants de la porte sont considérés comme des amers particuliers.

Les connaissances tactiques utiles à l'exécution d'une mission sont extraites par le superviseur, associées à chaque action et fournies au niveau " action ", les portes sont associées aux routes qui les traversent et les amers aux lieux qui les contiennent.

4.3 Spécification de missions

Les trois niveaux habituellement définis pour l'architecture d'un robot mobile, et sur lesquels l'opérateur peut intervenir sont :

- le niveau planification : description de méta-plans définissant les objectifs et les contraintes d'une mission;
- le niveau contrôle : description de tâches par un langage graphique ou algorithmique ;
- le niveau fonctionnel : commande des moteurs et traitement des données capteur.

Niveau planification Notre étude peut être classée dans cette catégorie. Nous avons défini un langage de spécification d'objectifs. Ce langage permet d'exprimer des contraintes temporelles ou spatiales associées à chaque tâche ainsi que leur interdépendance. Nous en parlerons plus en détail par la suite.

Niveau contrôle On peut citer dans ce domaine les travaux de Noreils [Noreils, 1989]. Ils visent à permettre à l'utilisateur de manipuler les différentes fonctionnalités présentes sur la machine, par le biais d'un formalisme proche du Lisp. Ce formalisme permet les actions suivantes :

- exécution d'ordres destinés aux différents modules (lecture des ultra-sons, suivi d'indice visuel , etc.) ;
- réaction aux événements asynchrones internes (charge batterie, surtension) ou externes (détection d'obstacle, recherche d'objet) par la notion de surveillance;

- parallélisme implicite de tâches.

L'opérateur intervient ici à un assez bas niveau, notamment sur les capteurs et effecteurs. Des travaux ultérieurs [Noreils, 1991] visent à faciliter la programmation de missions en élaborant une interface homme-machine adaptée. Le langage graphique utilisé est de type iconique. Ces travaux tendent à élargir le domaine d'intervention de l'opérateur aux trois différents niveaux définis plus haut.

Niveau fonctionnel Pour la programmation de mission à bas niveau les fabricants de plate-formes mobiles proposent généralement une bibliothèque de fonctions en langage C pour commander de façon rudimentaire les effecteurs et les capteurs. C'est le cas pour Robuter de Robosoft, ou LabMate de TRC.

4.3.1 Langage de spécification d'objectifs

La définition d'un langage pour la description de tâches exprimées en termes d'objectifs pour un robot mobile relève de choix qu'il n'est pas toujours aisé de justifier a priori, vu la jeunesse du domaine. C'est un compromis entre les fonctions de l'application envisagée d'une part et les capacités d'action, de perception et de raisonnement dont le système dispose réellement d'autre part. Le premier point tient compte des attentes d'un utilisateur, le deuxième tient compte d'un certain état de l'art. Nous allons décrire notre langage de spécification et montrer comment ce langage permet d'exprimer des contraintes temporelles ou spatiales ainsi que l'inter-dépendance des tâches.

La mission est constituée de deux types de tâches : les tâches de navigation et les tâches spécifiques à l'application. L'opérateur peut associer à chaque tâche de navigation une contrainte de temps et un critère d'optimisation du trajet afin de minimiser pour cette tâche un des critères suivant : la consommation d'énergie, la durée du trajet, le risque (lié à l'encombrement de la route) et la distance parcourue. Le terme de tâches parallèles recouvre toutes les tâches spécifiques à l'application, ici la surveillance et l'intervention en téléopération.

Une description de ce langage en grammaire BNF est donnée ci-dessous :

<mission> ::= **Contraintes** [<localisation>] [<contrainte de temps>],
[D <distance>,] [E <energie>,] [R <risque>] :
{<tâche de nav.> } $^+$ { <tâche parallèles> } *

<tâche de nav.> ::= **Etre** <emplacement>, [<contrainte de temps>], [<optimisation>];
::= **Patrouiller** <lieu> <temps d'arrêt> [<contrainte de temps>], [<optimisation>

<tâche parallèle>:= **Surveiller** [<lieu>] [<contrainte de temps>] (<événement>,<réaction>);
::= **Faire** [<lieu>] [<contrainte de temps>] <fichier d'actions>;
::= **Téléopérer**[<lieu>] [<contrainte de temps>];
::= **Communiquer** [<lieu>] [<contrainte de temps>] <données>;

Interprétation des contraintes de localisation et des contraintes temporelles

Les contraintes temporelles sont décrites par un intervalle et un mode associé, absolu ou relatif. Pour les tâches de navigation, la contrainte temporelle définit l'emplacement but à atteindre ou le lieu où patrouiller. Pour la contrainte temporelle des tâches de navigation, le mode **absolu** exprime que les bornes de l'intervalle de temps se réferent aux dates données par l'horloge du système. Pour le mode **relatif**, la première valeur exprime une durée relativement à la tâche précédente et la seconde la durée de la tâche elle même. Ces contraintes sont des bornes supérieures.

Dans le cas des tâches parallèles, une contrainte de temps de mode relatif exprime une durée relative à l'instant auquel la contrainte de localisation est satisfaite. Dans le cas où il n'y a pas de contrainte de localisation, cette durée est relative à la date de début d'exécution de la mission. Une contrainte absolue fixe les limites temporelles d'activation de la tâche indépendemment des actions de navigation.

Une tâche parallèle sans contrainte de temps sera exécutée toutes les fois où la contrainte de localisation est satisfaite. Si aucune contrainte n'est spécifiée la tâche parallèle est active pendant toute la durée de la mission. Certaines ne seront réalisées que lorsque le robot est à l'arrêt.

Exemple de mission

Voici un exemple simple de mission, chacun de ces éléments est repris pour être commenté :

---

contraintes entrepôt-12, [09/00/00, 15/30/00] **absolu**;

patrouiller SalleB, 3/00, [20/00, 30/00] **relatif**, **temps**;

être poste2 [10/00, 01/00/00] **relatif risque**;

surveiller SalleB (gaz,alarme);

---

1. La mission doit s'exécuter de 9h à 15h 30, aujourd'hui même dans l'entrepôt 12. Ainsi tous les lieux ou emplacements référencés par la suite doivent être contenus dans le lieu " entrepôt 12". Tous les emplacements intermédiaires choisis dans le plan par le superviseur doivent également appartenir à ce même lieu.

2. Le robot doit arriver sur un emplacement de la "salle B" 20 minutes après le début de l'exécution de la mission en prenant le chemin le plus rapide. Il doit visiter tous les emplacements du lieu et s'y arrêter 3 minutes. Cette patrouille doit durer 30 minutes.

3. Le robot doit arriver à l'emplacement poste2, 10 minutes après la fin de la patrouille et doit rester à cet emplacement pendant une heure. Le trajet planifié doit minimiser le risque. Le risque est lié à une estimation ou une mesure de l'encombrement de chaque route du réseau.

4. Le robot doit prévenir de la présence de gaz pendant qu'il se trouve dans le lieu SalleB.

La spécification de mission, fondée sur le langage ci-dessus, a fait l'objet de la réalisation d'une interface graphique interactive, comme l'illustre la figure 4.2.

4.3.2 Aide à la spécification de mission

Le superviseur doit planifier la mission spécifiée par l'opérateur, et donner en retour d'une part une décomposition des tâches en actions et d'autre part une description de la mission dans le langage de spécification. Cela donne une information globale sur la prévision de l'exécution de la mission. En cas d'échec de planification, le superviseur donne un plan incomplet, du début de la mission jusqu'à la tâche cause de l'échec. L'opérateur intervient alors pour corriger la mission. Sa correction est aidée par une estimation de l'erreur de spécification.

Comme nous le verrons dans le chapitre consacré à la planification, le superviseur n'utilise que les données cartographiques et tactiques décrites grâce à l'IHM. Nous verrons aussi que les tâches de la mission sont planifiées en deux étapes. Tout d'abord les tâches de navigation sont planifiées afin de déterminer le trajet suivi par le robot et ainsi estimer la durée du parcours, son coût énergétique ainsi que son risque. Une fois que ce trajet est déterminé, la planification des tâches parallèles consiste à les placer dans le temps relativement aux actions de navigation qui satisfont les contraintes de localisation des tâches parallèles.

Erreurs de spécification et spécification incomplète

Les erreurs de spécification de mission peuvent être classées en plusieurs catégories :

1. les erreurs syntaxiques : la possibilité de faire des erreurs de nature syntaxique est limitée par l'usage intensif de la désignation directe des objets par la souris.

2. les erreurs sémantiques : ces erreurs ne sont pas traitées par l'interface mais par le superviseur. Celui ci peut déterminer lors de la planification qu'une mission n'est pas réalisable.

Les erreurs sémantiques sont de trois types :

1. les contraintes spécifiées sont trop fortes (temps, énergie, risque, distance, localisation),

2. le réseau de routes et d'emplacements n'est pas connexe,

3. il y a un chevauchement de tâches incompatibles.

De même que le compilateur d'un langage de programmation classique tente d'indiquer au mieux à l'opérateur la localisation et la nature de l'erreur de programmation, le superviseur doit le faire pour une spécification de mission. Mais il y a une différence qui vient de deux points :

1. la spécification peut être incomplète. A part la contrainte de localisation pour les tâches de navigation, toutes les contraintes sont optionnelles.

2. l'opérateur n'étant pas a priori un expert, le superviseur doit l'aider à trouver une spécification correcte. C'est à dire réaliste selon les capacités du robot et les caractéristiques de l'environnement.

La diversité des informations et la complexité des traitements qui permettent de déterminer si une spécification de mission est réaliste ne peuvent pas être maîtrisées rapidement par un opérateur non spécialiste. Dans ce sens il y a une réelle collaboration entre le superviseur et l'opérateur pour spécifier une mission. L'opérateur exprime ce qu'il veut voir réalisé, le superviseur contrôle ce qui peut l'être effectivement a priori, la mission pouvant toujours échouer à l'exécution. Le superviseur fournit donc une aide à la spécification sous deux formes :

Sorties réutilisables - Comme résultat de la planification, l'opérateur dispose non seulement d'un script décrivant la décomposition des tâches en actions mais aussi de la description de la mission dans le langage de spécification, avec toutes les contraintes exprimées. Ce résultat de la planification peut être utilisé comme une nouvelle spécification de cette mission. Nous verrons pour le suivi de mission que ces deux spécifications ne sont pas équivalentes même si les plans sont identiques.

Exemple de spécification :

Contraintes bâtiment-B, E 80 % :

Etre entrée-C, [–, 00:15] Relatif, énergie ;
Etre sortie-D [11:15, 11:30] Absolu, distance ;

Surveiller salle22 (feu, alarme);
Communiquer sortie-D, images;

Le planificateur produit une description de la mission planifiée de niveau tâches et de niveau actions. Le plan de niveau tâches est présenté ci-dessous avec en caractères gras les informations additionnelles apportées par la planification.

Contraintes bâtiment-B,**[10:00 11:30] Absolu, D 1240m, R 432, E 66 %**:

Etre entrée-C, **[00:20**, 00:15] Relatif, énergie;
Etre sortie-D [11:15, 11:30] Absolu, distance;

Surveiller salle22, **[10:20, 10:30] Absolu**,(feu, alarme);
Communiquer sortie-D,**[11:13, 11:22] Absolu** images;

Cela donne à l'opérateur une estimation globale des coûts en temps, énergie, risque et distance pour cette mission. Cette description peut servir de nouvelle spécification et permet ainsi un cycle de mise au point rapide.

Dans ce cas les spécifications définies à partir des estimations obtenues par planification étant plus strictes que les contraintes spécifiées initialement, le système dispose de moins de marge lors de l'exécution. Les risques d'échec sont alors plus grands. Même si les plans produits sont identiques pour les deux spécifications, à l'exécution le contrôleur prend en compte les contraintes spécifiées pour diagnostiquer un échec nécessitant l'intervention de l'opérateur. Celui-ci doit alors redéfinir des objectifs de la mission.

Plans incomplets - En cas d'erreur sémantique détectée lors de la planification d'une tâche, un plan incomplet est calculé approchant au mieux du but spécifié. Ceci afin que l'opérateur puisse mieux comprendre la cause de l'erreur. Ainsi, l'opérateur sait quelles contraintes relaxer ou quelles données cartographiques et tactiques corriger.

4.4 Suivi d'exécution

L'exécution de plan est une forme de contrôle d'exécution symbolique effectuée par le superviseur. Le processus de planification établit un canevas pour la mission qui doit être adapté pendant son exécution. Une telle élaboration est possible en traitant la mission comme une hiérarchie de buts séquentiels, le but de plus haut niveau étant d'accomplir la mission selon les contraintes définies. Le second niveau est d'accomplir chaque tâche selon les contraintes et l'odonnancement prévu. Le niveau le plus bas consiste à atteindre chaque emplacement de l'itinéraire prévu et d'exécuter les actions parallèles.

A chaque niveau du système, un échec peut survenir soit parce que le véhicule est physiquement bloqué, soit à cause de la violation d'une ou plusieurs contraintes. Un échec à un niveau nécessite une replanification au niveau supérieur. Un échec au niveau mission peut être traité par une stratégie par défaut telle que demander l'intervention de l'opérateur ou un retour à la base.

L'opérateur peut interagir à différents niveaux de l'architecture de contrôle. A travers l'interface, l'opérateur peut définir une nouvelle boucle de contrôle de haut niveau qui renforce ainsi la robustesse de l'architecture [Noreils, 1991]. De plus les phases d'intervention de l'opérateur peuvent être spécifiées dans la mission (spécification de la tâche parallèle de téléopération) pour effectuer des opérations difficiles.

L'interface de suivi de mission doit permettre à l'opérateur de :

- initialiser le robot,
- lancer et suivre l'exécution d'une mission,
- interrompre son exécution,
- modifier et reprendre la mission interrompue,
- piloter le robot.

Cette interface permet également de simuler l'exécution d'une mission et aussi d'effectuer une répétition d'une mission précédemment exécutée.

4.4.1 Opérations sur la mission

Le langage de spécification de mission est complété par un ensemble de requêtes nécessaires à toutes les étapes du suivi de mission.

planifier <mission> : pour une mission spécifiée, demande sa planification au superviseur.

exécuter <mission> : pour une mission planifiée, demande son exécution.

interrompre : suspend l'exécution de la mission en cours, autorise l'exécution d'une autre mission. Il n'y a qu'une mission active à la fois.

annuler <mission> : pour une mission en cours ou interrompue (suspendue), l'élimine de l'ensemble des missions prises en compte par le superviseur.

reprendre <mission> : demande la replanification et l'exécution d'une mission interrompue, là où elle a été interrompue.

supprimer <mission> <tâche> : pour une mission donnée, demande la suppression d'une tâche. Si la mission est en cours, cette requête n'est acceptée que pour une tâche en cours ou non exécutée.

4.4.2 Informations reçues

L'IHM offre deux types de vues pour les informations temporelles et spatiales. L'une est utilisée pour les données prédites et l'autre pour les données perçues. L'opérateur peut ainsi les comparer et agir en conséquence s'il juge que le superviseur ne réagit pas de façon adéquate.

Les vues spatiales permettent à l'opérateur de situer le robot dans son environnement. Elles contiennent les informations géométriques, cartographiques et tactiques. L'opérateur peut y observer la trajectoire réelle suivie par robot. Il est trop coûteux de fournir un modèle 3D complet de l'environnement tel qu'il pourrait être construit à partir de la vision. Le système embarqué ne transmet qu'un modèle local 2D de l'espace libre autour du robot et une vue frontale des lignes verticales extraites par le système de vision active [Crowley et Christensen, 1994].

Les vues temporelles de la mission permettent à l'opérateur d'observer sur une échelle de temps variable les instants successifs des diverses localisations du robot et les tâches parallèles associées. On peut distinguer deux parties dans une vue temporelle :

- le passé : le plan présenté est celui qui a été réellement exécuté.
- le futur : cette partie permet de présenter visuellement les modifications apportées au plan d'origine par l'utilisateur ou par le superviseur.

Les informations sur l'environnement perçu par le robot sont représentées dans trois modèles distincts :

Le modèle sensoriel représente l'espace libre autour du robot, déterminé grâce aux capteurs à ultra-sons.

Le modèle local représente l'environnement construit par le système de perception à partir des données du modèle sensoriel.

Le modèle local composite représente la superposition de l'environnement décrit par l'opérateur (encore appelé modèle global) avec le modèle local.

Un historique est créé récapitulant tous les événements importants qui se sont passés au cours de l'exécution d'une mission. Il contient les positions du robot enregistrées periodiquement, les positions d'obstacles contournés, le début et la fin d'exécution des tâches et actions, les comptes-rendus de tous les échecs qui sont survenus, les alarmes demandées et détectées par le contrôleur de surveillance, les comptes-rendus des auto-tests, les changements de mode (automatique, télé-opéré et apprentissage). Un filtrage adéquat de cet historique permet de retrouver rapidement la cause d'un échec d'exécution. Il permet également de répéter l'exécution d'une mission en simulant le robot par les messages qu'il a transmis.

Les figures 4.1 et 4.2 représentent l'interface pour la spécification de l'environnement et de la mission.

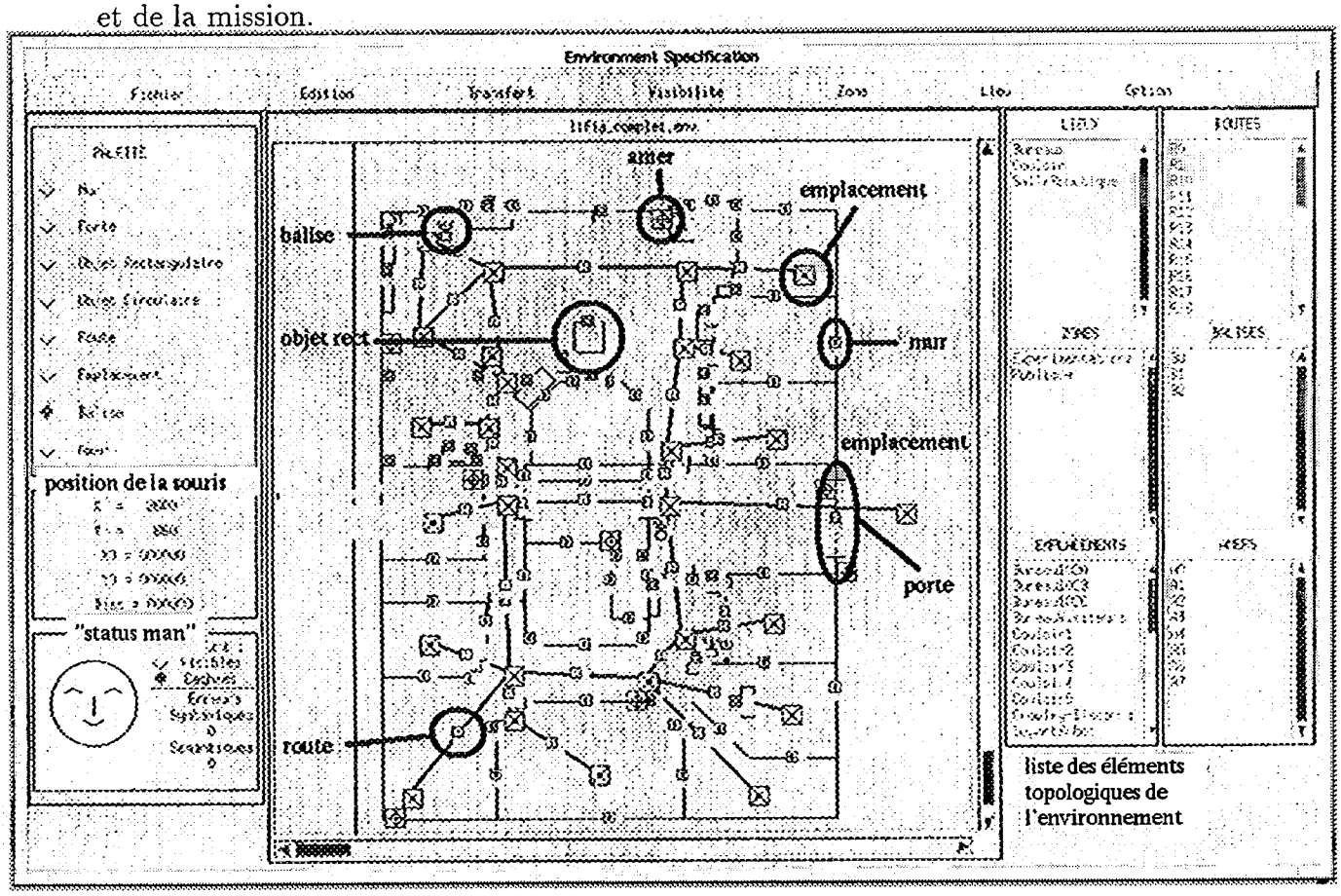


FIG. 4.1 - *Interface pour la spécification d'environnement.*

4.5 Principes mis en œuvre pour la réalisation de l'interface

Dans [Abowd *et al.*, 1992] les auteurs définissent une liste de propriétés des interfaces homme-machine afin de les évaluer. Cette analyse repose sur un modèle simple et général de l'utilisateur. Ainsi la formalisation de ces propriétés peut se définir uniquement en termes du modèle du système car elle repose sur un raffinement centré tâche des propriétés d'un système interactif.

Ces propriétés sont reliées à trois aspects de l'interaction orientée but : complétude des buts[1], flexibilité d'interaction et robustesse d'interaction. Le premier point est trop général. Il est impossible de prouver formellement qu'un tel système répond parfaitement à ses spécifications. Nous allons donc nous concentrer sur les principes retenus pour notre application parmi ceux suggérés par [Abowd *et al.*, 1992].

1. Le système permet de réaliser tous les buts identifiés de l'utilisateur pour une application donnée.

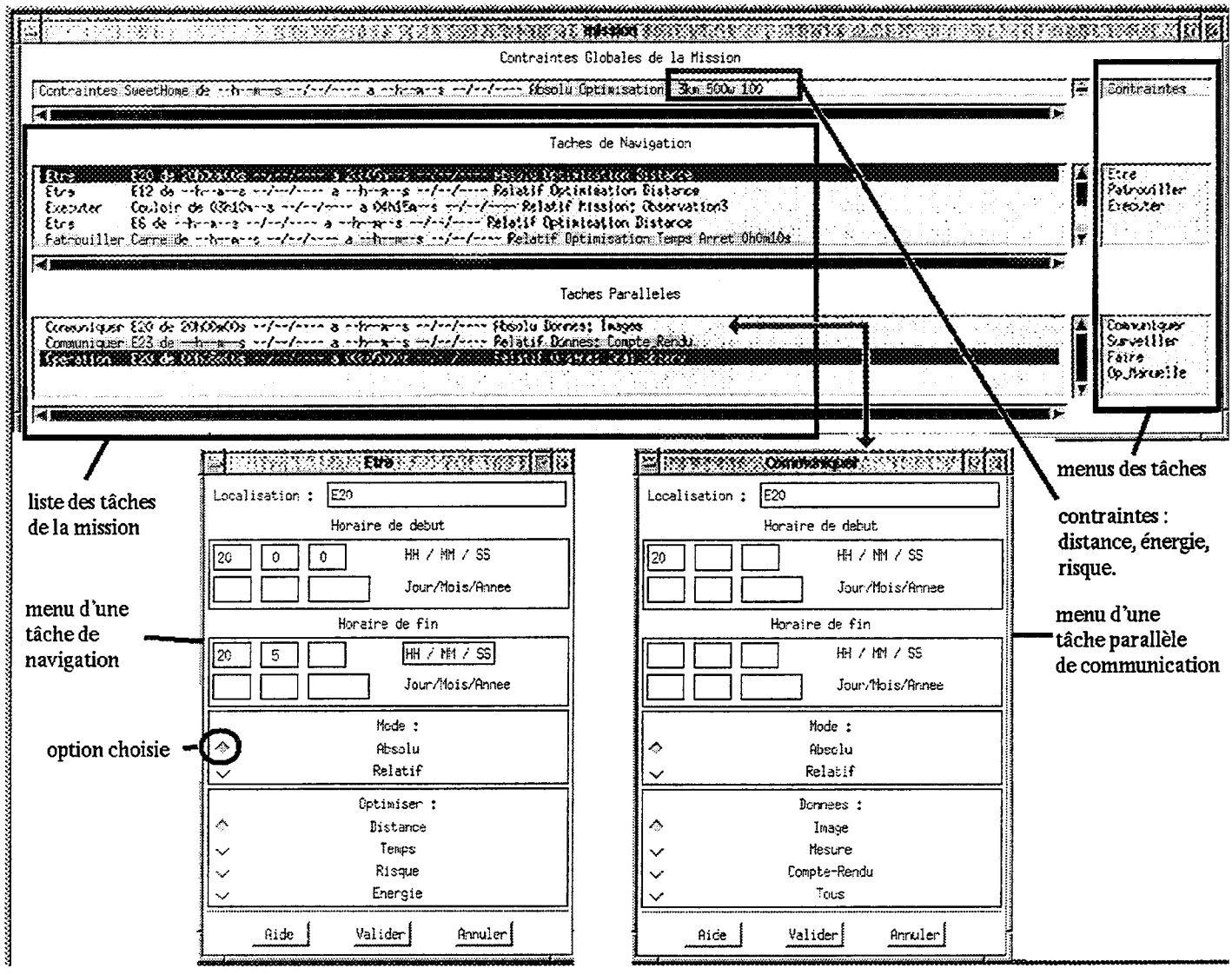


FIG. 4.2 - *Interface pour la spécification de mission.*

4.5.1 Flexibilité d'interaction

Pour une interaction flexible il faut que le système maximise le choix de l'utilisateur pendant l'exécution d'une action.

Le cheminement de la pensée de l'utilisateur et le séquencement de ses actions[2] ne doit pas être figé par le système, en conséquence ce dernier doit être :

- non préemptif : l'utilisateur a le choix de l'action suivante,
- à plusieurs fils d'activités : l'utilisateur peut réaliser plusieurs tâches concurremment,
- entrées réutilisables pour accélérer les saisies lors des phases de spécification.

Des propriétés additionnelles sont utiles pour faciliter la compréhension du fonctionnement du système ou pour sa mise au point.

- la représentation multiple : un même concept peut se présenter selon différentes formes. Dans notre domaine ces formes seront textuelles, spatiales, temporelles ou paramétriques. Ce principe est utile aussi bien lors des spécifications que lors du suivi de mission,
- égale opportunité : le système offre plusieurs possibilités pour la saisie de données concernant une même entité, elles seront complétées par le système pour produire une description correcte de cette entité,[3]
- sorties réutilisables : ceci permet l'utilisation de valeurs calculées par le système comme nouvelles données d'entrée, et donc un cycle de mise au point de ces valeurs d'entrée. Nous avons constaté l'utilité de ce principe pour la spécification de mission,
- transfert d'initiative ou " migrabilité " : Le transfert d'initiative est utile pour le suivi d'exécution. Dans l'interaction avec un processus semi-autonome distant qui agit dans le monde réel, le transfert d'initiative permet a l'opérateur ou au système distant de prendre le contrôle sur l'autre pour l'empêcher de faire une erreur.

Les sections précédentes ont montré comment le superviseur est impliqué pour la réalisation de ces propriétés.

2. Distinction entre tâche et action pour l'I.H.M. : Si la tâche est : "détruire un fichier", les actions sont les opérations de base du système : désigner le fichier, lui appliquer la commande et confirmer sa destruction.

3. Exemple d'égale opportunité : un segment peut être défini par un ensemble redondant de paramètres, la position des extrémités et la longueur du segment. L'opérateur peut donner aussi bien la position des deux extrémités, le système détermine alors la longueur, ou l'opérateur donne un point et la longueur, le système calcule la position de l'autre extrémité.

4.5.2 Robustesse d'interaction

Une interaction robuste suppose que l'utilisateur puisse observer l'effet de l'exécution de ses actions et qu'en cas d'erreur il ait les moyens de se retrouver dans une situation satisfaisante.

Observabilité et honnêteté : le système rend perceptibles tous les états internes pertinents par rapport à l'activité en cours et leur représentation doit être conçue pour être interprétée correctement.

Prédictibilité : l'utilisateur peut prédire les états futurs et les temps de réponse du système depuis l'état perçu courant.

Ces deux propriétés sont très difficiles à obtenir pour l'interface avec un système semi-autonome distant. Comme nous le verrons par la suite elles reposent sur la conception du noyau fonctionnel, c'est à dire tout le système embarqué. Le modèle d'interaction choisi pour réaliser ces propriétés est décrit dans la section suivante.

4.6 Le modèle PAC

De nombreux auteurs identifient différents niveaux d'abstraction dans les systèmes interactifs, ces composants logiques doivent être organisés dans une architecture logicielle. On peut traiter ces niveaux d'abstractions en couches ou alors décentraliser les niveaux logiques. Cette approche est la base des modèles multi-agents. Nous avons conçu notre I.H.M. en suivant le modèle multi-agents PAC proposé par J. Coutaz [Coutaz, 1987].

Le modèle PAC structure récursivement un système interactif en une hiérarchie d'agents. Un agent définit une compétence à un certain niveau d'abstraction. Cet agent se compose de trois facettes appelées: présentation, abstraction et contrôle quel que soit le niveau d'abstraction:

1. la présentation est le comportement perceptible de l'agent,
2. l'abstraction contient le noyau fonctionnel de l'agent, réalisé indépendamment du média géré par la présentation,
3. la partie contrôle relie l'abstraction à la présentation, contrôle le comportement et la cohérence de ces facettes, mémorise un état local pour assurer un dialogue à plusieurs fils d'activité et assure les relations avec les autres agents.

Le modèle PAC permet de mettre en place une hiérarchie de niveaux d'abstraction des agents d'interaction et les relations existant entre ces niveaux. Cette hiérarchie peut être interprétée du plus haut niveau jusqu'au plus bas.

- La racine de la hiérarchie: à ce niveau, la partie abstraction correspond au noyau fonctionnel du système interactif. Dans le cas de la spécification et du suivi de

mission, l'abstraction contient le superviseur. La partie contrôle fournit le mécanisme d'indirection qui permet de faire le lien entre l'identité des objets gérés par le superviseur et ceux gérés par la partie présentation. Le contrôle gère l'état d'interaction défini par le noyau fonctionnel. Il vérifie par exemple l'ensemble des fonctions valides, dans le cas du "suivi d'exécution", il gère le mode de fonctionnement du robot: automatique ou télé-opéré. La partie présentation est à ce niveau le "gestionnaire de fenêtres" qui doit maintenir les relations spatiales entre les présentations des différents sous-agents.

- Les feuilles de cette hiérarchie sont constituées d'un agent PAC élémentaire. C'est la plus petite unité de l'interface avec laquelle l'opérateur peut interagir. Par exemple, pour la spécification d'environnement, on associe à chaque mur un agent PAC. La partie abstraction contient les notions qui définissent un mur dans l'environnement réel, notamment ses dimensions exprimées en mètres. La partie présentation définit la forme, la taille et la position en pixels de l'objet graphique représentant le mur. La partie contrôle effectue les traductions entre les deux représentations ainsi que les transformations entre les systèmes de coordonnées abstrait et graphique. Il mémorise l'état de l'interaction entre l'opérateur et l'agent. Cela autorise ainsi plusieurs fils d'activités pour l'opérateur

- Les niveaux intermédiaires de la hiérarchie concernent la représentation des combinaisons et des relations entre agents, réalisant les actions syntaxiques telles que la visibilité ou la mise à l'échelle d'un groupe d'agents PAC.

En résumé, le modèle PAC propose une méthodologie qui permet de mettre en œuvre les propriétées liées à la flexibilité de l'interaction. La mise en œuvre des autres propriétés attendues relèvent de choix de conception du noyau fonctionnel.

4.7 Conclusion

Dans ce chapitre nous avons présenté une interface homme-machine conçue pour l'interaction entre un opérateur et un robot mobile autonome. Nous avons défini les connaissances fournies par l'opérateur pour que le système puisse fonctionner ainsi que les interactions entre cet opérateur et le système. Pour cela nous avons proposé un langage de spécification d'objectifs pour la description de mission de surveillance en tentant de distinguer ce qui peut être général pour les domaines d'application d'un robot mobile, les tâches de navigation, de ce qui est spécifique à l'application, ici la surveillance.

Le modèle choisi pour réaliser cette interface donne les bases pour une interaction efficace et permet de mettre en œuvre des propriétés considérées comme fondamentales pour une bonne interaction homme-machine. Nous ne pouvons proposer qu'une maquette et il reste du travail pour atteindre la réalisation complète de nos objectifs. Cependant quelques exemples ont été détaillés pour montrer l'intérêt d'une phase de conception simultanée de l'interface homme-machine et du superviseur, alors que très

couramment encore, pour des projets de robotique mobile, la conception de l'interface vient après la conception du noyau fonctionnel de l'application - ici le Superviseur - et n'interfère pas dans la conception de ce dernier.

Ainsi les chapitres architecture et interface homme-machine nous ont permis de décrire en quelque sorte les spécifications externes du Superviseur. La conception de celui-ci est détaillée dans les chapitres suivants concernant la planification et le contrôle d'exécution.

Chapitre 5

Planification de mission

Ce chapitre présente une technique pour planifier les missions de surveillance telles que nous les avons présentées dans le chapitre précédent. Cette technique s'applique donc à un environnement discrétisé en réseau d'emplacements et de routes. La mission affectée au robot est composée de deux parties : les tâches de navigation et les tâches opérationnelles, aussi appelées tâches parallèles. Les tâches de navigation définissent la suite des « points de passage » obligés de l'itinéraire à planifier. La spécification de mission permet de poser des contraintes opérationnelles sur les tâches et la spécification d'environnement permet d'associer des contraintes temporelles d'accessibilité à chaque élément du réseau.

L'algorithme de planification d'itinéraire présenté effectue la recherche d'un chemin optimal pour chaque tâche de navigation selon le critère d'optimisation et les contraintes de chacune d'elles en tenant compte des contraintes sur l'environnement et des possibilités de réinitialisation de certains paramètres. Les tâches parallèles sont ensuite décomposées en actions selon l'itinéraire planifié. Une fois que toutes les actions parallèles sont planifiées, elles sont ordonnées par rapport aux actions de navigation dans un graphe d'exécution.

5.1 Introduction

Dans notre architecture nous séparons le problème de la planification d'itinéraire de celui de la génération de trajectoire. Le problème de l'évitement d'obstacles, que ceux-ci soient statiques ou dynamiques, n'est pas résolu au niveau d'abstraction du superviseur mais par le contrôleur de véhicule.

Si nous ne tenons pas compte, pour la planification des missions, du fait que l'environnement est *dynamique*, nous considérons plus globalement que son *accessibilité* varie au cours du temps. Nous allons voir dans quelle mesure cette hypothèse diffère de celle étudiée pour les environnements dynamiques. Pour ces environnements, les recherches actuelles sont en fait dérivées des méthodes de planification d'itinéraire dans l'espace des configurations [Kant et Zucker, 1988; Fugimura et Samet, 1989; Pan et Luo, 1990; Fraichard, 1992]. Elles supposent que les obstacles mobiles sont des polygones ou des disques, se déplaçant à vitesse constante (et connue) dans une direction fixe (connue).

Les méthodes proposées utilisent une structure hiérarchisée en deux niveaux définis par une décomposition chemin / vitesse.

1. recherche du chemin optimal par frôlement des obstacles statiques dans l'espace des configurations ;

2. planification d'un profil de vitesse sur les segments du chemin optimal, afin d'éviter les collisions avec les obstacles mobiles croisant ce chemin.

Ainsi les obstacles mobiles dans l'espace (x,y) sont statiques dans l'espace (abscisse curviligne, temps). Sur le chemin trouvé à la première étape, [Pan et Luo, 1990] préfèrent planifier des périodes d'attente, alors que [Kant et Zucker, 1988] déterminent un profil de vitesse que le robot devra suivre, [Fraichard, 1992] détermine un profil de vitesse et un éventuel changement de voie de circulation.

Cependant, comme dans notre architecture, il faut que le bas niveau réalise l'exécution de la trajectoire en assurant l'évitement d'obstacles imprévus en temps réel. Dans [Kant et Zucker, 1988], quand un obstacle imprévu est détecté sur la trajectoire, une accélération radiale à cet obstacle est générée et vient se rajouter au mouvement global. Ce système permet de s'affranchir des petites incertitudes sur le mouvement des obstacles. Par contre si la réaction à un événement imprévu est telle qu'elle empêche la réalisation d'un sous-but, il faut alors appeler le planificateur pour éventuellement revoir le chemin global.

Ces méthodes, même si elles apportent un plus par rapport au cas statique, ont un champ d'application qui reste très restreint car une bonne connaissance a priori du mouvement des obstacles mobiles est nécessaire. Ce qui est généralement difficile sinon impossible dans un environnement réel, surtout quand les obstacles mobiles ne sont pas également des robots.

Notre but est de traiter dans un même formalisme le problème de la recherche d'un chemin optimal et celui de la satisfaction des contraintes opérationnelles ou liées à l'environnement. Pour cela nous considérons deux types de fonctions de coût : les fonctions de coût croissantes (pour le temps ou la distance) et les fonctions de coût croissantes par morceaux. Ce dernier type permet de gérer, par exemple, l'énergie ou l'incertitude de localisation. La contrainte d'énergie concerne l'énergie minimale devant être disponible pendant la mission. Si le robot est capable d'effectuer des opérations de recharge à des emplacements spécifiquement aménagés pour cela, la consommation d'énergie peut être considérée comme un coût croissant par morceau entre deux opérations de recharge. De la même manière, on considère la variation de l'incertitude en position le long d'une trajectoire comme un coût pour cette trajectoire. Si aucune relocalisation ne peut être effectuée, ce coût est strictement croissant. Observer un amer à une place spécifique permet de réduire cette incertitude. Si l'on peut prédire ou mesurer la variation de l'incertitude en position pour chaque trajectoire prédéfinie, on peut déterminer une séquence de trajectoires qui minimise cette incertitude et choisir un seuil sur cette incertitude pour forcer l'observation d'amers.

Le problème que nous considérons diffère selon plusieurs points des problèmes précédemment résolus en planification d'itinéraire :

- La connaissance disponible sur l'environnement est plus globale que celle utilisée pour la planification en *environnement dynamique*. Nous n'avons pas connaissance des obstacles mobiles mais seulement des intervalles de temps d'accessibilité de certaines parties de l'environnement.

- Les actions de navigation envoyées au contrôleur de véhicule sont définies par la position à atteindre et une vitesse à maintenir. Le planificateur utilise une vitesse nominale définie pour chaque route. Les variations éventuelles de vitesse sont déterminées par le contrôleur de véhicule, responsable de l'évitement d'obstacles. Afin de satisfaire les contraintes d'accessibilité, le planificateur détermine un temps d'attente sur chaque emplacement.

- Les contraintes d'accessibilité sont supposées connues sur la durée totale de la mission. Il est alors possible d'inclure le temps d'attente dans le coût temporel du trajet et choisir le plus rapide. La méthode proposée par [Pan et Luo, 1990] doit énumérer tous les chemins possibles, notre méthode est basée sur une recherche du type *meilleur d'abord* guidée par une heuristique.

- Pour planifier des actions de réinitialisation comme *recharger* ou *relocaliser*, nous devons considérer des étapes de retour-arrière afin de satisfaire les contraintes associées, tout en minimisant le nombre de ces actions. Les trajets possibles contenant ces actions ne doivent être évalués que s'il est impossible de trouver un trajet, satisfaisant les contraintes, qui ne les contienne pas. Le temps nécessaire à de telles actions est inclus dans le coût temporel du trajet. Pour limiter la complexité de la planification, le problème de satisfaction des contraintes est résolu pour chaque tâche indépendamment, chaque tâche étant planifiée selon son propre critère d'optimisation. Quand la planification échoue pour une tâche, les tâches précédentes ne sont replanifiées que si la solution consiste à insérer dans leurs plans des actions de réinitialisation. Elles ne sont pas replanifiées si l'échec correspond au dépassement de contrainte d'un coût non réinitialisable.

- Afin de permettre la gestion de l'incertitude des coûts d'exécution, le planificateur détermine les marges disponibles pour chaque paramètre. Ces marges sont définies par la différence entre les contraintes imposées par l'opérateur et le coût prévu par la planification. Ces marges seront utilisées lors de l'exécution.

On trouve dans [Gondran et Minoux, 1985] une formulation algébrique du problème du plus court chemin avec contraintes temporelles dans laquelle on associe des intervalles d'accessibilité aux éléments du graphe. Cette formulation matricielle permet d'appliquer les méthodes de résolution algébriques mais cette formulation est mal adaptée dans notre cas, à cause des différents types de contraintes et optimisations à gérer. Pour des graphes de très grandes dimensions, pour lesquels les critères de validité d'un chemin sont multiples, il s'avère plus simple et plus efficace d'envisager une

méthode de parcours de graphe guidé par une heuristique plutôt qu'une formulation algébrique.

5.2 Formulation du problème

L'espace de travail du robot est structuré en un réseau d'emplacements et de routes. L'information sur l'environnement utilisée pour la planification concerne les contraintes d'accessibilité de chaque élément du réseau, les coûts de navigation de chaque route et les emplacements spécifiques où le robot peut effectuer des actions de réinitialisation : recharges ou relocalisations.

5.2.1 Description de l'environnement

Le réseau : Pl est l'ensemble des emplacements connectés par des routes ; Ro est l'ensemble des routes.

L'accessibilité : Pour chaque élément du réseau, les contraintes d'accessibilité sont représentées par un ensemble d'intervalles disjoints. Pour $e \in Pl$ (resp. $r \in Ro$), l'ensemble des intervalles $\Phi(e) \subset [0, \infty[$ (resp. $\Phi(r) \subset [0, \infty[$) définit quand e (resp. r) est accessible.

Les coûts de navigation : À chaque route $r_{i,j}$ reliant les emplacements i et j, on associe un vecteur de coûts positifs $w(r_{i,j})$ correspond à la longueur de la route, le temps de parcours, la quantité d'énergie nécessaire, le facteur de risque, et la variation d'incertitude en position. Pour chaque route, une vitesse nominale V_{nom} est donnée.

Les emplacements de réinitialisation : Un emplacement de réinitialisation est un emplacement où un coût c peut être réinitialisé à une valeur minimale. La durée de l'action est $t_{init}(c)$. L'ensemble des emplacements est noté $RPL_c \in Pl$.

5.2.2 Tâches de navigation

Une mission contient une liste de tâches de navigation $(T_1, T_2, \ldots, T_n)$.

Une tâche notée T_k est définie par un but, un critère d'optimisation op et un vecteur de contraintes C. Le but de la tâche est décrit par une localisation à atteindre, notée $loc(T) \in Pl$ et par une durée d'attente spécifiée à cet emplacement, notée $das(T) \in [0, \infty[$.

Un critère d'optimisation noté op est choisi pour chaque tâche de navigation parmi l'ensemble des paramètres :

$$\text{PARAMETRES} = \{\text{distance, temps, énergie, risque, localisation}\}$$

Un vecteur de contraintes noté $C \in \mathbb{R}^{p+}$ avec $p = |\text{PARAMETRES}|$.

Pour les paramètres considérés, les contraintes associées ont la signification suivante :

1. C_{distance} définit la distance maximum que le robot est autorisé à parcourir.
2. C_{temps} définit la durée maximale de la tâche.
3. C_{risque} définit le risque maximal que peut accepter le robot.
4. $C_{\text{énergie}}$ définit la quantité maximale d'énergie utilisable entre deux opérations de recharge. Nécessairement $C_{\text{énergie}} < E_{max}$ où E_{max} est la quantité d'énergie maximale dont peut disposer le robot.
5. $C_{\text{localisation}}$ définit l'incertitude en position maximale autorisée entre deux actions de relocalisation.

Pour simplifier, toutes les tâches auront les mêmes contraintes de distance, énergie, risque et localisation. Comme les coûts seront évalués cumulativement depuis le début de la mission, ces contraintes sont prises en compte comme des contraintes globales à la mission.

5.2.3 Définition des actions de navigation

Chaque tâche est décomposée en une séquence d'actions ou plan $Plan(T_k) = (A_{k,1}, \ldots, A_{k,l})$. Les différents types d'actions sont :

Aller⟨place⟩,

Rester⟨durée⟩ et

Reinit⟨paramètre⟩.

La planification permet de calculer un vecteur de coûts $w \in \mathbb{R}^{p+}$ pour chaque tâche et chaque action. Le coût d'une action A est déterminé selon son type :

- $A = Aller\langle e_2 \rangle$ depuis e_1 par la route r_{e_1,e_2} alors $w(A) = w(r_{e_1,e_2})$.
- $A = Rester\langle \text{durée} \rangle$ alors $w_{temps}(A) =$ durée et nul pour les autres paramètres.
- $A = Reinit\langle \text{paramètre} \rangle$ alors $w_{temps}(A) = t_{init}(\text{paramètre})$ et nul pour les autres paramètres.

5.3 Algorithme de recherche d'itinéraire

L'algorithme de recherche d'itinéraire s'inspire de l'algorithme A*. Il s'en distingue pour résoudre le problème de la satisfaction des différentes contraintes. Cela impose de garder l'ensemble des meilleurs chemins non totalement comparables et ne retenir que le meilleur satisfaisant toutes les contraintes, pour le critère d'optimisation choisi.

Étant donné un graphe G et une fonction de coût g définie sur les nœuds adjacents dans G, A* garantit de trouver un chemin de coût minimum entre n'importe quel couple de nœuds de G. De plus, A* est optimal dans le sens où il détermine le meilleur chemin en examinant le nombre minimum de nœuds nécessaire pour garantir que ce chemin est de coût minimal. Pour le problème de planification de chemin, l'heuristique est en général la distance. Dans notre cas, les coûts sont estimés par des fonctions de la distance.

Pour examiner le minimum de nœuds, le critère de sélection de A* consiste à éliminer un nœud s'il trouve un nœud de coût inférieur pour le même état. Notre problème de satisfaction de contraintes multiples conduit à comparer des vecteurs de coûts qui ne sont pas totalement comparables. Pour définir un critère de sélection nous utilisons un ordre partiel à l'aide de la notion de *coût efficace*:

Définition 1 Un vecteur de coût $w \in \mathbb{R}^p_\infty = (\mathbb{R} \cup \{+\infty\})^p$ est *efficace* comparé à un sous-ensemble W de $\mathbb{R}^p_\infty$ si aucun vecteur $w' \in W, w' \neq w$ n'a des composantes inférieures ou égales à celles de w.

5.3.1 Définition de l'espace d'états et de l'arbre de recherche

Les notations classiques pour A* sont empruntées à N. Nilsson et J. Pearl.

Un état γ de notre graphe d'états est défini par le couple $\gamma = (e, \phi)$ où e identifie l'emplacement à atteindre et ϕ l'intervalle de temps, ou *fenêtre temporelle*, tel que $\phi \in \Phi(e)$. Le domaine sous-jacent est un réseau d'emplacements et de routes sur lesquels les intervalles d'accessibilité sont donnés. L'algorithme doit trouver un itinéraire (une séquence d'actions de navigation) partant de l'état initial γ_0^0 pour la première tâche à l'état but $\gamma_\star^n$ de la dernière tâche de navigation de la mission. L'itinéraire pour chaque tâche T_k est défini de l'état γ_0^k à l'état $\gamma_\star^k$. L'emplacement $e(\gamma_\star^k)$ est le but de la tâche T_k et la contrainte de temps associée à cette tâche doit être compatible avec la fenêtre temporelle de cet emplacement.

Pour explorer cet espace d'états et trouver une séquence d'actions de coût minimal pour chaque tâche, nous construisons un graphe de recherche G dont les nœuds sont définis par les attributs et fonctions suivantes (pour un nœud n):

- l'état atteint $\gamma(n) = (e, \phi)$.
- le type du nœud: $type(n) \in \{aller, rester, reinit\}$, il détermine l'action à exécuter.
- le nœud père $p(n)$ dans l'arbre de recherche. Si n est un successeur du nœud m alors $p(n) = m$.

- $\pi(n)$ l'indice du chemin γ_{k-1}, γ_k calculé pour la tâche T_k.
- $a(n) \in \mathbb{R}^+$, l'instant où le nœud est atteint.
- $d^-(n) \in \mathbb{R}^+$, la borne inférieure des instants de départ possibles de $p(n)$ pour atteindre n.
- $d^+(n) \in \mathbb{R}^+$, la borne supérieure des instants de départ possibles de $p(n)$ pour atteindre n.

5.3.2 Fonctions de coût

Plusieurs nœuds peuvent être décrits par un même état mais avoir des vecteurs de coûts différents. Par extension de la définition 1, nous introduisons la notion de *chemin efficace*.

Définition 2 Un *chemin efficace* est un chemin d'un nœud n à un nœud m pour lequel le vecteur de coûts en m est un *vecteur de coûts efficace* par rapport aux autres vecteurs de coûts des chemins de n à m.

En supposant que n_0 est l'unique nœud de l'état initial γ_0, pour un nœud n et un critère c, la fonction $g_c(n)$ détermine le coût de c en n pour le chemin $P(n_0, n)$.

À chaque nœud n correspond une action A, ainsi la définition du coût élémentaire d'une action $w(A)$ peut être étendue pour le type de nœud correspondant. La fonction de coût g est alors définie par :

$$\begin{array}{rccl} g: & G & \rightarrow & \mathbb{R}^{p+} \\ & n & \rightarrow & \sum_{m \in P(n_0,n)} w(m) \end{array}$$

Cette fonction est monotone croissante sur le chemin $P(n_0, n_\star)$.
Tous les coûts sont croissants avec la distance parcourue mais certains peuvent être réinitialisés à une valeur minimale quand l'action correspondante peut être effectuée. La valeur minimale de réinitialisation peut dépendre de l'emplacement où est effectuée cette action. Elle est notée $Min_c(e)$.

Soit $\tilde{g}$ la fonction de coût définie pour $n \in P(n_0, n_\star)$ par :

$$\begin{array}{rccl} \tilde{g}: & G & \rightarrow & \mathbb{R}^{p+} \\ & n & \rightarrow & [\tilde{g}_1(n), \ldots, \tilde{g}_p(n)] \end{array}$$

L'ensemble des paramètres, de cardinalité p est partitionné en deux: $PARAMETRES = PARAM_1 \cup PARAM_2$, où $PARAM_1$ est l'ensemble des paramètres dont la fonction de coût associée est strictement croissante et $PARAM_2$ est l'ensemble des paramètres dont la fonction de coût associée est croissante par morceaux. On définit les fonctions $\tilde{g}_i$ de la manière suivante :

$$\text{Pour } i \in PARAM_1 \ : \ \tilde{g}_i(n) = \tilde{g}_i(p(n)) + w_i(n)$$

Cette définition est équivalente à celle de la fonction g. Pour $i \in PARAM_2$:

$$\begin{cases} \tilde{g}_i(n) = Min_i(e(n)) & \text{si } type(n) = reinit_i, \\ \tilde{g}_i(n) = \tilde{g}_i(p(n)) + w_i(n) & \text{sinon.} \end{cases}$$

Les fonctions classiques d'estimation de coûts f et h sont définies pour chaque paramètre. Pour planifier la tâche T_k, les fonctions heuristiques h_i sont calculées de l'état courant $\gamma(n)$ à $\gamma_\star^k$. Afin dévaluer l'efficacité de la procédure de recherche d'itinéraire, nous introduisons la terminologie suivante :

- $k_i(m,n)$ est le coût d'un chemin optimal de m à n, pour le critère i,
- $h_i^*(n) = k_i(n, n_\star)$ est appelée l'heuristique *parfaite.*

Les heuristiques choisies sont des fonctions croissantes avec la distance parcourue, ainsi elles satisfont les propriétés suivantes [Hart *et al.*, 1968; Pearl, 1981] :

1. $\forall n \in G, h_i(n) \leq h_i^*(n)$, alors h_i est une heuristique *admissible*,
2. $\forall n, m \in G, h_i(n) \leq k_i(n,m) + h_i(m)$ et $h_i(n_\star) = 0$, alors h_i est une heuristique *consistante.*

La fonction d'estimation $f_i(n)$ est donnée par :

$$f_i(n) = \tilde{g}_i(n) + h_i(n).$$

5.3.3 Génération des successeurs

Un nœud est défini par $\gamma(n) = (e_i, \phi_i)$.

- Soit τ_0 l'instant de début de la mission. L'instant auquel un nœud n est atteint est donné par : $\tau_n = \tau_0 + g_t(n)$.
- L'emplacement $e(n)$ est noté avec l'indice i : $e(n) = e_i$. L'accessibilité à cet emplacement est donné par :

$$\tau_n \in \phi_i, \text{ avec } \phi_i \in \Phi(e_i)$$

Les successeurs possibles du nœud n sont définis par les règles suivantes :

- Si l'emplacement e_i appartient à RPl et $type(n) \neq reinit_c$, alors un successeur possible de n est un nœud de type $reinit_c$ défini par le même état. La contrainte suivante doit être satisfaite, stipulant que la durée de l'action de réinitialisation doit être contenue dans la fenêtre temporelle du nœud n :

$$[\tau_n, \tau_n + t_{reinit}(c)] \in \phi_i$$

```
procédure Itineraire(G,γ0,{T1,...,Tn})
début
n0 := Init(γ0, τ0);
InitGraphe(G, n0);
OUVERT := {n0};
FERME := ∅;
REINIT[i] := ∅ pour i ∈ PARAM2;
n := n0;
Π := 1;
pour T ∈ T1,...,Tn faire
    π(T) := Π;                          /* indice du nouveau plan de T. */
    ρ := 0;                             /* indice du plan qui echoue. */
    nechec := nil;                      /* nœud de reference. */
    pechec := nil;                      /* parametre cause de l'echec. */
    pour i ∈ PARAMETRES faire
        echec[i] := Faux;               /* parametre i cause d'un echec. */
        Emin[i] := +∞;                  /* pour la redefinition des contraintes. */
        K[i] := 0;                      /* pour le calcul des contraintes finales. */
    fin pour
    tant que non (EtatBut(n, T)) faire
        SCS := Successeurs(n);
        Fermer(n);
        pour n ∈ SCS faire
            Calcul des gi(n), g̃i(n), hi(n), fi(n) pour i ∈ PARAMETRES;
            Verifier(n, T, ρ);
            SelectEfficace(G, n, T);/* Elimination des nœuds hors d'un chemin efficace */
        fin pour
        tant que (OUVERT/π(T) = ∅) faire
            si (T ≠ T1 et ρ ≠ 0 et ContraintesFinaleValides(K, ρ, T)) alors
                T := Redefinir(Pred(T));
            sinon
                ContruirePlan(G, nechec);   /* Construction d'un plan incomplet */
                Retourner(Faux);            /* Echec de la planification */
            fin si
        fin tant que
        n := meilleur(OUVERT/π(T));         /* fop(n) est minimum */
    fin tant que
    Π := Π + 1;
    n := CreerSucc(n, rester, Π);           /* Action "rester" de T */
    T := Succ(T);
fin pour
ConstruirePlan(G, n);
Retourner(Vrai);                            /* Succes de la planification */
fin
```

FIG. 5.1 - *Pseudo-code de la procédure : Itineraire*

```
procédure EtatBut(n, T)
début
si e(n) = loc(T) alors
    Fermer(n);
    si [a(n), a(n) + das(T)] ⊄ φ(n) alors
        Retourner(Faux)
    fin si
    pour i ∈ PARAM_2 faire
        si g̃_i(n) < CF_i[T] alors
            Retourner(Vrai)
        fin si
    fin pour
    Eliminer(n);
fin si
Retourner(Faux)
fin
```

FIG. 5.2 - *Pseudo-code de la procédure : EtatBut*

- Si l'emplacement e_i est connecté à un emplacement e_j par la route r_{ij}, les successeurs de n sont des nœuds de type *aller* définis par l'état (e_j, ϕ_j) où ϕ_j appartient à l'ensemble des fenêtres temporelles pendant lesquelles cette place est accessible depuis e_i selon les contraintes d'accessibilité $\Phi(r_{ij})$ et $\Phi(e_j)$.

Les étapes suivantes décrivent comment déterminer les successeurs d'un nœud, pour l'action *aller*, qui soient compatibles avec les contraintes d'accessibilité et les contraintes temporelles de la tâche courante.

- L'intervalle des départs possibles de e_i vers tout autre emplacement est donné par :

$$d_i = [\tau_n, C_t[\cap \phi_i$$

où C_t est la contrainte temporelle de la tâche courante.

- L'ensemble D_{ij} des intervalles possibles pour aller de e_i à e_j est :

$$D_{ij} = \{d_i\} \cap \Phi(r_{ij})$$

- On définit un opérateur externe $\oplus$ de décalage d'un intervalle de temps :

$$I \in \mathcal{I}, \mathcal{T} \in \mathbb{R}^+, \mathcal{T} \oplus I = \{t + \mathcal{T} / t \in I\}$$

- Connaissant $w_{temps}(r_{ij})$ l'estimation du temps nécessaire pour aller de e_i à e_j, on détermine A_{ij} l'ensemble de intervalles d'arrivée possibles en e_j depuis e_i :

$$A_{ij} = (w_{temps}(r_{ij}) \oplus D_{ij}) \cap \Phi(e_j)$$

- On note Φ_{ij} l'ensemble des intervalles des arrivées possibles en e_j compatibles avec les contraintes d'accessibilité de e_j :

$$\Phi_{ij} = \{\phi \in \Phi(j) \ / \ A_{ij} \cap \phi \neq \emptyset\}$$

L'ensemble Φ_{ij} définit les fenêtres temporelles valides en e_j en partant de e_i à un instant donné. Ainsi les successeurs du nœud n en e_j sont définis par l'ensemble :

$$\text{SCS}(n) = \{n_s = (e_j, \phi_s), \exists r_{i,j} \in Ro, \phi_s \in \Phi_{ij}\}$$

Un nœud n_s, successeur de n est donc atteint à l'instant $a(n_s) = g_{temps}(n_s)$. Soit $a_{ij} \in A_{ij}$ l'intervalle d'arrivée possible associé à n_s, il faut partir du nœud n à l'instant $d^-(n_s) = inf(a_{ij}) - w_{temps}(r)$ où r est la route de n à n_s. On peut déterminer une première fois l'instant de départ au plus tard $d^+(n_s) = sup(a_{ij}) - w_{temps}(r)$. Cette valeur est corrigée lorsque l'itinéraire complet est trouvé.

5.3.4 Satisfaction des contraintes

Les contraintes $C_i[T_k]$ pour la tâche T_k et le paramètre i sont satisfaites pour le nœud n si :

$$\tilde{g}_i(n) \leq C_i[T_k]$$

Pour chaque paramètre j correspondant à une fonction de coût monotone croissante la violation de la contrainte associée peut être détectée plus précocement si nous considérons le coût évalué $f_j(n)$. Si $f_j(n) > C_j[T_k]$ alors pour tout chemin $P(n, n_*)$ il existe un nœud m tel que $\tilde{g}_j(m) > C_j[T_k]$. Cette propriété permet de réduire l'espace de recherche.

5.3.5 Remise en cause d'un plan

Lorsqu'une tâche T_k ne peut être planifiée et qu'au moins un des dépassements de contrainte concerne un coût réinitialisable, alors le plan d'une ou plusieurs tâches précédentes peut être remis en cause. Il n'est remis en cause qu'à cette condition. Ce n'est pas réellement une replanification des tâches précédentes puisque la liste des nœuds ouverts lors de leur planification précédente est conservée. D'autre part, on ajoute à cette liste les nœuds de réinitialisation non encore ouverts.

La remise en cause du plan des tâches précédentes suppose qu'aucune action de réinitialisation n'a pu être insérée dans le plan de la tâche T_k. Il faut donc forcer

```
procédure Verifier(n, T, ρ)
début
si g̃(n) ≪ C[T] et loc(n) ∈ loc(M) alors
    Retourner(ρ);
fin si
si loc(n) ∉ loc(M) et f_o(n) < f_o(n_echec) alors
    n_echec := n;
    p_echec :=localisation;
fin si
pour i ∈ PARAM_1 faire
    si f_i(n) ≥ C_i[T] et f_o(n) < f_o(n_echec) alors
        n_echec := n;
        p_echec := i;
    fin si
fin pour
pour i ∈ PARAM_2 faire
    si g̃_i(n) ≥ C_i[T] alors
        OUVERT := OUVERT ∪ REINIT_i/π(T);
        ρ := π(T);
        si f_o(n) < f_o(n_echec) alors
            n_echec := n;
            p_echec := i;
        fin si
    fin si
fin pour
Fermer(n);
Retourner(ρ);
fin
```

FIG. 5.3 - *Pseudo-code de la procédure : Verifier*

l'insertion de ce type d'actions dans le plan des tâches précédentes. Il suffit pour cela de redéfinir les contraintes associés aux paramètres réinitialisables qui sont la cause de l'échec de planification.

Pour redéfinir les contraintes (cf la procédure *Redefinir*), nous nous basons sur le critère suivant : la replanification de T_{k-1} doit permettre de progresser lors de la replanification consécutive de T_k. C'est-à-dire que les coûts planifiés pour T_{k-1} doivent permettre d'atteindre ensuite pour T_k au moins un état non atteignable précédemment (et auquel correspondent un ou plusieurs nœuds échec). Les nouvelles contraintes de T_{k-1} seront nécessairement plus strictes.

Leur vérification ne peut s'effectuer de la même façon que lors d'une planification normale. Pour les coûts réinitialisables, les nouvelles contraintes ne s'appliquent qu'aux nœuds d'état final de T_{k-1}. En effet, l'état où l'action de réinitialisation est possible peut être arbitrairement voisin (en termes de coûts) de l'état final de T_{k-1} et il ne faut pas que les nouvelles contraintes sur T_{k-1} empêchent d'atteindre cet état sur lequel une action de réinitialisation peut permettre de satisfaire les contraintes de la tâche suivante T_k.

Pour forcer l'insertion d'une action de réinitialisation pour le paramètre i, nous introduisons un nouveau type de contrainte, appelée *contrainte finale* notée CF définie comme suit :

Contrainte finale : On prend pour chaque paramètre réinitialisable i la valeur minimale d'échec, notée $Emin$, déterminée sur l'ensemble des nœuds échec de T_k. Ceci permet de déduire pour chaque paramètre le coût minimal K_i pour aller du nœud de départ de T_k (noté n_0^k) à un nœud échec n_e. La contrainte finale $CF_i[T_{k-1}]$ est la contrainte $C_i[T_k]$ réduite du coût K_i.

Pour $i \in PARAM_2$ et $n_e, n'_e \in ECHEC/\pi(T_k)$:

$$
\begin{aligned}
Emin_i &= g_i(n_e), \tilde{g}_i(n_e) > C_i[T_k] \text{ et } \nexists n'_e \,/\, \tilde{g}_i(n'_e) < \tilde{g}(n_e) \\
K_i &= Emin_i - g_i(n_0^k) \\
CF_i[T_{k-1}] &= C_i[T_{k-1}] - K_i
\end{aligned}
$$

D'autre part, les nœuds échec sont susceptibles d'avoir des coûts non comparables. Imposer que chaque coût réinitialisable respecte la contrainte finale correspondante est une exigence trop forte et peut conduire à ignorer des solutions. Sur l'état final, un nœud doit donc respecter au moins une des contraintes associées aux coûts réinitialisables mais pas nécessairement toutes. Cette condition est vérifiée par le prédicat EtatBut.

Il est d'autre part possible de réduire l'espace d'états pour la replanification de T_{k-1} car on connaît les coûts minimaux des paramètres non réinitialisables sur les nœuds frontières calculés pour T_k. On peut donc réduire la valeur des contraintes associées pour T_{k-1}, et ces nouvelles contraintes C' doivent être vérifiées pour tous les nœuds de l'espace d'état construit lors de la planification de T_{k-1}.

Pour $i \in PARAM_1$,
$K_i = (\min_{m \in ECHEC/\pi(T_k)} g_i(m)) - g_i(n_0^k)$
$C'_i[T_{k-1}] = min(C_i[T_{k-1}], C_i[T_k] - K_i)$

S'il n'est pas possible d'insérer des actions de réinitialisation dans T_{k-1} sans dépasser les contraintes des coûts non réinitialisables alors le plan de la tâche T_{k-2} est remis en cause. Comme l'indice du plan courant ($\pi(T_{k-1})$) n'est pas l'indice de celui qui a échoué (indice égal à ρ), les contraintes sont redéfinies en déduisant pour chaque paramètre le coût pour aller de γ_0^{k-1} à $\gamma_\star^{k-1}$, ce chemin ayant été déterminé antérieurement.

```
procédure Redefinir(T, K)
début
pour i ∈ PARAM_2 faire
    si echec[i] alors
        pour n ∈ REINIT[i]/π(T) faire
            Ouvrir(n);
        fin pour
    fin si
fin pour
pour i ∈ PARAM_1 faire
    C_i[T] := min(C_i[T], K_i);          /* Nouvelles contraintes de T */
fin pour
pour i ∈ PARAM_2 faire
    CF_i[T] := K_i;                      /* Contraintes finales de T */
fin pour
fin
```

FIG. 5.4 - *Pseudo-code de la procédure : Redefinir*

5.3.6 Conditions pour ouvrir ou éliminer un nœud

Un nœud valide n, successeur du nœud courant, est inséré dans la liste OUVERT s'il satisfait les conditions, données dans l'algorithme *SelectEfficace*, liées à l'existence d'autres nœuds des listes OUVERT et FERMÉ. Un nouveau nœud n est inséré dans OUVERT si

$$\forall m \in (\text{OUVERT} \cup \text{FERMÉ})/\pi(n) :$$
$$e(m) \neq e(n) \vee \phi(m) \neq \phi(n)$$

Dans le cas contraire, plusieurs nœuds peuvent décrire le même état et l'on doit comparer leurs coûts respectifs.

L'algorithme *SelectEfficace* n'autorise l'ouverture d'un nouveau nœud que s'il appartient à un chemin efficace comparé aux autres nœuds décrivant le même état.

procédure **ContraintesFinalesValides**(K, ρ, T)
début
si $\pi(T) = \rho$ **alors**
  pour $i \in$ PARAMETRES **faire**
    $K_i := C_i[T] - [(\min_{m \in E_e/\rho} g_i(m)) - g_i(\gamma_0(T))]$;
    si $K_i \leq 0$ **alors** *Retourner*(Faux);
  fin pour
sinon
  pour $i \in$ PARAMETRES **faire**
    $K_i := K_i - [(\min_{\gamma(m)=\gamma_f(T)} g_i(m)) - g_i(\gamma_0(T))]$;
    si $K_i \leq 0$ **alors** *Retourner*(Faux);
  fin pour
fin si
Retourner(Vrai);
fin

FIG. 5.5 - *Pseudo-code de la procédure : ContraintesFinalesValides*

procédure **SelectEfficace(G,n,T)**
début
pour $m \in (\text{OUVERT} \cup \text{FERME})/\pi(n)$ tel que $e(m) = e(n) \wedge \phi(m) = \phi(n)$ **faire**
  si $\forall i \in$ PARAMETRES, $\tilde{g}_i(n) \geq \tilde{g}_i(m)$ **alors**
    Eliminer(n);
    Fin;
  fin si
  si $\forall i \in$ PARAMETRES, $\tilde{g}_i(n) \leq \tilde{g}_i(m)$ **alors**
    si $m \in$ OUVERT **alors**
      Eliminer(m);
    fin si
  fin si
fin pour
Ouvrir(n);
fin

FIG. 5.6 - *Pseudo-code de la procédure : SelectEfficace*

5.4 Évaluation de l'algorithme de recherche d'itinéraire

Les caractéristiques de notre algorithme doivent être comparées avec celles de l'algorithme A*. L'algorithme A* construit un arbre recouvrant de l'espace d'état. Cette propriété n'est plus vraie dans le cas de notre algorithme. Comme nous l'avons vu précédemment, afin de trouver le meilleur chemin qui satisfait toutes les contraintes, nous devons évaluer plusieurs chemins efficaces et un état peut appartenir à différents chemins qui sont représentés par des nœuds différents (voir figures 5.7 et 5.8). Mais comme la recherche est basée sur un unique critère d'optimisation, les propriétés de complétude et de complexité sont similaires à celles de A* [Pearl, 1981].

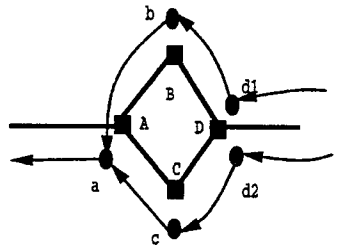


FIG. 5.7 - *Plusieurs nœuds peuvent être conservés pour un même état si les vecteurs de coût (ici d1, d2) ne sont pas comparables, (états: A,B,C,D, nœuds: a,b,c,d1, d2).*

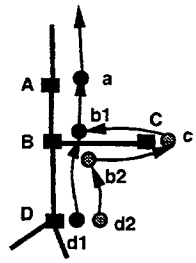


FIG. 5.8 - *Un détour peut être planifié pour effectuer une action de réinitialisation en C, (états: A, B, C, D, nœuds: a, b1, b2, c, d1, d2).*

5.4.1 Terminaison, complétude et optimalité

Pour la planification d'une tâche: Sachant que A* termine toujours sur les graphes finis, nous examinons le cas des nœuds multiples pour un même état.

Le coût d'une action (ou d'un nœud) *aller* est défini par le coût de la route associée. Chaque composante de ce vecteur de coûts est calculée en fonction de la distance parcourue sur cette route. Ces fonctions de coûts w_i sont des fonctions croissantes qui doivent vérifier l'inégalité triangulaire afin d'être consistantes avec la notion de coût lié à une distance parcourue: soient $x, y, z \in \mathbb{R}$ alors pour tout paramètre i, si $x < y + z$

les fonctions w_i vérifient $w_i(x) < w_i(y) + w_i(z)$. Ces fonctions de coûts peuvent être n'importe quelle norme ou fonction linéaire.

Nous évaluons le nombre maximum de nœuds par état en l'absence d'action de réinitialisation.

Définition uniforme des fonctions de coûts - Si pour chaque paramètre, la même fonction de coût est utilisée pour chaque route du réseau, alors il passe au plus un chemin par état de l'espace d'état (comme pour A*). C'est une conséquence de l'inégalité triangulaire sur les fonctions de coûts.

Définition non-uniforme des fonctions de coûts - Soit deux nœuds n et m, une route r, reliant $e(n)$ à $e(m)$. On note g(m) le coût de l'unique chemin efficace de n_0 à m passant par n, et $g'(m)$ le coût de l'unique chemin efficace de n_0 à m évitant la route r. On suppose que les deux chemins permettent d'arriver en $e(m)$ dans la même fenêtre temporelle.

Si certaines fonctions de coût w_i pour cette route r sont définies différemment de toutes les autres routes du réseau, on peut avoir certaines composantes i, j telles que $g_i(m) > g'_i(m)$ et $g_j(m) < g'_j(m)$. Comme les vecteurs de coûts ne sont pas comparables, l'état correspondant à l'emplacement $e(m)$ doit être décrit par deux nœuds correspondant l'un au chemin passant par r et l'autre évitant de passer par r.

Par récurrence : si k routes sont définies avec des fonctions de coût différentes des autres routes du réseau, dans le pire des cas, certains états doivent être décrits avec 2^k nœuds. Supposons que 2^k chemins efficaces arrivent en $e(m)$ en passant par r et que 2^k chemins efficaces arrivent en $e(m)$ évitant cette route. Tous les vecteurs de coûts peuvent ne pas être comparables deux à deux, ainsi l'état en $e(m)$ doit être décrit par 2^{k+1} nœuds.

Réinitialisation des coûts - Le cas des actions de réinitialisation peut être traité comme un arc du graphe de recherche sur lequel certaines fonctions d'évaluation ont été redéfinies. Ainsi un réseau avec des fonctions de coûts uniformes et k' emplacements de réinitialisation peut conduire à un graphe de recherche ayant au plus $2^{k'}$ nœuds par état.

L'algorithme est complet pour la planification de chaque tâche de par sa stratégie de recherche du meilleur d'abord, de la même façon que l'algorithme A*.

Pour chaque tâche, l'itinéraire produit est optimal parce que d'une part les fonctions de coûts sont des fonctions croissantes avec la distance vérifiant l'inégalité triangulaire et d'autre part le critère de sélection du meilleur nœud est le même que pour A*.

Avec replanification : L'algorithme n'est pas complet si l'on considère la mission dans son ensemble car nous restreignons l'utilisation du mécanisme de cheminement arrière [backtracking] seulement pour les cas d'échecs concernant les coûts réinitialisables. La redéfinition des contraintes assure que tous les nœuds ouverts compatibles

avec les nouvelles contraintes vont être évalués. Lors d'une replanification les contraintes sont plus strictes, le nombre de chemins possibles diminue. Le mécanisme de cheminement arrière qui remet en cause le plan de tâches précédentes s'arrête lorsque les liste OUVERT et FERMÉ(i) sont vides ou que la première tâche ne peut pas être (re)planifiée.

5.4.2 Complexité

Planification d'une tâche : Dans le pire des cas, la complexité de A* est $O(\eta^\rho)$; où η est le nombre de nœuds et ρ le nombre d'arcs du chemin solution. Pour un espace d'état et un critère d'optimisation donné, posons que A* évalue M nœuds, alors comme nous l'avons montré précédemment notre algorithme évalue au plus $(2^k) \times M$ nœuds, où k est le nombre de n-uples de fonctions de coûts distincts.

Cas des replanification : Quand l'algorithme replanifie certaines tâches, les nœuds évalués lors de la première planification sont conservés. Ainsi l'algorithme évalue de nouveaux nœuds avec d'une part les contraintes à vérifier en chaque nœud et d'autre part les contraintes appliquées au nœuds but. Néanmoins, compte tenu que les contraintes finales sur les coûts réinitialisables sont disjonctives, dans le cas où tous les nœuds échec ont q coûts réinitialisables dépassant leurs contraintes respectives, il faut nécessairement q remises en cause successives des plans pour une même tâche. Elles permettent d'insérer dans le plan final les q actions de réinitialisation (de type distinct) nécessaires pour prolonger l'itinéraire au delà des nœuds échec qui ont déclenché le processus de remise en cause des plans.

5.4.3 Résultats expérimentaux

Pour évaluer la complexité pratique de notre algorithme de recherche d'itinéraire, nous allons nous placer dans le cas où l'environnement à une topologie de grille régulière plutôt qu'un graphe aléatoire. Ce choix est motivé par plusieurs raisons : (1) la plupart des auteurs évaluent ce type d'algorithme sur des grilles régulières, ce qui permet de reproduire et comparer aisément les résultats ; (2) on dispose d'une évaluation analytique précise de la complexité de l'algorithme A* pour les grilles régulières [Pearl, 1981] ; (3) la topologie de l'environnement d'évolution envisagé - environnement structuré intérieur - est souvent proche de celle d'une grille régulière.

Nous avons évalué l'algorithme sur des grilles de taille identique mais contenant une quantité croissante d'arcs dont les coûts élémentaires sont choisis de manière aléatoire dans un certain intervalle. On observe que plus la densité d'arcs de coût aléatoire augmente, plus le nombre de nœuds augmente tout en restant bien inférieur à la borne théorique.

La table 5.9 montre l'accroissement du nombre de nœuds en fonction du nombre d'arcs de coûts aléatoires. La figure 5.10 indique le nombre de nœuds créés pour trouver le chemin optimal entre le point de départ $\boxed{1}$ et l'arrivée en $\boxed{2}$ dans une grille 20x20 comportant 20% d'arcs de coûts aléatoires.

% arcs/total	Nbr. d'arcs	Nbr. de nœuds
0	0	256
2,5	18	446
5	35	723
7	50	781
10	67	835
20	127	977
30	173	1155
50	285	1286

FIG. 5.9 - *Accroissement du nombre de nœuds en fonction du pourcentage d'arcs de coûts aléatoires.*

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	4	5	5	4	4	3	3	3	2	0	0	0	0	0
0	0	0	0	1	1	4	5	5	5	6	6	7	8	8	2	0	0	0	0
0	0	0	0	3	1	4	5	5	5	6	6	7	7	7	7	1	0	0	0
0	0	0	3	3	1	4	5	5	5	6	6	7	7	7	7	2	0	0	0
0	0	0	3	2	1	3	5	5	5	6	6	7	7	7	7	3	0	0	0
0	0	2	3	2	1	3	4	4	5	6	6	6	6	6	6	4	0	0	0
0	0	3	3	2	1	2	3	3	3	3	3	5	6	6	6	4	0	0	0
0	0	3	3	2	1	2	2	2	2	3	3	4	6	6	6	5	0	0	0
0	2	2	2	2	1	2	2	2	2	3	3	4	4	5	5	5	0	0	0
0	2	2	2	2	1	1	1	1	2	3	3	3	3	3	3	3	0	0	0
0	2	2	1	1	1	1	1	1	2	2	2	2	2	2	3	3	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	2	1	1	1	2	2	2	2	2	2	2	2	3	1	0	0	0	0
0	0	1	1	1	1	2	2	2	2	2	3	3	2	0	0	0	0	0	0
0	0	0	1	1	1	2	2	2	3	2	2	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	2	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIG. 5.10 - *Nombre de nœuds créés pour chaque intersection sur la grille 20x20.*

5.5 Construction du plan

Le plan complet associé à la mission spécifiée doit comporter les actions de navigation et les actions parallèles que sont les actions de surveillance.

Une fois que le dernier état but de la mission est atteint, l'itinéraire est déterminé en remontant la chaîne des pointeurs du nœud final au nœud initial. Lors de cette phase, il faut déterminer en chaque nœud les marges disponibles pour chaque paramètre. Ces marges seront ensuite associées aux actions correspondant à chaque nœud.

Les tâches parallèles sont décomposées en actions en fonction de leurs contraintes et de l'itinéraire planifié. Un intervalle d'exécution est calculé pour chaque action parallèle. L'ordonnancement des actions de navigation et des actions parallèles est réalisé en construisant un graphe d'exécution.

5.5.1 Calcul des marges

Les coûts planifiés et les marges de chaque tâche sont déterminés sur les nœuds buts correspondant à ces tâches, de même pour les actions. Il faut distinguer trois cas : (1) les marges de temps, (2) les marges des autres coûts non réinitialisables et (3) les marges des coûts réinitialisables.

Pour déterminer les marges de temps en chaque nœud, il faut d'abord mettre à jour la valeur de départ au plus tard d^+.

Pour le nœud final $n_\star$ d'une tâche T :

$$d^+(n_\star) = min(C_{temps}[T] - w_{temps}(r), d^+(n_\star))$$

Pour les nœuds précédents, jusqu'au nœud initial :

$$d^+(p(n)) = min(d^+(p(n)), d^+(n) - w_{temps}(r_{p(p(n)),p(n)}))$$

où $w_{temps}(r_{p(p(n)),p(n)})$ est le temps de trajet du nœud $p(p(n))$ à $p(n)$.

La marge temporelle est donnée par :

$$M_{temps}(n) = d^+(n) - a(p(n))$$

Pour les autres coûts non réinitialisables i, la marge associée est :

$$M_i(n) = C_i[T] - g_i(n)$$

Pour les coûts réinitialisables ($i \in PARAM_2$) trois cas se présentent :

1. Si n est le nœud final: $M_i(n) = C_i[M]_p - \tilde{g}_i(n)$;
2. Si n n'est pas le nœud initial et si $type(n) = Reinit_i$ alors pour le nœud précédent : $M_i(p(n)) = C_i[T] - \tilde{g}_i(p(n))$;

3. Pour un nœud n autre que le nœud initial et de type *Aller*, *Rester* ou $Reinit_j$ avec $i \neq j$ alors la marge du nœud précédent est : $M_i(p(n)) = M_i(n)$.

Les nœuds étant typés, la construction du plan pour les actions de navigation est immédiate.

5.5.2 Détermination des actions parallèles

La planification des tâches de surveillance de communication ou de manipulation diffère de celles de navigation en ce sens que lorsqu'on spécifie par exemple :

```
Surveiller(feu, alarme) SalleRobotique [12/30/00 14/00/00] Absolu;
```

Cette tâche n'implique pas un déplacement dans le lieu "SalleRobotique" ; mais si l'itinéraire planifié amène le robot dans la "SalleRobotique", éventuellement plusieurs fois au cours de la mission, alors l'événement spécifié sera pris en compte à chaque fois, mais uniquement pendant l'intervalle de temps spécifié.

- Une tâche de surveillance avec une contrainte de localisation sans contrainte horaire implique qu'à chaque fois que dans la mission le robot s'arrête à un emplacement inclus dans la localisation de la tâche, la surveillance est activée.

- Une tâche de surveillance sans localisation s'exécute indépendemment de l'itinéraire, pendant l'intervalle de temps spécifié.

Pour chaque tâche parallèle, une action est créée à chaque fois que l'itinéraire satisfait les contraintes de la tâche. L'intervalle d'exécution est calculé pour chaque action. Dans la plupart des cas il coïncide avec l'intervalle de temps de l'action "rester" ayant la même localisation.

5.5.3 Construction du graphe d'exécution

Le graphe d'exécution est constitué des intervalles d'exécution de chaque élément du plan (tâches et actions) et des relations temporelles qui relient certains de ces intervalles. Ce graphe est utilisé pendant la phase d'exécution pour vérifier les contraintes de précédence entre les tâches et entre les actions et pour permettre de synchroniser les actions de navigation et les actions parallèles. Le graphe d'exécution est une hiérarchie de sous graphes à deux niveaux : tâches et actions.

La définition des relations entre les intervalles repose sur le système de Allen [Allen, 1982]. Il structure les relations entre des événements à l'aide d'un graphe temporel complet dont chaque nœud dénote un intervalle. Il définit treize relations primitives décrivant les positions relatives de deux intervalles (cf fig 5.11). Ces relations, mutuellement exclusives, permettent de relier tout couple d'intervalles. Si deux intervalles n'ont pas de bornes connues, on peut leur associer une liste de relations primitives possibles, au contraire si les intervalles sont datés alors il ne sont reliés que par une unique relation primitive.

Les arcs du graphe sont donc des relations disjonctives décrites par un ensemble de primitives. Comme l'on considère ces relations comme des ensembles on peut leur appliquer les opérations d'union, d'intersection qui expriment respectivement la disjonction et la conjonction.

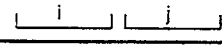
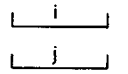
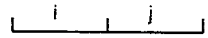
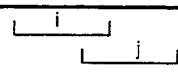
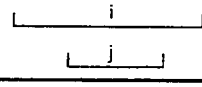
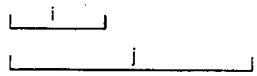
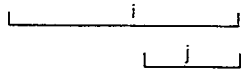
Relation	Symbole	Symbole de la relation inverse	Exemple graphique
i before j	<	>	
i equals j	=		
i meets j	m	mi	
i overlaps j	o	oi	
i contains j	c	ci	
i starts j	s	si	
i finished by j	f	fi	

FIG. 5.11 - *Les 13 relations temporelles possibles entre deux intervalles i et j.*

Un graphe d'intervalle est une paire $< \mathcal{I}, \mathcal{R} >$ telle que $\mathcal{I}$ est un ensemble d'intervalles et $\mathcal{R}$ un ensemble de relations sur les intervalles. Tous les intervalles d'un tel graphe sont connectés. Soit C une disjonction de primitives. La relation temporelle entre deux intervalles I_1 et I_2 se note $R =< I_1, C, I_2 >$.

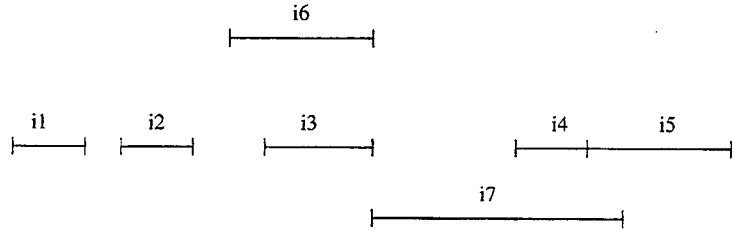


FIG. 5.12 - *Une mission décrite par les intervalles associés aux actions*

La notion de graphe de séquence a été proposée par Dorn [Dorn, 1992] pour réduire la taille du graphe d'intervalles. La construction de ces graphes est basée sur l'idée que

la connaissance de la nature de la relation temporelle existant entre deux intervalles nous permet de réduire la taille du graphe. Afin de ne pas calculer toutes les relations et générer un graphe complet, nous pouvons à l'aide de règles simples restreindre ce graphe à un graphe de séquence. On différenciera la relation présente dans le graphe $R \in \mathcal{R}$ de la relation calculable mais non représentée $R(I_1, I_2)$.

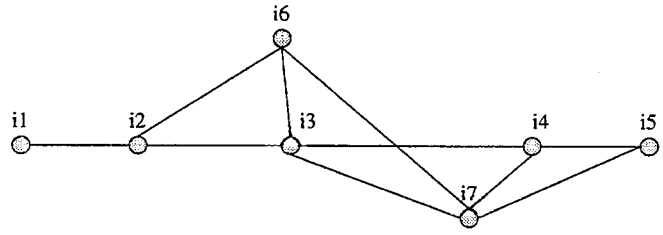


FIG. 5.13 - *Graphe de séquence*

Un graphe de séquence est un graphe d'intervalles incomplet, les propriétés d'une chaîne de séquence sont utilisées pour réduire le nombre d'arcs dans le graphe.

Une contrainte temporelle C est une contrainte de séquence si :

$$sequence(C) \leftrightarrow C \in \{<, m\} \wedge sequence(I_1, I_2) \leftrightarrow R =< I_1, C, I_2 > \wedge sequence(R)$$

Une chaîne de séquence est un sous graphe où toutes les contraintes sont des contraintes de séquence.

$$sequenceChaine(< \mathcal{I}, \mathcal{R} >) \leftrightarrow \forall < I_1, C, I_2 > \in \mathcal{R} \wedge sequence(C)$$

Un graphe de séquence vérifie la propriété suivante :

$$sequenceGraphe(< \mathcal{I}, \mathcal{R} >) \rightarrow \forall I_1, I_2 \in \mathcal{I} \wedge < I_1, C_1, I_2 > \in \mathcal{R} \; \nexists I_3 \in \mathcal{I}[sequence(I_1, I_3) \wedge sequence(I_3, I_2)]$$

D'après la définition des graphes de séquence, nous pouvons déduire les règles de construction d'un tel type de graphe pour les intervalles d'exécution précédemment calculés.

- Si deux intervalles I_1, I_2 sont non connectés et $r_1 = R(I_1, I_2)$ la relation calculable entre ces deux intervalles, alors ajouter la relation $R_1 =< I_1, r_1, I_2 >$.
- Soit la relation $R_1 =< I_1, C_1, I_2 >$ et un intervalle I_3 nouvellement calculé, deux cas se présentent :

  (1) $r_1 = R(I_1, I_3)$ et $r_2 = R(I_2, I_3)$.
  Si $C_1 \in \{<, m\}$ et $r_1, r_2 \in \{<, m\}$, alors ajouter $R_2 =< I_2, r_2, I_3 >$.

  (2) $r_2 = R(I_3, I_2)$
  Si $C_1 =<$ et $r_1, r_2 \in \{<, m\}$ alors
  ajouter $R_2 =< I_1, r_1, I_3 >$,

ajouter $R_3 =< I_3, r_2, I_2 >$,
supprimer R_1.

Le graphe d'exécution ne contient que des intervalles instanciés, la contrainte temporelle qui les relie se réduit donc à un singleton. Les règles de construction permettent d'obtenir un graphe cohérent de taille réduite. Ces règles sont applicables aussi bien pendant la phase de planification que pendant la phase d'exécution lorsqu'il faut mettre à jour le graphe.

5.6 Conclusion

Étant donnés une mission et un environnement, il n'est pas possible, sans un mécanisme de remise en cause du plan ou sans évaluer tous les chemins possibles, de s'assurer qu'il n'existe pas d'itinéraire satisfaisant toutes les contraintes et qui soit, compte tenu de ces contraintes, le meilleur possible pour chaque tâche. Cela vient de ce que chaque tâche a son propre critère d'optimisation. Si elles avaient toutes le même critère d'optimisation, il n'y aurait qu'une seule liste des nœuds ouverts pour toute la mission et l'échec de planification pour une tâche se résoudrait en explorant les successeurs du meilleur nœud ouvert de la tâche précédente. Comme les fonctions d'évaluation sont distinctes pour chaque paramètre, il n'y a pas de stratégie pour chercher de manière optimale un chemin qui satisfait une contrainte qui ne porte pas sur le paramètre critère d'optimisation.

Pour ne pas avoir à énumérer tous les chemins (sous optimaux) les cas replanification sont restreints aux situations dans lesquelles les contraintes associées aux coûts réinitialisables ne sont pas satisfaites. L'algorithme que nous proposons permet donc de chercher un chemin optimal pour chaque tâche qui tienne compte à la fois des contraintes sur la mission, des contraintes temporelles sur l'environnement et des possibilités de réinitialisation. Les performances de cet algorithme dépendent de la répartition (quantité, localisation) des points de réinitialisation, des contraintes temporelles sur l'environnement et de l'uniformité des fonctions de coûts.

Nous n'avons pas cherché, bien qu'il soit trivial de le faire, à combiner les différentes fonctions de coût en une unique fonction combinaison linéaire de celles ci. Ce choix a été guidé par une raison pragmatique : l'opérateur peut difficilement prévoir l'itinéraire résultant du choix de coefficients non nuls pour cette combinaison linéaire. Par contre il a l'intuition du chemin le plus rapide ou du chemin le plus court car il connaît l'environnement et il a fixé les vitesses de consigne pour chaque route. La possibilité de combiner les fonctions de coûts en une seule somme pondérée ne s'avère nécessaire que si beaucoup d'itinéraires ont le même coût pour le paramètre prépondérant.

Chapitre 6

Contrôle d'exécution

Contrôler l'exécution d'une mission d'un robot mobile c'est lui faire accomplir des actions selon le plan prévu, diagnostiquer d'éventuelles défaillances dans leur exécution et y remédier. Nous avons vu dans le chapitre concernant l'architecture que ce contrôle s'exerce à tous les niveaux du système. Il consiste en une hiérarchie de boucles d'asservissement fermées : commande - observations - consigne.

Le Superviseur se situe au sommet de cette hiérarchie. Les traitements qu'il effectue sont essentiellement symboliques. Le contrôle qu'il exerce sur l'exécution de la mission se base sur l'évaluation de l'état du système, défini à partir des événements discrets correspondant aux changements d'états des sous-systèmes. Mais l'observation par le Superviseur du fonctionnement des sous-systèmes qu'il contrôle ne se limite pas aux événements discrets. Ayant à satisfaire des contraintes opérationnelles, le Superviseur doit également observer l'évolution des paramètres correspondant à ces contraintes.

Le Superviseur construit un plan par décomposition hiérarchique d'une mission en tâches et actions. Les modifications à apporter au plan sont appliquées à un niveau déterminé de cette hiérarchie en fonction de l'ensemble des observations (coûts observés des paramètres contraints et changements d'états). Ce niveau de traitement est sélectionné en fonction de l'évaluation des conséquences possibles de ces observations sur la réalisation de l'action courante, de la tâche ou de la mission.

Nous présenterons l'aspect événementiel du contrôle d'exécution dans la section 6.1, nous aborderons dans la section 6.2 le problème de la gestion des contraintes opérationnelles. La dernière section expose comment sont intégrés les différents mécanismes mis en œuvre.

6.1 Système à Événements Discrets

Le concept de *Système à Événements Discrets* à été introduit dans les dix dernières années afin de faciliter la conception et l'analyse de systèmes complexes. Un *Système à Événement Discrets* est un système à états discrets, dirigé par les événements[1]. L'évolution de son état dépend entièrement de l'apparition d'événements discrets au cours

1. Pour avoir un aperçu plus complet de cette approche voir [Cassandras, 1993].

du temps. Les différents états du système sont définis par un ensemble de propriétés satisfaites pendant un certain temps et les transitions correspondent soit aux actions que le système peut accomplir soit aux événements externes qui peuvent survenir de manière asynchrone.

Ce formalisme a été employé en robotique mobile pour décrire la coordination et la résolution de conflits entre activités concurrentes pour l'accès à des ressources critiques, [Kosecka *et al.*, 1993; Causse et Christensen, 1994]. Nous nous intéresserons ici à son utilisation pour la spécification et la réalisation de l'interface entre le Superviseur et le module de navigation.

6.1.1 Définition des événements

Pour suivre l'exécution des actions de navigation et coordonner les commandes issues du Superviseur avec les changements d'états du module de navigation le Superviseur gère un automate d'états finis que l'on appelle *automate de coordination.* La figure 6.1 représente l'automate de coordination pour les actions de navigation. Les commandes considérées pour la navigation sont :

- init : pour l'initialisation du module de navigation ;
- aller : cette commande est paramétrée par la vitesse de consigne et le mode de navigation ;
- stop : cette commande est paramétrée pour un arrêt immédiat ou un arrêt au prochain emplacement.

Les événements en provenance du module de navigation sont les suivants :

- A - Attente : le robot est en fonctionnement et attend un ordre ;
- E - En cours : le robot exécute un mouvement ;
- T - Terminée : fin de l'action en cours ;
- B - Bloqué : le robot a détecté un obstacle.

6.1.2 Définition des transitions

L'automate contient trois types de transitions :

1. les transitions sur les événements issus du sous système contrôlé, notées : $\langle evt \rangle \uparrow$;
2. les transitions sur les commandes issues du superviseur, notées : $\langle cmd \rangle \downarrow$;
3. les transitions temporelles, notées ϵ.

Un état de l'automate possède au plus une transition temporelle. Celle-ci est déclenchée quand le système reste dans cet état pendant une durée supérieure à un seuil donné. Ceci permet de faire face de manière spécifique aux situations dans lesquelles un sous-système ne fournit pas un des événements attendus dans le délai imparti.

La transition de l'état *Mvt* à l'état *Acq* indique que le Superviseur peut émettre une nouvelle commande *aller* alors qu'une telle commande est déjà en cours. Ceci permet de modifier de manière asynchrone les paramètres de l'action en cours (consigne de vitesse, mode de navigation) sans arrêter le véhicule. Une transition a pour effet la mise à jour dans la base de données du Superviseur des données relatives à l'action, la tâche ou l'environnement.

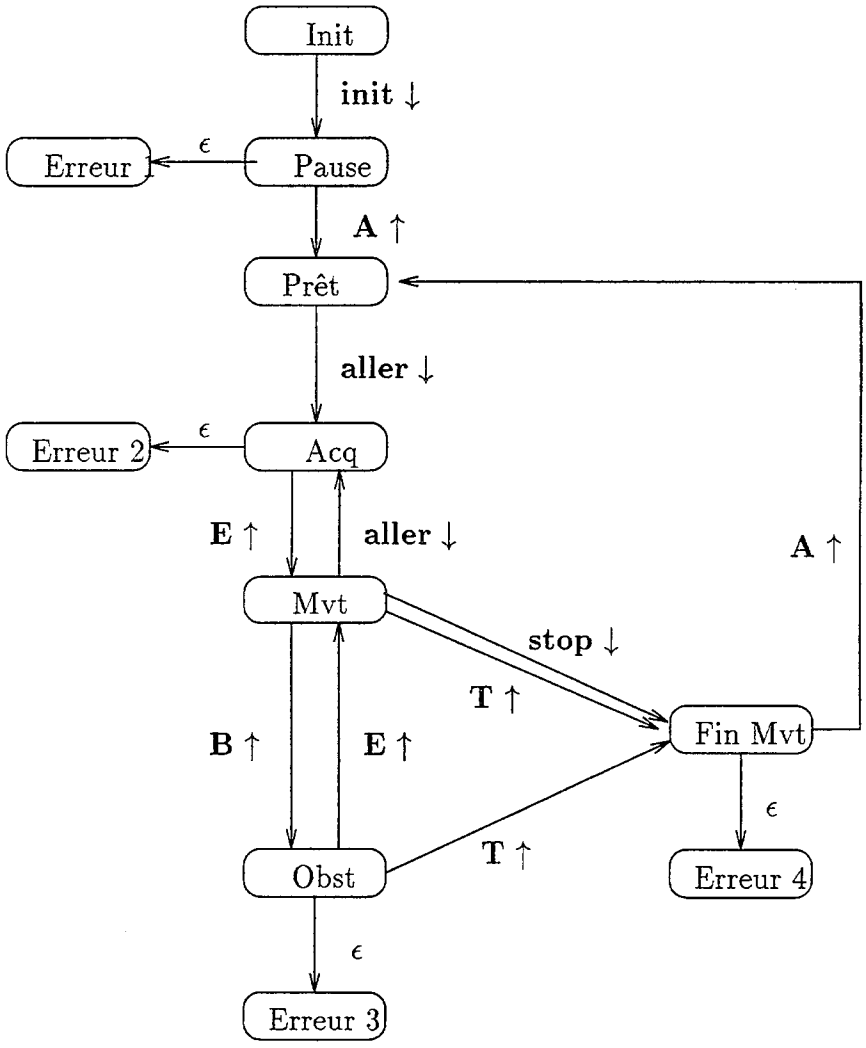


FIG. 6.1 - *Automate de coordination pour l'action "aller"*

Une approche similaire est suivie dans [Chatila *et al.*, 1992] où une telle distinction est faite entre événements internes et événements externes. Les changements d'état

ont également pour effet la mise à jour d'une base de données. Cette mise à jour est explicitement associée aux transitions de l'automate décrivant un type d'action donné.

Un des bénéfices de la coordination par automate d'état finis, outre sa simplicité, est qu'elle permet une très grande indépendance dans la réalisation des modules coordonnés par ce type d'interface. En effet, sans avoir à modifier l'automate, le système de navigation a pu passer d'une approche géométrique de la détermination et l'exécution de trajectoire [Bobet et Reignier, 1989] à une approche réactive [Reignier, 1993].

La gestion des contraintes opérationnelles s'effectue aussi bien lorsque le véhicule est en mouvement que lorsqu'il est à l'arrêt. Lorsque le véhicule se déplace, l'automate de coordination est dans l'état Mvt. Les transitions $Mvt \rightarrow Acq$ et $Mvt \rightarrow FinMvt$ peuvent être déclenchées sur la base de l'évaluation des paramètres contraints pour arrêter le véhicule ou modifier les paramètres de l'action en cours. La section suivante présente ce mécanisme d'évaluation.

6.2 Gestion des contraintes opérationnelles

L'environnement n'étant pas complètement prévisible, il est impossible d'assurer que les objectifs seront tous atteints. Il est notamment impossible d'assurer que toutes les contraintes seront satisfaites. Cependant ces contraintes n'ont pas nécessairement une importance égale du point de vue de l'opérateur. D'autre part, les paramètres ne sont pas indépendants et la modification du plan peut avoir sur certains d'entre eux des effets contraires (i.e. accélérer réduit la durée de la tâche mais augmente l'énergie consommée). Ces différentes raisons nous ont conduit à définir un ordre de priorité sur ces paramètres. L'opérateur dispose donc de deux moyens pour définir leur importance relative :

1. Lorsqu'il spécifie une tâche, l'opérateur peut privilégier un paramètre en le choisissant comme critère d'optimisation. Non seulement le coût prédit de ce paramètre ne doit pas dépasser la contrainte associée mais le coût réel doit dépasser le moins possible le coût prédit. Pour les autres paramètres, il importe seulement que leur coût ne dépasse pas la contrainte associée.

2. Les différents paramètres doivent être ordonnés selon une priorité croissante forçant ainsi le Superviseur à réduire l'erreur associée au paramètre de plus forte priorité, quitte à ignorer un temps les erreurs pour les paramètres de priorité plus faible. La priorité d'un paramètre p est notée $PR(p)$; $Ord(PARAM)$ est l'ensemble des paramètres ordonnés par priorité décroissante.

La différence entre la valeur de la contrainte et la valeur du coût prévu, pour chaque paramètre, détermine les marges associées à chaque tâche et à chaque action. Ces marges peuvent permettre de modifier le plan tout en satisfaisant les contraintes. Tout événement imprévu entraîne un sur-coût et le plan doit être modifié pour absorber ce sur-coût. Si les marges disponibles sont trop faibles, il est presque certain que le robot ne pourra pas effectuer toutes les tâches de la mission sans violer certaines contraintes.

Le plan sera donc modifié en fonction de l'erreur observée pour chaque paramètre ainsi qu'en fonction du choix d'optimisation et des contraintes. Par exemple, dans le cas où le robot perd du temps et de l'énergie à éviter des obstacles, la modification à apporter au plan dépend de la spécification de la mission donnée par l'opérateur. Si la tâche est optimisée en temps, le robot accélère pour se rapprocher au mieux du temps prédit, il ralentit au contraire si la contrainte d'énergie est très stricte et qu'elle est prioritaire par rapport à la contrainte de temps.

6.2.1 Hiérarchie des niveaux de modification du plan

Le calcul d'un plan complet étant coûteux, nous définissons trois niveaux de modification du plan :

1. *amendement*: seule l'action courante est redéfinie ;
2. *planification* (ou replanification) : la tâche courante est replanifiée, avec modification éventuelle du critère d'optimisation ;
3. *définition*: la mission est redéfinie, soit par le Superviseur, soit par l'opérateur.

La définition de ces niveaux de modification du plan est analogue à la structure proposée par Mesarovic (cf. chapitre 3) ; celle-ci organise un système de contrôle en trois niveaux :

1. le niveau *commande* définit les paramètres de commande à appliquer au processus contrôlé,
2. le niveau *adaptation* détermine la loi de commande à appliquer en fonction des observations fournies par le niveau commande,
3. le niveau *décision* détermine les objectifs à atteindre en fonction des observations fournies par le niveau adaptation.

Dans l'architecture proposée par Mesarovic, chaque module responsable d'un niveau de contrôle fonctionne parallèlement aux autres et dispose de son propre cycle de contrôle. Notre approche est différente car il est apparu inutile de paralléliser complètement ces traitements au sein du Superviseur. Nous identifions dans le cycle de contrôle les différents points de décision afin de répartir les traitements correspondant dans des modules indépendants et faire fonctionner ces modules en temps partagé.

6.2.2 Définition du cycle de contrôle

Les points de décision du cycle de contrôle et les traitements associés sont :

1. l'évaluation de l'état courant et sélection du niveau de modification du plan ;

2. la modification du plan selon le niveau sélectionné : amendement, replanification ou redéfinition ;

3. la validation du plan : la validation du plan consiste d'une part à déterminer si le plan calculé correspond bien aux contraintes définies par le niveau de sélection courant et d'autre part à mettre à jour le graphe d'exécution. Le choix de l'action à exécuter peut s'effectuer une fois que le graphe est mis à jour.

L'entrelacement des traitements correspondant aux différentes phases du cycle de contrôle sera présenté dans la section 6.6. Ce mécanisme permet d'évaluer l'état du système à chaque réception d'une nouvelle donnée en provenance des sous-systèmes et d'évaluer des plans alternatifs ou d'effectuer la mise à jour du graphe d'exécution pendant l'exécution d'une l'action.

6.3 Évaluation et sélection

La phase d'évaluation et de sélection doit permettre de décider, en fonction de l'état du robot, s'il y a lieu de modifier le plan. La sélection consiste à choisir un niveau de modification du plan. Ce choix repose sur la comparaison entre les coûts observés et les coûts prédits des paramètres sur lesquels des contraintes ont été spécifiées. Cette phase permet de déterminer le paramètre dont le coût observé justifie une modification du plan et à quel niveau hiérarchique celle-ci doit avoir lieu. La sélection S est définie par deux attributs : la priorité $S.p$ et le niveau de sélection $S.n$ du paramètre évalué.

L'estimation de l'erreur ne peut se faire de manière précise parce que l'on ne dispose pas d'un modèle précis de la cinématique du véhicule, et encore moins de sa dynamique. Le modèle trapezoïdal de la consigne de vitesse ne permet de prendre en compte que des trajectoires parfaites, rectilignes. Pour prendre en compte le cas où cette trajectoire est différente nous avons basé notre prévision sur des données statistiques d'exécution des actions.

6.3.1 Définition des observations

On définit l'observation O_p du paramètre p pour l'action A de la manière suivante : soit Y_p la donnée brute en provenance des capteurs, K la commande en cours et z la distance au but , alors $O_p[A](K) = Y_p + W_p(K, z)$ où W_p est la prédiction du coût pour le paramètre p pour parcourir la distance z en appliquant la commande K.

Exemple pour le paramètre temps :

L'espace de commande est $\mathcal{K} = \{V_{min}, V_{nom}, V_{max}\}$, on observe la distance restant à parcourir z et v la vitesse du véhicule. Le modèle en trapèze de la consigne de vitesse nous donne, pour une constante d'accélération α et une commande $K \in \mathcal{K}$, le temps nécessaire pour atteindre le but W_{temps}.

$$\text{Si } \frac{K^2}{2\alpha} < z \text{ alors } W_{temps} = \frac{z}{v} + \frac{v}{2\alpha} \text{ sinon } W_{temps} = \frac{\sqrt{z}}{2\alpha}$$

L'instant d'arrivée est donc $O_t[A](K) = t_0 + W_{temps}$ où t_0 est l'instant d'observation.

On ne possède pas toujours, pour chaque paramètre p, un modèle satisfaisant permettant d'effectuer une prédiction en tout point de l'itinéraire. Dans ce cas, la valeur d'observation $O_p[A](K)$ prise est uniquement la valeur mesurée de ce paramètre à l'instant d'observation sans tenir compte du coût, jugé non prédictible, pour finir l'action. Cette valeur est alors excessivement optimiste et tend à retarder le moment où le plan doit être modifié.

Les étapes d'évaluation et de sélection peuvent avoir lieu alors qu'une replanification est déjà en cours. La comparaison du critère de sélection de chaque paramètre avec le critère de sélection qui a provoqué la replanification en cours, permet de décider s'il est nécessaire d'interrompre celle-ci pour en relancer une nouvelle. Ce mécanisme tire parti de l'entrelacement des phases d'évaluation/sélection, de modification du planification et de validation, et permet ainsi de réduire le nombre de replanifications.

6.3.2 Détermination du niveau de modification du plan

Quels que soient l'action A et le paramètre considérés, les coûts prédits $P[A]$, les marges $M[A]$ et les contraintes $C[A]$ déterminées lors de la planification sont reliés par les inégalités suivantes :

$$P[A] \leq P[A] + M[A] \leq C[A] \tag{6.1}$$

Les valeurs de $P[A]$, $M[A]$, et $C[A]$ permettent de définir les bornes d'intervalles auxquels on associe un niveau de modification du plan. Afin de simplifier les notations dans la suite de ce paragraphe, les valeurs relatives à l'action courante seront notées O,P,M et C.

1. niveau *normal*: le plan ne nécessite aucune modification.

   Si le paramètre évalué est le paramètre optimisé pour la tâche en cours, alors le coût observé doit vérifier :

   $$O_{po}(K) \leq P_{po} \tag{6.2}$$

   si ce n'est pas le paramètre optimisé, le coût observé doit vérifier :

   $$O_p(K) \leq P_p + M_p \tag{6.3}$$

2. niveau *amendement*: la modification porte seulement sur les paramètres de l'action de navigation **aller** en cours (ou de l'action de navigation suivante si celle en cours n'est pas un déplacement).

Pour le paramètre optimisé *po*, le plan est amendé s'il existe une commande K' permettant d'obtenir une observation corrigée $O_{po}(K')$ qui soit inférieure au coût prédit.

$$P_{po} < O_{po}(K) \leq P_{po} + M_{po} \text{ et } \exists K',\ O_{po}(K') \leq P_{po} \tag{6.4}$$

Pour un paramètre de priorité supérieure, il est tenu compte de la marge disponible :

$$P_p + M_p < O_p(K) \text{ et } \exists K',\ O_p(K') \leq P_p + M_p \tag{6.5}$$

Pour un paramètre de priorité inférieure au paramètre optimisé, il n'y a pas d'amendement du plan afin de ne pas perturber inutilement les coûts des paramètres plus prioritaires.

3. niveau *planification* : un nouveau plan doit être calculé pour la tâche en cours.Ce niveau est sélectionné si pour toute commande correctrice K' le coût observé pour l'action en cours reste inférieur à la contrainte mais consomme toute la marge disponible.

$$\forall K' \,/\, O_p(K') \ \leq \ O_p(K) \text{ et } P_p + M_p < O_p(K') \leq C_p \tag{6.6}$$

Si le paramètre p est de priorité supérieure au paramètre optimisé, la tâche est replanifiée avec p comme nouveau critère d'optimisation.

4. niveau *définition* : Ce niveau correspond au cas où l'observation dépasse la contrainte pour l'action en cours.

$$C_p < O_p(K) \tag{6.7}$$

De nouveaux objectifs doivent être définis car il est peu probable qu'une simple replanification permette de résorber le sur-coût observé. Ces nouveaux objectifs peuvent correspondre à des scénarios prévus à l'avance ou à des transformations de la mission qui permettent de conserver le maximum de tâches qui la composaient initialement. Dans tous les cas où le niveau *définition* est sélectionné, l'échec de la replanification requiert l'intervention de l'opérateur pour redéfinir des objectifs.

6.3.3 Filtrage des requêtes de modification du plan

Pour modifier le plan, le module d'évaluation/sélection émet une requête vers le module de planification indiquant l'action à partir de laquelle le plan doit être modifié, le niveau de modification ainsi que le paramètre concerné.

Cette requête n'est pas systématiquement émise pour ne pas faire perdre du temps au module de planification et pour laisser le temps au système d'appliquer les modifications du plan. La décision d'émettre une requête alors que la précédente est en cours de traitement se base sur la comparaison entre le niveau et la priorité de la sélection courante par rapport à ceux de la sélection précédente. Ceci permet d'interrompre une planification en cours si un écart plus important est observé mais d'éviter de relancer la planification pour une même priorité et un même niveau de sélection.

On dispose du prédicat $Satisf(Req)$ qui indique si la requête Req a été satisfaite. Dans le cas où une tâche doit être replanifiée, on considère que la requête est satisfaite dès que le nouveau plan est calculé et que l'on peut évaluer le niveau de sélection par rapport aux nouvelles prédictions fournies.

---

procédure **Selection(O[A],A,K,S,po)**
début
pour ($p \in Ord(PARAM)$) **faire**
  $niveau := Niveau(p, O[A], A, K, po)$; /* *Détermination du niveau de sélection* */
  si ($niveau > NORMAL$) **alors**
    si ($Satisf(Req)$ **ou** $PR(p) > Sel.p$ **ou** ($PR(p) = Sel.p$ **et** $niveau > Sel.n$)) **alors**
      $Req := Requete(ModifierPlan(A, niveau, p))$;
      $Sel := (PR(p), niveau)$;
    fin si
    sortie;
  fin si
fin pour
$Sel := (MINIMUM, NORMAL)$; /* *Priorité minimum, niveau normal* */
fin

FIG. 6.2 - *Pseudo-code de la procédure : Selection*

6.3.4 Mise à jour des prédictions et des marges

Lorsque le sous-but de l'action de navigation A_q est atteint, les coûts observés permettent de mettre à jour les marges et les prédictions pour l'action de navigation suivante A_{q+1}. Les prédictions doivent être mises à jour car elles représentent des coûts cumulés pour chaque paramètre depuis le début de la mission. Les valeurs mises à jour, M' et P' pour chaque paramètre sont données par :

$$M'[A_{q+1}] = M[A_{q+1}] + (P[A_q] - O[A_q]) \quad (6.8)$$
$$P'[A_{q+1}] = P[A_{q+1}] - (P[A_q] - O[A_q]) \quad (6.9)$$

6.4 Planification en cours d'exécution

Pour pouvoir limiter la planification à la tâche courante, il faut s'assurer que le nouveau plan obtenu pour cette tâche ne remette pas en cause le plan des tâches de navigation suivantes. Il faut donc également que les actions de réinitialisation, éventuellement planifiées initialement pour la partie non exécutée de cette tâche, fassent partie du nouveau plan. Il est suffisant pour réaliser ces deux conditions de redéfinir les contraintes de la tâche à replanifier de la manière suivante :

$$C'[T] = P[T] + M[T] \quad (6.10)$$

D'autre part, pour permettre de trouver plus efficacement un plan qui satisfasse la contrainte pour le paramètre cause de la replanification, ce paramètre devient le critère d'optimisation de la tâche.

Lorsque la tâche est replanifiée, les nouvelles marges de la tâche sont calculées à partir des marges initiales auxquelles on soustrait l'écart entre le nouveau plan et le précédent :

$$M'[T] = M[T] - (P'[T] - P[T]) \quad (6.11)$$

Si le fait d'avoir changé le critère d'optimisation suffit à corriger l'écart entre l'observation et la prédiction pour un paramètre donné, il n'y a pas d'inconvénients à tenter de replanifier cette même tâche ultérieurement avec un critère d'optimisation encore différent car on conserve les contraintes qui ont servi à définir le plan précédent. En effet d'après les équations (6.10) et (6.11) on a :

$$C''[T] = P'[T] + M'[T] \quad (6.12)$$
$$= P'[T] + M[T] - P'[T] + P[T] \quad (6.13)$$
$$= C'[T] \quad (6.14)$$

Une même tâche ne peut être replanifiée indéfiniment. En effet la priorité définie sur les critères permet d'éviter d'osciller entre des solutions correspondant à la satisfaction de contraintes incompatibles simultanément. D'autre part l'existence de marges de plus en plus réduites pour chaque critère diminue le nombre de plans possibles pour chaque replanification.

6.5 Validation du plan et choix de l'action

Une replanification de la tâche de navigation en cours et l'exécution d'une action de navigation peuvent s'effectuer simultanément. Il faut donc vérifier qu'à la fin de l'exécution de cette action le nouveau plan est valide pour enchaîner sur la première action de ce nouveau plan.

6.5.1 Critère de validité

Soit t_{select} l'instant où la replanification a été lancée, le nouveau plan est valide au temps $t_{validation}$ si :

1. Le critère de sélection n'est pas changé.

2. $M[A'_0] > O[A_j] - O_{t_{select}}$. On ne peut connaître la valeur de $M[A'_0]$ qu'à la fin de la planification de T'. $O[A_j]$ est le vecteur d'observation de la dernière action en cours ou terminée au moment de la validation.

6.5.2 Choix de la commande

Pour déterminer la commande à envoyer au *module de navigation*, le *Superviseur* dispose d'une certaine gamme de vitesses. Nous n'utiliserons que trois vitesses, données dans l'ordre croissant: $V_{min}, V_{nom}, V_{max}$, d'une part pour simplifier la prise de décision mais aussi parce que cela correspond aux spécifications générales du prototype MITHRA. Il faut rappeler que pour éviter des obstacles le *contrôleur du véhicule* peut toujours choisir une vitesse inférieure ou égale à celle de la commande issue du *Superviseur*. Les itinéraires éventuellement plus rapides sont ceux pour lesquels sur certaines routes faisant partie de l'itinéraire, le vitesse conseillée (et limite) est la vitesse V_{max}. Cette vitesse sera prise en compte lors de la planification pour déterminer le coût de la route correspondante.

En fonctionnement normal le Superviseur déclenche les actions selon les informations temporelles décrites par le graphe des intervalles et leurs relations. Il doit également surveiller la consommation d'énergie qui peut être sujette à de grandes variations. Il doit également collecter des données destinées à une exploitation ultérieure : l'historique de tous les messages internes du système et des données qui servent à compléter la description de l'environnement.

Le déclenchement des actions est lié à la gestion du temps, sous deux aspect : les tops fournis par l'horloge interne du système ($\mathcal{H}_{horloge}$) et l'utilisation des relations temporelles pour assurer l'ordonnancement des actions. Une action (resp. une tâche) n'est exécutable que si d'une part la borne inférieure de la contrainte de temps est atteinte et d'autre part sont marquées toutes les relations qui la relient avec des actions (resp. tâches) dont le déclenchement la précède. Ceci permet d'empêcher d'exécuter une action (resp. une tâche) si celles qui la précèdent n'ont pas commencé ou ne sont pas finies.

Pour déclencher et interrompre les actions selon les bornes des intervalles de temps et leur type, on se base sur les tops fournis par l'horloge interne. Ces tops sont émis chaque seconde.

Les tops sont calculés selon le mode des contraintes de temps de la mission :

1. Le top d'exécution pour les actions : $\mathcal{H}_{exec}$.
2. L'heure de début d'exécution de la mission : $\mathcal{H}_{debut}$.

La valeur de $\mathcal{H}_{exec}$ dépend du mode horaire de la mission :

- La mission est en mode absolu : $\mathcal{H}_{exec} = \mathcal{H}_{horloge}$
- La mission est en mode relatif : $\mathcal{H}_{exec} = \mathcal{H}_{horloge} - \mathcal{H}_{debut}$

Une action ou une tâche doit être en cours d'exécution si l'intervalle de temps d'exécution I vérifie : $inf(I) \leq \mathcal{H}_{exec} < sup(I)$.

6.5.3 Mise à jour du graphe d'exécution

Les intervalles d'exécution calculés lors de la planification sont mis à jour avec valeurs effectives de début et de fin des actions observées lors de l'exécution. D'autre part, pour assurer que les actions sont exécutées selon l'ordonnancement prévu, on effectue un marquage qui revient à un tri topologique sur le graphe d'exécution. Dès que toutes les relations partant des intervalles associés aux actions en cours sont marquées, l'exécution des actions suivantes est autorisée.

1. Marquage des Intervalles :
   Les intervalles de temps doivent être marqués pour indiquer qu'ils ont été modifiés. La marque d'un intervalle est son instant de création. Ainsi, quand à la date t_0 des actions $(A_0, A_1 \ldots A_n)$ sont planifiées pour une date d'exécution $inf(I) > t_0$, on marque les intervalles par la date t_0 .
   - En phase de planification, la marque est 0.
   - En phase d'exécution, la marque est l'instant où l'intervalle est modifié.
2. Marquage des Relations :
   Pour un intervalle I_j^e, on marque la relations R_{jk} qui le relie avec l'intervalle suivant I_k^e. Voici la règle de marquage et les propriétés qui permettent de parcourir le graphe d'exécution.
   (a) Marquage de la relation :
   La relation est marquée soit dès que la tâche ou l'action j change de phase est devient active soit quand elle est terminée, ceci selon les contraintes de séquentialité dérivées du graphe des relations.
   Soit $inf(I_j) = h$ et $R_{jk}(marque) < h$ et
   i. $T_j(phase)choisie \rightarrow active$,
   $R_{jk}(type) \in \{Overlaps, Contains, Finishes, Start, Equal\}$

ii. $T_j(phase)active \rightarrow terminee$,
$R_{jk}(type) \in \{Meet, Before\}$.

Alors $R_{jk}(marque) = h$.

(b) Il n'y a pas de relation non marquée avec un intervalle précédent :

$$\begin{cases} R_{ij}(marque) = h \\ \nexists inf(I_i) > h \end{cases}$$

(c) Il existe au moins une relation marquée avec un intervalle précédent :

$$\begin{cases} R_{ij}(marque) = h \\ \exists inf(I_i) \leq h \end{cases}$$

6.6 Intégration des modules du Superviseur

Dans les sections précédentes de ce chapitre nous avons indiqué à plusieurs reprises que les modules d'évaluation/sélection, de planification et de validation fonctionnent de manière asynchrone. Ces modules ne fonctionnement pas en parallèle mais les traitements qu'ils effectuent s'entrelacent dans le temps.

Il nous faut indiquer ici que le superviseur est réalisé à l'aide d'un système de production (OPS5) que nous avons adapté pour permettre une communication par messages avec les autres sous systèmes du robot. Les algorithmes que nous avons présentés sont décrits par des règles de production. Un des bénéfice de cette approche est la possibilité de gérer plusieurs *fils d'activité.*

Un fil d'activité est un traitement réalisé dans un contexte donné. Ce traitement peut être temporairement interrompu puis repris par changement de contexte. Dans un système à base de règles, un contexte est un ensemble de faits commun à plusieurs règles et qui permet l'enchaînement de ces règles tant que ces faits sont vérifiés. Les changements de contexte peuvent avoir lieu lorsque :

1. il n'y a plus de règles activables dans le contexte courant ;
2. une règle est exécutée, contenant une directive explicite de changement de contexte ;
3. un événement externe provoque l'activation de règles dans un contexte différent. Il faut préciser que le mécanisme de sélection des règles activables privilégie les règles traitant les faits les plus récents.

La coordination entre les différents modules du Superviseur peut être représentée au moyen d'automates associés à chaque module. La production d'un fait dans la mémoire de travail du système de production peut être interprétée comme l'envoi d'un message d'un module vers un autre et provoque dans le module receveur un changement d'état.

L'automate du module évaluation/sélection contient deux état : *Attente* et *Eval.*

- La transition *Attente* $\rightarrow$ *Eval* est déclenchée à la réception de l'information d'état en provenance du module de navigation.

- La transition *Eval* → *Attente* est déclenchée lorsque la sélection du niveau de modification du plan est effectuée. Selon le résultat de la sélection, l'automate produit un message pour déclencher une replanification ou une simple mise à jour des coûts prédits et des marges.

Le module de planification est décrit par deux états : *Attente* et *Planif.*

- La transition *Attente* → *Planif* est déclenchée à la réception de la requête de planification en provenance du module d'évaluation/sélection. Le module de planification dispose alors de la description d'une ou plusieurs tâches à planifier.

- La transition *Planif* → *Planif* est déclenchée lorsqu'une requête de planification est reçue alors qu'une planification est déjà en cours. Cette dernière est abandonnée et une nouvelle planification est lancée avec la nouvelle description de tâches.

- La transition *Planif* → *Attente* est déclenchée lorsque la planification est terminée. Le message émis dépend du succès de la planification. Ce message est reçu par le module de validation et par l'interface homme-machine.

Le module de validation est également décrit par un simple automate à deux états, un état d'attente et un état dans lequel s'effectue la validation du nouveau plan produit et la mise à jour du graphe d'exécution.

6.7 Conclusion

Le fonctionnement en parallèle du Superviseur et des sous modules de navigation et de perception a été mis à profit pour détecter au plus tôt les échecs d'exécution liés à la satisfaction des contraintes opérationnelles. Les replanifications sont effectuées sans attendre la fin de l'action de navigation en cours. Ceci permet de réduire le temps d'attente éventuellement nécessaire à la fin de cette action pour pouvoir exécuter le nouveau plan. La qualité des prédictions influe sur les performances du système car les prédictions permettent de décider quel est le bon moment pour déclencher des replanifications. En l'absence d'un modèle prédictif réellement satisfaisant, la décision de replanifier est prise au dernier moment et cela a toujours pour effet de réduire l'ensemble des solutions de reprise.

La mise en œuvre de ce système avec un système à base de règles tel qu'OPS5 nous a permis de faire coexister très simplement des traitements cycliques de nature aussi bien symbolique que numérique. Néanmoins, pour des raisons d'efficacité, le Superviseur ne peut être écrit exclusivement en suivant ce paradigme et certaines parties, comme l'algorithme de planification, doivent être écrites dans un langage procédural.

Chapitre 7

Conclusion

L'objectif général de notre travail est d'accroître la flexibilité des robots de service. Pour accroître cette flexibilité nous avons suivi deux approches complémentaires : contribuer d'une part à faciliter l'interaction entre le système robotique et un opérateur humain et d'autre part doter le système d'une représentation de la tâche et de l'environnement qui correspond le mieux possible aux nécessités de ce service.

Pour cela nous avons suivi l'approche consistant à simplifier la description des missions en la plaçant à un niveau d'abstraction élevé pour en permettre la spécification par un opérateur non-informaticien. Nous avons dans le même temps enrichi les moyens de choix de comportements du robot par la possibilité de définir des contraintes quantitatives sur l'exécution de ces missions.

L'étude bibliographique nous a permis de nous positionner par rapport aux recherches passées et actuelles concernant la robotique. Celles-ci portent sur les problèmes de perception, de représentation, de planification et de contrôle le tout englobé dans le problème plus général d'architecture. Nous nous sommes d'abord concentrés sur deux aspects qui sont étroitement liés : celui de la description des tâches et celui de la complexité de la planification. Les limites de l'approche par résolution de problème de type GPS ou STRIPS pour la planification nous ont amenés à choisir une approche appropriée à notre domaine d'application : la robotique mobile de surveillance. Celle-ci est caractérisée par le fait que le système ne modifie pas intentionnellement son environnement et que par conséquent il n'a pas à raisonner par avance sur les modifications éventuelles de l'environnement qui résulteraient de ses actions. Ce raisonnement, s'il est nécessaire, se fait a-posteriori.

Ceci nous a conduit naturellement au problème de contrôle d'exécution. Nous sommes partis de l'hypothèse qu'il n'était pas absolument nécessaire que le système de supervision ait des caractéristiques temps-réel pour que le système dans son ensemble soit fiable et efficace. La solution repose, comme d'autres auteurs l'ont montré, sur une architecture appropriée permettant la coexistance d'un sous-système réactif piloté par un sous-système délibératif.

Le formalisme proposé pour la spécification de mission nous a imposé de résoudre le problème de la nature symbolique et numérique de l'évaluation de l'état du système en cours d'exécution.

7.1 Notre contribution

Nous avons donc été amenés à proposer un ensemble de techniques permettant de doter un robot mobile d'un système de supervision pour la navigation sous contraintes. L'étude et la réalisation d'un tel système ont comporté trois aspects : dialogue avec une interface homme-machine, planification de mission et contrôle d'exécution.

Notre travail sur l'interface homme-machine a eu pour objet de montrer comment les critères pour une bonne interface déterminaient certaines des fonctionnalités du système de supervision et d'autre part comment les contraintes techniques de ce dernier limitaient les possibilités d'interaction. Il a montré également combien pouvait être utile une conception concertée de ces deux systèmes.

Notre problème de planification a pu être formulé comme un problème d'optimisation multi-contraintes dans un espace d'états discret. Notre algorithme de planification permet de choisir un itinéraire optimum satisfaisant diverses contraintes opérationnelles et les contraintes temporelles liées à l'environnement.

Il faut souligner que l'approche choisie peut conduire à une exploration exhaustive, donc coûteuse en temps, de l'espace d'états, défaut que n'ont pas les techniques d'exploration à horizon limité. Pour tout système qui doit interagir avec un environnement potentiellement changeant, se pose la question de l'efficacité en temps de calcul des mécanismes de planification et contrôle d'exécution mis en œuvre. Dans le chapitre architecture, nous avons exposé l'idée selon laquelle un système de planification n'a pas nécessairement besoin d'être un système temps-réel pour peu que l'on soit apte à définir les actions réflexes de base à activer selon le contexte.

Nous résumons ici ce qui justifie notre approche par rapport aux techniques d'exploration à horizon limité :

- pour le niveau d'abstraction considéré la densité est nettement moins élevée que celle qui serait nécessaire pour appliquer la même technique pour faire de la planification de trajectoire ; ainsi, dans la plupart des cas, le temps de planification hors ligne peut rester raisonnable.

- nous n'avons pas tenté de satisfaire des objectifs de planification temps-réel par la méthode proposée par Korf [Korf, 1990] car le système doit informer l'opérateur de l'existence d'un plan complet afin qu'il sache avant l'exécution si la mission est réaliste. Néanmoins, l'approche de Korf peut être utile lorsque la planification doit avoir lieu pendant l'exécution.

- nous avons défini une stratégie de contrôle d'exécution pour que pendant l'exécution de la mission les modifications du plan soient d'un coût le plus faible possible tout en respectant le principe d'une planification complète. Ainsi lorsqu'il faut replanifier une tâche, les contraintes associées sont redéfinies afin que le nouveau plan de cette tâche ne remette pas en cause le plan pour les tâches qu'il reste à exécuter. Cette redéfinition des contraintes a également pour effet de réduire l'espace d'états pour cette replanification.

Le contrôle d'exécution, même à ce niveau d'abstraction élevé, n'est pas uniquement basé sur une évaluation qualitative, mais aussi quantitative de l'état du système car les missions sont contraintes quantitativement. Cette approche pose le problème de la connaissance d'un modèle satisfaisant du comportement du robot. En l'absence d'un modèle analytique suffisement précis, on peut se baser sur des mesures statistiques d'exécution. Cependant, nous avons appliqué cette méthode uniquement en simulation, faute d'un terrain d'experimentation de taille suffisante.

Les algorithmes que nous avons proposés sont adaptés pour un certain type d'application des robots mobiles. Elles sont caractérisées par le fait que les tâches sont routinières et effectuées dans un environnement connu et structuré. L'exécution doit être suffisamment prévisible pour que les contraintes imposées par l'opérateur puissent être relativement proches d'un comportement moyen satisfaisant. Les situations de replanification complète en cours d'exécution doivent rester l'exception.

Nous avons limité notre étude du contrôle d'exécution à l'aspect gestion des contraintes opérationnelles pour la navigation. Dans ce cadre, nous avons proposé un mécanisme original qui limite les effets de la non prédictibilité des temps de planification et qui coordonne les activités du niveau réactif et du niveau délibératif.

Dans le cas d'un système robotique mobile complexe, il est certain que la supervision ne doit pas se limiter à cela mais doit également gérer la coordination des différentes activités du système (détection d'obstacles, repérage d'amers, etc) si celui-ci possède des capteurs directionnels comme c'est le cas pour les robots mobiles équipés d'un système de vision active [Kosecka *et al.*, 1993]. Cette question dépasse le cadre de ce manuscrit mais le lecteur intéressé peut se référer à l'article [Causse et Christensen, 1994] dans lequel nous avons abordé ce problème sous l'angle de la conception d'un système de contrôle hiérarchique modélisé par des réseaux de Petri colorés.

7.2 Perspectives

Les différentes parties du système sont conçues de manière à former un tout cohérent et suffisant à des fins de démonstration. Mais on peut concevoir sans difficultés que des parties de ce système peuvent constituer des briques d'un futur système plus complexe, soit en termes de tâches plus complexes (intégrant de la manipulation), soit en termes de nombre de robots cohabitant dans le même environnement.

Mais si l'on s'interroge sur les perspectives de ce travail, on peut également s'interroger sur les perspectives plus générales de la recherche en robotique. C'est une évidence de dire que la recherche en robotique, et notamment en robotique mobile suppose des investissements humains et matériels lourds. Ces recherches sont freinées au jour le jour autant par des problèmes techniques que par des problèmes de nature réellement scientifique. C'est sans doute pour cela que Etzioni prenant le contre-pied de Brooks dans "Intelligence without Robots" [Etzioni, 1993] milite en faveur d'une nouvelle voie de recherche en Intelligence Artificielle basée sur l'étude de Softbots (software robots). Ces softbots sont assimilés à des agents autonomes et doivent évoluer dans le monde "réel", changeant et imprévisible, que sont les systèmes informatiques complexes. En

fait, si ce *monde* possède effectivement ces caractéristiques qui poussent Etzioni à le qualifier de réel c'est parce que ces systèmes sont pilotés par des humains. Il y a cependant une différence de nature essentielle entre le monde envisagé par Etzioni et le monde réel. Dreyfus dans [Dreyfus et Dreyfus, 1989] précise cette différence. Elle vient de ce que lorsqu'on s'intéresse à une partie du monde réel, nous sommes consciemment ou inconsciemment obligés de présupposer le monde réel dans son ensemble. Alors que le monde envisagé par Etzioni, de même que celui qu'avait envisagé Winograd pour le système SHRDLU[Winograd, 1973], doit être étendu pour prétendre affronter le monde réel. Et jusqu'à présent on ne connaît pas d'extension qui ait donné réellement satisfaction. Ceci n'est qu'une des difficultés à laquelle est confrontée la robotique, et pour encore longtemps, mais elle est fondamentale si l'on veut faire entrer un jour des robots factotum dans nos foyers.

Table des figures

Bibliographie

[Abowd *et al.*, 1992] G. Abowd, J. Coutaz, et L Nigay. – Structuring the space of interactive system properties. – In *IFIP WG2.7 Conference on Engineering for Human-Computer Interaction*, Ellivuori, Finland, Aout 1992.

[Albus *et al.*, 1987] J. Albus, H. McCain, et R. Lumia. – NASA/NBS standard reference model for telerobot control system architecture. – Tech Report n° 1235, NBS, 1987.

[Allen, 1982] J.F. Allen. – Maintaining knowledge about temporal intervals. – *Communications of the ACM*, 26:832–843, 1982.

[Anderson et Donath, 1990] T.L. Anderson et M. Donath. – Animal behavior as a paradigm for developing robot autonomy. – *Robotics and Autonomous Systems*, 6(1,2):145–168, Juin 1990.

[Arkin, 1987] R. C. Arkin. – *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments.* – Thèse de Doctorat, Univ. Of Massachusetts, September 1987.

[Arkin, 1990] R. C. Arkin. – Integrating behavioral, perceptual and world knowledge in reactive navigation. – *Robotics and Autonomous Systems*, 6(1,2):105–122, Juin 1990.

[Barrouil et Mampey, 1990] C. Barrouil et R. Mampey. – Raisonnement revisable pour la planification de missio ns de robots mobiles. – In *Seminaire EC2 "Les robots mobiles : état de l'art et appl ications"*, 1990.

[Beer *et al.*, 1990] R. D. Beer, H. J. Chiel, et L. S. Sterling. – A biological perspective on autonomous agent design. – *Robotics and Autonomous Systems*, 6(1,2):169–186, Juin 1990.

[Bobet et Reignier, 1989] P. Bobet et P. Reignier. – Perception et navigation pour un robot mobile. – Projet ENSIMAG, LIFIA - INPG, Grenoble, 1989.

[Boissier, 1993] O. Boissier. – *Le Problème du contrôle dans un système général de vision.* – Thèse de Doctorat, LIFIA - IMAG, Grenoble - France, Janvier 1993.

[Brooks, 1985] R. A. Brooks. – A robust layered control system for a mobile robot. – Rapport technique n° 864, M.I.T., 1985.

[Brooks, 1989] R. Brooks. – A robot that walks: Emergent behavior from a caref ully evolved network. – *Neural Computation*, 1:2:253–262, 1989.

[Brooks, 1991] R. Brooks. – New approaches to robotics. – *Science*, 253:1127–1232, Septembre 1991.

[Bylander, 1991] T. Bylander. – Complexity results for planning. – In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 1991.

[Camargo, 1991] R. Ferraz De Camargo. – *Architecture materielle et logicielle pour le contrôle d'exécution d'un robot mobile autonome.* – Thèse de Doctorat, Université Paul Sabatier, Juillet 1991.

[Cassandras, 1993] Ch. G. Cassandras. – *Discrete Event Systems - Modeling and Performance Analysis.* – Aksen Associates, 1993.

[Causse et Christensen, 1994] O. Causse et H. I. Christensen. – Hierarchical Control Design based on Petri Net Modeling for an Autonomous Mobile Robot. – In *SMART Workshop*, Ispra, Italy, Avril 1994.

[Chapman, 1987] D. Chapman. – Planning for conjunctive goals. – *Artificial Intelligence*, 32:333–377, 1987.

[Chatila *et al.*, 1992] R. Chatila, R. Alami, B. Degallaix, et H. Laruelle. – Integrated planning and execution control of autonomous robot actions. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 2689–2696, Nice, May 1992.

[Chatila et Laumond, 1985] R. Chatila et J. P. Laumond. – Position referencing and consistent world modeling for mobile robots. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, St Louis, USA, Mars 1985.

[Chochon, 1991] H. Chochon. – Object oriented design of mobile robot control systems. – In *Second International Symposium on Experimental Robotics*, Toulouse, France, Juin 1991.

[Christensen et Grove, 1991] J. Christensen et A. Grove. – A formal model for classical planning. – In *International Joint Conference on Artificial Intelligence*, pp. 246–251, 1991.

[Connell, 1992] J. H. Connell. – Sss: A hybrid architecture applied to robot navigation. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Nice, France, Mai 1992.

[Coutaz, 1987] J. Coutaz. – PAC, an implementation model for dialog design. – In *Interact'87*, Stuttgart, September 1987.

[Crowley *et al.*, 1991] J. L. Crowley, O. Causse, et P. Reignier. – Layers of control in autonomous navigation. – In *Mechatronics and Robotics*, Aachen, 1991.

[Crowley et Christensen, 1994] J. Crowley et H. Christensen. – *Vision as Process.* – Sringer Verlag, Basic Research Series, 1994.

[Crowley et Coutaz, 1986] J. L. Crowley et J. Coutaz. – Navigation et modélisation pour un robot mobile. – *Technique et Science Informatique*, 5(5), Octobre 1986.

[Crowley, 1981] J. L. Crowley. – *A Representation for Visual Information.* – Thèse de Doctorat, Carnegie-Mellon University, 1981.

[Crowley, 1986] J. L. Crowley. – Navigation and world modeling for a mobile robot: A progress report. – *3rd International Symposium on Robotics in Construction, Marseilles*, Juin 1986.

[Crowley, 1987] J. L. Crowley. – The state of art in mobile robotics. – In *The Fourth International Symposium on Robotics in Construction*, Haifa, Israel, June 1987.

[Crowley, 1989] J. L. Crowley. – World modeling and position estimation for a mobile robot using ultrasonic ranging. – In *IEEE International conference on Robotics and Automation*, pp. Vol 2. 674–680, May 1989.

[de Saint Vincent, 1990] A. Robert de Saint Vincent. – Téléopération niveau tâche et intelligence embarquée pour le robot mobile AMR. – In *Séminaire Les Robots Mobi iles*, Paris-La-Défense, Mars 1990.

[Dean *et al.*, 1988] T. Dean, R.J. Firby, et D. Miller. – Hierarchical planning involving deadlines, travel time, and resources. – *Computational Intelligence*, 4(4):381–398, 1988.

[Dorn, 1992] J. Dorn. – Temporal reasoning in sequence graphs. – In *Proc. of the Nat. Conf. on Artificial Intelligence*, San Jose, California, March 1992.

[Dreyfus et Dreyfus, 1989] H. Dreyfus et S. Dreyfus. – *Mind over Machine : the power of human intuition and expertise in the era of the computer.* – Basil Blackwell, 1989.

[Elfes et Talukdar, 1983] A. Elfes et S. N. Talukdar. – A distributed control system for the CMU rover. – In *8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Allemagne, 1983.

[Erman, 1980] L. Erman. – The HEARSAY-II speech understanding system: integrating knowledge to resolve uncertainty. – *Computing Surveys*, 12:2:213–253, 1980.

[Etzioni, 1993] O. Etzioni. – Intelligence without robots: A reply to brooks. – *AI Magazine*, pp. 7–13, Winter 1993.

[Fennema, 1991] C. L. Fennema. – *Interweaving Reason, Action and Perception.* – Thèse de Doctorat, Univ. of Massachusetts, September 1991.

[Fikes *et al.*, 1972] R. E. Fikes, P. E. Hart, et N. J. Nilsson. – Learning and executing generalized robot plans. – *Artificial Intelligence*, 3(3):251–288, 1972.

[Fikes et Nilsson., 1971] R. E. Fikes et J. N. Nilsson. – Strips: a new approach to the application of theorem proving to problem solving. – *Artificial Intelligence*, 2:189–208, 1971.

[Firby, 1987] R. J. Firby. – An investigation into reactive planning in complex domains. – In *Proc. of the Nat. Conf. on Artificial Intelligence*, 1987.

[Fraichard, 1992] Th. Fraichard. – *Planification de mouvement pour mobile non-holonome en espace de travail dynamique.* – Thèse de Doctorat, Institut National Polytechnique de Grenoble, Avril 1992.

[Fugimura et Samet, 1989] K. Fugimura et H. Samet. – A hierarchical strategy for path planning among moving obstacles. – *IEEE transactions on Robotics and Automation*, 5(1):61–69, February 1989.

[Gaspart, 1987] P. Gaspart. – *Langages de programmation de la robotique.* – Traité des Nouvelles Technologies. Hermes, 1987.

[Georgeff et Lansky, 1986] M. P. Georgeff et A. L. Lansky. – Procedural knowledge. – In *Proceedings of the IEEE Special Issue on Knowledege Representation*, pp. 74:1383–1398, 1986.

[Georgeff et Lansky, 1987] M. P. Georgeff et A. L. Lansky. – Reasoning and planning in dynamic domains: An experiment with a mobile robot. – Rapport technique n° 380, Artificial Intelligence Center, SRI International , Menlo Park, California, 1987.

[Georgeff, 1987] M. P. Georgeff. – Planning. – Technical Note n° 418, SRI International, 1987.

[Gondran et Minoux, 1985] M. Gondran et M. Minoux. – *Graphes et algorithmes.* – Eyrolles, Collection de la Direction des Etudes et Recherches d'Electricité de France, 1985.

[Goto et Stentz, 1987] Y. Goto et A. Stentz. – Mobile robot navigation: the CMU system. – *IEEE Expert*, 2:4:44–54, 1987.

[Gupta et Nau, 1991] N. Gupta et D. S. Nau. – Complexity results for blocks-world planning. – In *Proc. of the Nat. Conf. on Artificial Intelligence*, Anaheim, CA, USA, Juillet 1991.

[Hart *et al.*, 1968] Hart, Nilsson, et Raphael. – A formal basis for the heuristic determination of minimum cost paths. – *IEEE transaction on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[Hebert *et al.*, 1989] M. Hebert, C. Caillas, E. Krotkov, I. S. Kweon, et T. Kanade. – Terrain mapping for a roving planetary explorer. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 1989.

[Hörmann, 1991] A. Hörmann. – A survey of control architecture for advanced autonomous systems. – In *ESPRIT II Exploratory Action 5610, Workshop on Mobile Robotics for Civil Works*, Brussels, Juin 1991.

[Kaebling, 1986] L. P. Kaebling. – An architecture for intelligent reactive systems. – Rapport technique n° 400, SRI, Octobre 1986.

[Kant et Zucker, 1988] K. Kant et S. Zucker. – Planning collision-free trajectories in time-varying environments : a two level hierarchy. – In *IEEE Int. Conf. on Robotics and Automation*, 1988.

[Kœnig et Crochon, 1988] A. Kœnig et E. Crochon. – TRAM : Tableau noir pour robotique autonome mobile. – In *8ième Journées Internationales sur les Systèmes Experts et leurs Applications, Avignon*, 1988.

[Korf, 1990] R. E. Korf. – Real-time heuristic search. – *Artificial Intelligence*, 42(2-3):189–211, Mars 1990.

[Kosecka *et al.*, 1993] J. Kosecka, H. I. Christensen, et R. Bajcsy. – Discrete event modeling of navigation and gaze control. – *Int. Journal of Computer Vision*, 1993.

[Kuipers et Byun, 1988] B. J. Kuipers et Yung-Tai Byun. – A robust, qualitative method for robot spatial learning. – In *The 7th National Conference on Artificial Intelligence. AAAI*, 1988.

[Latombe, 1990] J-C. Latombe. – *Robot motion planning.* – Kluwer Academic Press, 1990.

[Laumond *et al.*, 1989] J-P. Laumond, T. Siméon, R. Chatila, et G. Giralt. – Trajectory planning and motion control for mobile robots. – In J-D. Boissonnat et J-P. Laumond (édité par), *Geometry and Robotics*, pp. 133–149. Lecture Notes in Computer Science, Vol 391, Springer-Verlag, 1989.

[Lozano-Perez et Wesley, 1979] T. Lozano-Perez et M. A. Wesley. – An algorithm for planning collision-free paths among polyhedral obstacles. – In *Communication fo the ACM, Vol 22*, pp. 560–570, October 1979.

[Lozano-Pérez, 1987] T. Lozano-Pérez. – A simple motion planning algorithm for general robot manipulators. – *IEEE Trans. Robotics and Automation*, 3(3):224–238, June 1987.

[Maes, 1989] P. Maes. – A spreading activation network for action selection. – In *Intelligent Autonomous Systems 2*, pp. 875–886, Amsterdam , Hollande, 1989.

[Maximov et Meystel, 1992] Y. Maximov et A. Meystel. – Optimum design of multiresolution hierarchical control systems. – In *IEEE International Symposium on Intelligent Control*, pp. 514–520, Glasgow, Scotland, U.K., Aout 1992.

[Mesarovic *et al.*, 1970] M. D. Mesarovic, D. Macko, et Y. Takahara. – *Theory of hierarchical, multilevel systems.* – Academic Press, New York, San Fransisco, London, 1970.

[Moravec, 1988] H. P. Moravec. – Sensor fusion in certainty grids for mobile robots. – *AI magazine*, pp. 61–74, Summer 1988.

[Newell et Simon, 1963] A. Newell et H. A. Simon. – *GPS, a program that simulates human thought.* – Computer and Thought. Mc Graw-Hill, New York, 1963.

[Nilsson, 1980] N. J. Nilsson. – *Principles of Artificial Intelligence.* – Tioga Publishing Company, Palo Alto, California, 1980.

[Nilsson, 1984] N. J. Nilsson. – Shakey the robot. – Technical Note n° 347, Artificial Intelligence Center, SRI International, Menlo Park, California, 1984.

[Noreils, 1989] F. Noreils. – *Contrôle d'exécution de plans d'actions pour un robot mobile.* – Thèse de Doctorat, Université Paul Sabatier, Toulouse, Novembre 1989.

[Noreils, 1991] F. Noreils. – Adding a man/machine interface to an architecture for mobile robots. – In *Proc. of the IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, Osaka, Japon, 1991.

[Ow *et al.*, 1988] Peng Si Ow, S. F. Smith, et A.Thiriez. – Reactive plan revision. – In *Proc. of the Nat. Conf. on Artificial Intelligence*, 1988.

[Pan et Luo, 1990] T. J. Pan et R. C. Luo. – Motion planning for mobile robots in a dynamic environment with moving obstacles. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 578–583, Cincinnati, OH (USA), May 1990.

[Payton, 1986] D. Payton. – An architecture for reflexive autonomous vehicle control. – In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, p. 1838ff, 1986.

[Pearl, 1981] J. Pearl. – *Heuristics.* – Addison Wesley, 1981.

[Reignier et Crowley, 1991] P. Reignier et J. L. Crowley. – Raisonnement géométrique et contrôle d'exécution pour la mobilité. – In *Rapport interne - LIFIA*, 1991.

[Reignier, 1993] P. Reignier. – Fuzzy logic techniques for mobile robot obstacle avoidance. – In *IRS*, Zakopane, Poland, 1993.

[Sacerdoti, 1973] E. D. Sacerdoti. – Planning in a hierarchiy of abstraction spaces. – In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pp. 412–430, Stanford, California, June 1973.

[Sacerdoti, 1975] E. D. Sacerdoti. – The nonlinear nature of plans. – In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 1975.

[Schiele et Crowley, 1994] B. Schiele et J. L. Crowley. – A comparison of position estimation techniques using occupancy grids. – *Robotics and Autonomous Systems*, 12(3–4), Apr. 1994.

[Schoppers, 1987] M. J. Schoppers. – Universal plans for reactive robots in unpredictable environnements. – In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 1987.

[Schoppers, 1992] M. Schoppers. – On hierarchical architectures for robot control software. – *AICS mailing list*, October 1992.

[Sheridan, 1992] Th. B. Sheridan. – *Telerobotics, automation and human supervisory control.* – MIT Press, 1992.

[Simmons, 1990] R. Simmons. – Concurrent planning and execution for a walking robot. – Rapport technique n° CMU-RI-TR-90-16, Carnegie Mellon University, Pittsburgh, Juil. 1990.

[Stefik, 1981] M. Stefik. – Planning and metaplanning (molgen). – *Artificial Intelligence*, 1981.

[Sussman, 1973] G. J. Sussman. – A computational model for skill acquisition. – Rapport technique n° 297, MIT AI, Cambridge, MA, 1973.

[Tate, 1974] A. Tate. – INTERPLAN: A plan generation system wich can deal with interactions between goals. – *Machine Intelligence Research Unit Memorandum, Edinburg*, 1974.

[Tate, 1976] A. Tate. – NONLIN : A hierarchical non-linear planner. – DAI Memo n° 25, Edinburgh : Dept. of Artificial Intelligence, Univ. of Edinburgh, 1976.

[Warren, 1974] D.H.D. Warren. – WARPLAN : A system for generating plans. – Memo n° 76, Department of Computational Logic, Edinburgh, 1974.

[Winograd, 1973] T. Winograd. – *A procedural model of language understanding.* – Computer Models of Thought and Language. W.H. Freeman Press, 1973.

Thèse de Monsieur Olivier CAUSSE

"Navigation sous contraintes : planification et contrôle d'exécution pour un robot mobile autonome"

Résumé :

Le travail présenté dans ce mémoire propose une contribution à la planification et au contrôle d'exécution de missions pour un robot mobile autonome, en environnement structuré, dynamique et partiellement connu : il porte sur l'étude et la réalisation d'un système de supervision pour robot mobile. Ce superviseur, qui se trouve au sommet d'une hiérarchie de processus de perception et de commande du robot, a pour rôle de planifier la mission spécifiée par un opérateur et d'en contrôler l'exécution tout en assurant le dialogue entre l'opérateur et le robot.

Nous proposons d'abord un langage permettant, à un operateur non informaticien, de spécifier des mission avec des contraintes pouvant porter soit sur la mission elle même (coûts en durée, énergie, risque, distance, localisation) soit sur l'environnement (accessibilité variable dans le temps, possibilités de recharge ou de relocalisation). La planification s'énonce alors comme un problème de recherche d'un chemin optimal contraint dans un graphe. Pour la résolution de ce problème nous introduisons la notion de chemin efficace appliquée à un nouvel algorithme de recherche heuristique pour résoudre le problème des contraintes multiples. La détermination des chemins efficaces repose sur une représentation d'état adaptée aux contraintes sur l'environnement. Les caractéristiques de complexité de cet algorithme de planification sont comparées à celles de l'algorithme A*.

L'exécution de la mission doit assurer l'activation et la supervision des processus chargés d'accomplir les tâches planifiées. L'architecture de contrôle du robot défini une décomposition hiérarchique du contrôle entre le niveau réactif qui réalise les actions et le superviseur qui évalue si le comportement produit satisfait les contraintes. L'évaluation mise en œuvre permet une réponse échelonnée au problème d'échec d'exécution. L'organisation du superviseur en modules asynchrones tient compte de la non prédictibilité des temps de planification, de sorte que le suivi d'exécution et la recherche de plans alternatifs s'exécutent sans pénaliser la réactivité du système.

L'ensemble de ces problèmes nous a amené à réaliser d'une part, une interface homme-machine évoluée et d'autre part, un système de supervision en OPS5. Le fonctionnement cyclique, en chaînage avant, de ce système de production assure une bonne réactivité pour le contrôle des processus temps-réel et pour le dialogue avec un opérateur.

Mots clés : robotique mobile, planification d'itinéraire, supervision, interface homme-robot.