Département de formation doctorale en informatique **École doctorale IAE + M**
UFR STMIA

Architecture de contrôle pour robot mobile en environnement intérieur structuré

THÈSE

présentée et soutenue publiquement le 19 Juin 1998

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Olivier AYCARD

Composition du jury

Rapporteurs :	M. René Schott	Professeur Université Henri Poincaré–Nancy I
	Mme Marie-Claude Thomas	Professeur Université de Nice–Sophia Antipolis
	M. Raja Chatila	Directeur de Recherche LAAS–CNRS
Directeurs :	M. Jean-Paul Haton	Professeur Institut Universitaire de France
	M. François Charpillet	Chargé de recherche INRIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Mots-clés: Robotique mobile, Architecture de contrôle, Navigation réactive, Apprentissage et reconnaissance de balises naturelles, Modèles de Markov cachés, Processus de décision markoviens

Un robot mobile évoluant dans un environnement dynamique est muni de capteurs afin de percevoir son environnement. Les données brutes issues de ces capteurs ne sont pas exploitables directement. Elles doivent être interprétées de manière adéquate pour être utilisables. Cette interprétation joue un rôle crucial, puisqu'elle permet au robot de maintenir un modèle de son environnement. Plutôt que de construire et mettre à jour un modèle complet de l'environnement, nous nous sommes intéressés dans ce mémoire à apprendre et à reconnaître les balises naturelles d'un environnement intérieur structuré que sont par exemple les portes ouvertes ou les intersections en utilisant des modèles de Markov cachés. Ces balises sont suffisantes pour la réalisation des fonctionnalités dans lesquelles un modèle de l'environnement est nécessaire. Elles constituent des points de repères de l'environnement utilisables pour la localisation. Elles déterminent l'état de l'environnement, et servent donc à un robot mobile en train de se déplacer à choisir la meilleure action à exécuter pour atteindre son but.

Notre approche a été validée par la réalisation d'un module de localisation qui en fonction de la position du robot par rapport aux balises de l'environnement détermine sa position dans l'environnement et la réalisation d'un module de contrôle d'exécutions d'actions qui en fonction de l'état des balises de l'environnement choisit la meilleure action à exécuter. Ces deux modules ont été intégrés dans une architecture de contrôle que nous avons développée. Cette architecture comporte deux niveaux (un niveau pour la planification des missions et l'autre pour leur exécution).

Notre mémoire se décompose en deux parties :

- La première partie traite de la conception et de la réalisation de notre architecture de contrôle.
- La deuxième partie décrit notre approche et son utilisation pour réaliser un module de localisation et un module de contrôle d'exécution.

Notre approche a été validée sur un robot mobile Nomad200, et de nombreuses illustrations de nos travaux se trouvent dans le mémoire.

Remerciements

Je remercie toutes les personnes qui ont participé de près ou de loin à la réussite de ma Thèse : ma famille, mes directeurs de Thèse, mes rapporteurs de Thèse, tous mes amis et mes collègues de travail.

A ma femme,
A ma mère,
A ma sœur.

Table des matières

Table des figures

1

Introduction générale

1.1 Motivations

L'Intelligence Artificielle cherche à mécaniser des capacités habituellement attribuées à l'Intelligence Humaine, telles que la perception, l'acquisition de connaissances, le raisonnement et la prise de décision. La robotique mobile constitue ainsi un cadre privilégié des techniques d'Intelligence Artificielle. De plus, la robotique mobile constitue un véritable défi pour les décennies à venir. En effet, les domaines d'application des robots d'intervention ou de service sont nombreux : surveillance de sites, aide aux handicapés, manipulation d'objets, maintenance de centrales nucléaires, exploration planétaire ou sous-marine. Les premiers travaux en robotique mobile en milieu industriel ne conféraient au véhicule qu'une intelligence très limitée et nécessitaient la réalisation d'importants et coûteux travaux d'infrastructure. C'était le cas en particulier des chariots filoguidés, limitant le déplacement du robot à des voies lui étant réservées, ou des robots industriels se déplaçant en suivant des lignes peintes au sol ou au plafond, et se localisant grâce à l'utilisation de codes barres. Etant donné que dans ce type d'applications, les actions du robot sont relativement limitées et exécutées de manière parfaite, le problème du contrôle d'un robot mobile est réduit essentiellement au problème de la planification de tâches. Pourtant ces restrictions et ces aménagements ne peuvent s'appliquer que pour un nombre très restreint d'applications.

Dans des applications plus complexes, le robot mobile réalisant des missions non répétitives dans un environnement qui ne lui est pas dédié, doit posséder un minimum d'intelligence. En effet, dans un environnement vaste et dynamique, les missions qui lui sont attribuées ne peuvent être complètement prédéterminées. Le robot doit donc être autonome au niveau de l'interprétation et de la planification des missions, et aussi au niveau de leur exécution. Le degré d'autonomie peut être plus ou moins élevé. Il se caractérise par la variété des situations que le robot peut appréhender, et se traduit par l'expression plus ou moins précise de la nature des tâches qui lui sont assignées, c'est-à-dire par leur sémantique.

Un robot mobile autonome doit posséder deux types de facultés : des facultés cognitives et des facultés physiques, qui s'intègrent dans une structure qui gère leurs organisations et leurs interactions ; l'architecture de contrôle.

Les facultés cognitives analysent et décomposent les missions, et supervisent leur exécution. En fonction de l'état courant du robot et de l'environnement, elles décident des actions futures en anticipant sur les états futurs. Elles doivent également intégrer les changements de l'environnement qui ont une influence sur la mission à réaliser et y réagir en temps borné.

Les facultés physiques permettent d'exécuter ces missions, de faire face aux imprévus et d'interpréter l'environnement. Elles constituent le lien entre les facultés cognitives et le monde physique. Les actions élémentaires constituant la mission du robot doivent être affinées en boucle de perception/action très proches des capteurs et des actionneurs. De plus, ces boucles doivent intégrer des imperfections, c'est le cas par exemple, lorsque la position d'un but à atteindre est incertaine, et que des imprévus tels que des obstacles se présentent pendant le cheminement du robot. D'autre part, les données brutes issues des capteurs ne sont pas exploitables directement. Elles doivent être interprétées de manière adéquate pour être utilisables.

Cette interprétation joue un rôle crucial dans une architecture de contrôle. Elle permet au robot de maintenir un modèle topologique unique de son environnement. Ce modèle intervient à tous les niveaux d'une architecture de contrôle. Il permet au robot de se localiser dans son environnement, c'est-à-dire de connaître sa position dans son environnement. Lors de l'exécution d'une mission, le robot doit en fonction du contexte choisir la meilleure manière d'exécuter les actions du plan. Il se sert du modèle de l'environnement pour analyser sa situation et décider comment accomplir une action. C'est d'autre part le seul moyen d'obtenir une abstraction symbolique des données des capteurs à transmettre à des modules du niveau cognitif tels que le planificateur de mission.

1.2 Sujet de recherche

Dans cette thèse, nous proposons une nouvelle méthode pour interpréter les données des capteurs. Plutôt que de construire et mettre à jour un modèle complet de l'environnement, nous nous sommes intéressés à reconnaître les balises naturelles d'un environnement intérieur structuré que sont par exemple les portes ouvertes ou les intersections de couloirs en utilisant des modèles stochastiques. Ces balises sont suffisantes pour la réalisation des fonctionnalités dans lesquelles un modèle de l'environnement est nécessaire. Elles constituent des points de repère de l'environnement utilisables pour la localisation. Elles déterminent l'état de l'environnement, et servent donc à un robot mobile en train de se déplacer à choisir la meilleure action à exécuter pour atteindre son but.

Notre méthode consiste à apprendre et à reconnaître les balises naturelles d'un environnement intérieur structuré en utilisant des modèles de Markov cachés. Ces balises naturelles possèdent des caractéristiques qu'un robot peut reconnaître à l'aide de ces capteurs. Ces caractéristiques sont basées sur les variations des capteurs dans le temps. On peut en effet caractériser un grand nombre de balises naturelles d'un environnement intérieur structuré par une suite de variations sur un ensemble de capteurs. Par exemple, lorsqu'un robot se déplace dans un couloir en longeant un des deux murs constituant ce couloir, si nous observons le capteur perpendiculaire au mur, lorsque le robot passe devant une intersection, on remarque que le début de l'intersection se caractérise par une brusque augmentation de la mesure de ce capteur, puis par peu ou pas de variation sur le capteur perpendiculaire au mur, pour se finir par une brusque diminution de ce capteur. Il peut être intéressant d'apprendre cette suite de caractéristiques afin de pouvoir les reconnaître. Une des principales propriétés des modèles de Markov cachés est leur capacité à apprendre la suite de caractéristiques d'un signal. Notre but dans cette thèse est d'utiliser des modèles de Markov cachés pour apprendre et reconnaître les caractéristiques des balises naturelles d'un environnement intérieur structuré.

Nous avons appliqué notre approche pour réaliser deux modules d'une architecture de contrôle.

- Un module de localisation qui permet à partir d'une balise reconnue d'interpréter à quelle balise de l'environnement elle correspond et connaissant la position de cette balise de trouver la position du robot.
- Un module de contrôle d'exécution d'actions constituant la mission, qui permet à chaque instant en fonction de l'action à exécuter et du contexte (càd, l'état de l'environnement déterminé grâce aux balises reconnues) de choisir la meilleure façon d'exécuter cette action.

Ces deux modules ont été intégrés dans une architecture de contrôle que nous avons développée. Cette architecture comporte deux niveaux (un niveau pour la planification des missions et l'autre pour leur exécution). Elle comprend les diverses facultés dont doit disposer un robot mobile autonome et répond aux besoins et exigences que doit vérifier une bonne architecture de contrôle.

1.3 Plan du mémoire

Notre mémoire se décompose en trois parties :

- La première partie traite de la conception et de la réalisation de notre architecture de contrôle, ainsi que de la description de l'ensemble des modules qui la constituent, en particulier nous détaillons notre module de navigation dont le rôle est primordial dans la conception de notre approche. Cette partie est composée de cinq chapitres.
  - Le premier chapitre constitue l'introduction de la première partie.
  - Dans le deuxième chapitre, nous étudions les principales approches existantes pour la conception d'une architecture de contrôle. Ces approches sont généralement classées en trois catégories : les architectures hiérarchiques, les architectures réactives et les architectures hybrides. Pour chaque type d'architecture, nous analysons ses avantages et inconvénients, et présentons quelques exemples. Cette classification repose sur deux critères importants : la décomposition du contrôle en niveaux et l'autonomie. Nous montrons comment ces deux critères sont pris en compte dans les architectures de contrôle. Pour terminer, nous examinons les différentes composantes des architectures présentées afin de définir les modules (càd, leur rôle et leur fonctionnement) que nous avons retenus pour la mise en place de notre architecture de contrôle.
  - A partir de ces approches, dans le troisième chapitre, nous choisissons un modèle d'architecture le plus générique possible. Notre modèle est composé de

niveaux. Les niveaux ont une structure identique, des modules identiques, et des modules spécifiques. Nous présentons ensuite les deux niveaux (instances du modèle) de notre architecture, à savoir un niveau cognitif qui regroupe les facultés cognitives, et le niveau physique qui regroupe les facultés physiques. Pour finir, nous détaillons la composition et le fonctionnement des deux niveaux.

  - Les deux derniers chapitres de la première partie sont consacrés au module de navigation, composante primordiale du niveau physique, qui permet au robot de se déplacer dans son environnement. De plus, il va jouer un rôle primordial dans notre approche. Afin de pouvoir étudier les variations des mesures des capteurs dans le temps, il faut que les capteurs observent toujours la même partie de l'environnement. Cette contrainte doit être intégrée par le module de navigation lors de sa conception.

    Dans le quatrième chapitre de cette partie, nous présentons tout d'abord les trois techniques existantes pour réaliser un tel module : les techniques à base de champ de potentiel, les techniques à base de réseau de neurones et le contrôle flou. Cette étude a pour but de définir notre module de navigation.
  - Dans le dernier chapitre, nous détaillons notre module de navigation qui repose sur un contrôleur flou à deux niveaux. Dans le premier niveau, les capteurs sont regroupés afin de constituer des zones et à chaque zone est associée un comportement (càd, un ensemble de règles floues) d'évitement d'obstacle. Dans le deuxième niveau, un mécanisme intègre l'ensemble des comportements et définit le comportement général du robot.
- La deuxième partie décrit notre approche et son utilisation pour réaliser un module de localisation et un module de contrôle d'exécution. Cette partie est aussi composée de cinq chapitres.
  - Le premier chapitre constitue l'introduction de la deuxième partie.
  - Dans le deuxième chapitre, nous présentons les deux principales approches existantes pour l'apprentissage et la reconnaissance des balises naturelles : l'une cherchant à expliquer les données des capteurs ou à construire une représentation symbolique des balises et l'autre cherchant à déduire des propriétés statistiques intéressantes des balises. D'autre part, nous examinons les critères qui interviennent dans la façon d'apprendre et de reconnaître des balises naturelles : le nombre d'observations utilisées, le nombre de capteurs utilisés et l'interprétation symbolique des observations.
  - Dans le troisième chapitre, nous présentons les modèles de Markov cachés. Nous présentons tout d'abord les différents types de modèles de Markov cachés : les modèles d'ordre un et deux, ainsi que leurs particularités et les applications auxquelles ils sont destinés. Nous détaillons ensuite l'algorithme de Viterbi qui à partir d'une séquence de données des capteurs permet de reconnaître à quelle modèle d'ordre un de balise elle correspond. Puis nous citons les différents critères d'apprentissage couramment utilisés pour l'apprentissage des modèles, et détaillons l'algorithme de Baum-Welch qui à partir d'un corpus d'apprentissage construit un modèle de Markov caché d'ordre un correspondant à ce corpus.

Cette première partie de chapitre est illustrée par un exemple d'application des modèles de Markov cachés à l'apprentissage et à la reconnaissance de balises. La dernière partie de ce chapitre est consacrée à l'extension des algorithmes de reconnaissance et d'apprentissage à l'ordre deux. Pour réaliser notre approche, nous utilisons un outil basé sur des modèles d'ordre deux.

  - Le quatrième chapitre décrit comment nous appliquons les modèles de Markov cachés à l'apprentissage et à la reconnaissance de balises naturelles dans un cadre réel. Nous expliquons tout d'abord comment nous construisons notre corpus d'apprentissage, et comment nous le segmentons et l'étiquetons. Nous détaillons ensuite comment nous réalisons l'apprentissage des modèles de balise naturelle. Pour terminer nous définissons une méthode pour évaluer le taux de reconnaissance, et à partir des premiers résultats, nous modifions la façon de réaliser l'apprentissage pour améliorer le taux de reconnaissance.
  - Dans le cinquième chapitre, nous appliquons notre approche pour définir un module de localisation et un module de contrôle d'exécution d'actions. Nous expliquons tout d'abord comment nous avons modifié l'algorithme de reconnaissance afin que le robot soit capable de reconnaître les balises au fur et à mesure qu'il les voit, et comment nous interprétons chaque balise reconnue afin de déterminer à quelle balise de l'environnement elle correspond. Puis nous détaillons le fonctionnement de notre module de localisation qui à partir de cette interprétation des balises reconnues et de la position de ces balises dans l'environnement détermine la position du robot dans l'environnement. Pour terminer, nous expliquons le fonctionnement de notre module de contrôle d'exécution d'une action de haut niveau qui en fonction de la position du robot (délivrée par le module de localisation) et du but à atteindre va choisir la meilleure action motrice à réaliser.
- La troisième partie est consacrée aux conclusions et perspectives de notre travail.

Notre approche a été validée sur un robot mobile Nomad200, et de nombreuses illustrations de nos travaux se trouvent dans le mémoire.

Première partie

Notre architecture de contrôle

2

Introduction

Les travaux présentés dans cette partie concernent l'organisation et l'intégration de l'ensemble des fonctions dont doit disposer un robot mobile autonome afin d'appréhender l'environnement et d'agir selon la tâche qui lui a été assignée. Il s'agit de fonctions opératoires telles que les fonctions de perception, de commande et de modélisation. D'autre part, la conception d'un robot mobile autonome, qui soit capable d'accomplir une variété de tâches dans un environnement mal connu et évolutif, nécessite également des capacités décisionnelles qui vont lui permettre d'interpréter la mission et de déterminer les actions adéquates selon le contexte. Ainsi les fonctions opératoires sont au service des fonctions décisionnelles et toutes ces fonctions doivent s'intégrer dans une structure comportant différents niveaux organisés et communicants entre eux : l'architecture de contrôle du robot.

Dans le second chapitre de cette partie, nous présentons les différentes catégories d'architectures communément admises pour le contrôle d'un robot mobile :

– les architectures hiérarchiques qui reposent sur la décomposition du contrôle en différents niveaux.
– les architectures purement réactives qui reposent sur la définition de réflexes.
– les architectures hybrides qui utilisent les avantages des deux types d'architecture précédentes tout en essayant d'éliminer leurs lacunes respectives.

Dans la deuxième partie de ce chapitre, nous analysons en nous appuyant sur des exemples précis, les critères retenus pour la définition d'une architecture de contrôle. Nous examinons plus précisément :

– comment le contrôle du robot est décomposé en niveaux.
– comment l'autonomie est intégrée dans un système de conduite.

Pour terminer, nous analysons quelles sont les composantes nécessaires pour la conception d'un tel système, ainsi que la façon dont elles s'organisent. Cette analyse nous permet de dégager l'organisation et les composantes de notre architecture.

Dans le troisième chapitre de cette partie, nous présentons, après une description rapide du robot mobile sur lequel nous avons mené nos expérimentations, l'architecture de contrôle que nous avons définie. Ce travail s'est réalisé en collaboration avec Philippe Morignot qui était en visite dans notre équipe pour une période de six mois. Philippe ayant contribué au développement de l'architecture AIBOT alors qu'il était à Stanford dans l'équipe de Barbara Hayes Roth, notre propre architecture dérive de celle-ci. Après une analyse de nos besoins et une analyse critique de cette architecture AIBOT nous proposons une nouvelle approche qui retient une décomposition en deux niveaux cognitifs et physiques des traitements. Notre principal apport réside dans une simplification importante des mécanismes de coordination et de pilotage des différentes composantes de l'architecture. Cette architecture a fait l'objet d'une application pratique. Elle coordonne l'ensemble des modules que nous avons été amenés à développer tout au long de notre travail. L'application que nous avons développée permet d'illustrer le fonctionnement de cette architecture.

Dans les deux derniers chapitres de cette partie, nous nous intéressons au module de navigation, composante essentielle du niveau physique. En effet, pour pouvoir réaliser des missions dans son environnement, notre robot mobile doit être capable de s'y déplacer. Le rôle du module de navigation est de déterminer à chaque instant en fonction du but à atteindre et à partir des données imprécises et bruitées des capteurs la meilleure action motrice.

Le quatrième chapitre décrit les trois techniques habituellement utilisées pour réaliser un module de navigation réactive :

- Les approches à base de champ de potentiel. Dans ces approches, les objectifs de navigation (atteindre le but, tout en évitant les obstacles) sont codés dans une fonction.
- Les approches à base de réseau de neurones. Dans ces approches, le système de navigation doit apprendre lui-même comment éviter les obstacles et atteindre un but prédéfini.
- Le contrôle flou. Dans ces approches, on définit la manière de contrôler le robot, sous forme de règles.

Chaque approche est illustrée par des exemples d'application en robotique mobile.

Cette description nous permet de mettre en avant les critères pour réaliser un module de navigation réactive et de motiver notre approche.

Le dernier chapitre est consacré à la mise en place de notre module de navigation. Notre approche est basée sur le contrôle flou qui est le type d'approche que nous avons retenu. Partant du principe d'émergence communément admis dans les systèmes multi-agents, nous pensons que le comportement global d'un module de navigation réactive émerge de la combinaison de comportements locaux. Pour définir notre contrôleur flou, nous utilisons les capteurs comme entrée, et nous générons une action motrice en deux étapes :

- Premièrement, nous définissons l'ensemble des comportements locaux. Pour cela, nous regroupons les capteurs du robot afin de former des zones et définissons des règles de comportement pour ces zones.
- Deuxièmement, nous utilisons un opérateur de composition qui permet d'élaborer un comportement global construit à partir des comportements locaux.

Nous décrivons aussi comment notre module de navigation réactive est intégré dans le niveau physique.

3

Les architectures de contrôle

3.1 Introduction

Dans ce chapitre, nous présentons les différentes catégories d'architectures communément admises pour le contrôle d'un robot mobile, ainsi que leurs qualités et défauts :

- les architectures hiérarchiques qui reposent sur la décomposition du contrôle en différents niveaux.
- les architectures purement réactives qui reposent sur la définition de réflexes.
- les architectures hybrides qui utilisent les avantages des deux types d'architectures précédentes tout en essayant d'éliminer leurs lacunes respectives.

De plus, nous citons pour chaque type d'architecture des exemples précis.

Cette classification repose en fait sur deux critères : la décomposition du contrôle en niveaux, et l'intégration de l'autonomie. En analysant les exemples, nous montrons dans la deuxième partie de ce chapitre, que la décomposition du contrôle en deux niveaux (un niveau décisionnel planifiant et décomposant les missions du robot et un niveau fonctionnel les exécutant) est une bonne solution. D'autre part, nous analysons les diverses situations auxquelles le robot doit réagir afin d'être complètement autonome. Ces situations vont du simple obstacle à éviter sur le chemin du robot jusqu'à la remise en cause complète d'une mission suite à un changement du contexte d'exécution de cette mission, en passant par la révision de l'action [1] en cours d'exécution. Cette analyse nous permet de mettre en évidence les facultés nécessaires à l'autonomie d'un robot mobile.

Dans la dernière partie de ce chapitre, nous montrons aussi que la conception d'une architecture de contrôle peut être vue de deux manières différentes :

- Soit au moment de la planification des actions, nous fixons complètement la façon dont les actions seront exécutées, et pendant l'exécution des actions nous donnons au robot les moyens de vérifier que les actions sont bien exécutées, tout en définissant des mécanismes d'exception qui permettent de gérer les imprévus. C'est par exemple le cas pour la gestion des déplacements, le planificateur fournit au robot une trajectoire à suivre, et pour se déplacer le long de cette trajectoire le robot dispose d'un ensemble d'asservissements qui lui permettent de la suivre. Il dispose d'autre part de mécanismes afin de détecter les imprévus et d'y faire face.

1. Une action est définie comme une suite de mouvements.

- Soit au moment de la planification, nous nous contentons de fixer des buts à atteindre et pendant l'exécution, nous laissons le robot libre de choisir la manière d'exécuter ces actions.

En partant du principe que la deuxième solution paraît plus intéressante puisqu'elle permet au robot d'être plus libre lors de l'exécution de ses missions[2], nous définissons à travers les exemples les composantes nécessaires à la conception d'une architecture de contrôle :

- Un planificateur de missions qui se trouve au niveau décisionnel, et qui permet au robot de planifier ses missions et de les replanifier le cas échéant.
- Au niveau non décisionnel :
  - Un contrôleur d'exécution d'actions qui décide en fonction du contexte et de l'action à réaliser quels modules doivent être activés.
  - Un planificateur de chemins qui se trouve au niveau fonctionnel et définit à partir d'un but à atteindre, des buts intermédiaires à atteindre.
  - Un navigateur qui permet au robot de se placer face à son but, de se déplacer vers son but tout en évitant les obstacles imprévus.
  - Un localisateur qui permet au robot de connaître sa position dans son environnement.

Cette analyse nous permet de dégager l'organisation et les composantes de notre architecture.

3.2 Les différents types d'architectures

3.2.1 Introduction

Dans cette section, nous présentons les trois types d'architectures communément admis pour le contrôle d'un robot mobile. Nous commençons par les architectures hiérarchiques qui sont les premières à avoir été utilisées, dont le principe repose sur la décomposition du contrôle en niveaux. Puis, nous présentons les architectures réactives qui ont vu le jour au milieu des années quatre-vingts, dont le fonctionnement est gouverné par la mise en place de réactions pour affronter les diverses situations auxquelles un robot mobile est confrontées lorsqu'il évolue dans son environnement. Pour terminer, nous nous intéressons aux architectures qui sont actuellement généralement utilisées pour contrôler un robot mobile : les architectures hybrides. Ces architectures hybrides cherchent à la fois à décomposer le contrôle en couches et à rendre chaque couche autonome de façon à ce qu'elles puissent faire face aux situations pour lesquelles elles ont été conçues.

Pour chaque type d'architecture, nous analysons son principe de fonctionnement ainsi que ses points forts et faibles. De plus, nous présentons pour chaque type, un ou deux exemples :

- Pour les architectures hiérarchiques, l'architecture du robot SHAKEY [Nilsson, 1969] qui constitue une des premières architectures réalisées pour le contrôle d'un

2. Une mission est définie comme un travail à réaliser de manière autonome par le robot dans son environnement

robot mobile, et l'architecture NASREM de [Albus et Lumia, 1987] proposée par la NASA.
- L'architecture subsomption de [Brooks, 1986] qui est une référence pour les architectures réactives.
- Deux architectures hybrides actuelles : celle du robot mobile HILARE de l'équipe RIA du LAAS, et celle du véhicule autonome du projet SHARP de l'INRIA Rhône-Alpes.

3.2.2 Les Architectures Hiérarchiques

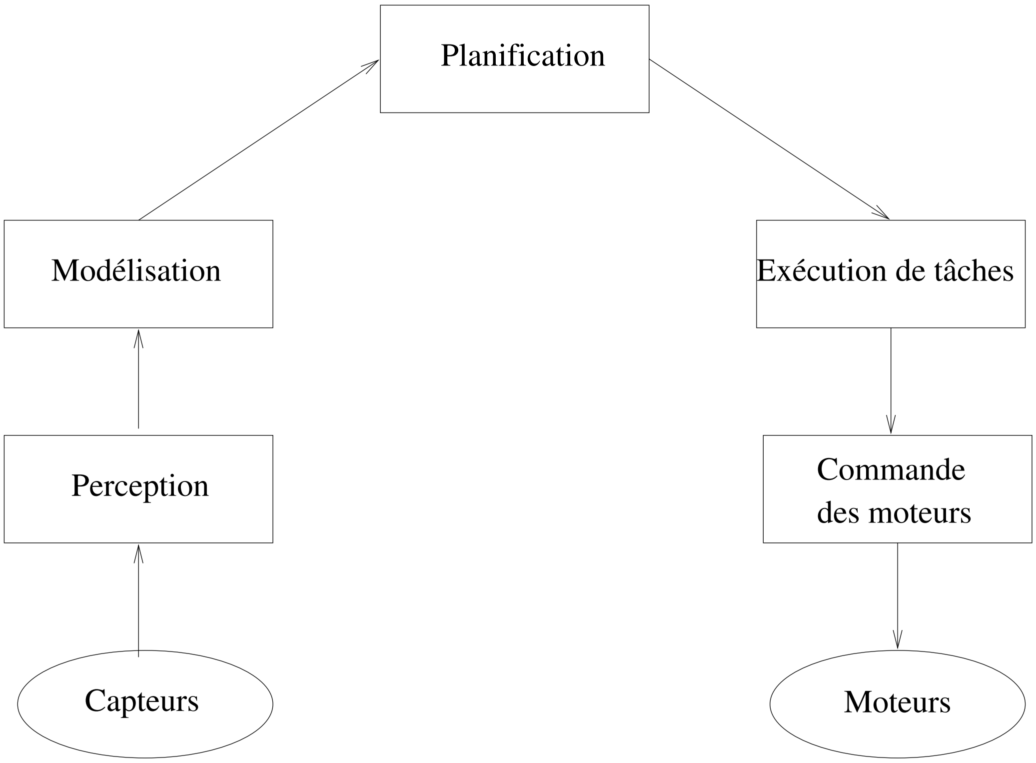


FIG. 3.1 – *Les architectures hiérarchiques*

Ces architectures sont le résultat des premières recherches en robotique mobile. Elles sont fondées sur l'application des techniques d'Intelligence Artificielle (telles que la planification de tâches [3], ou la représentation des connaissances) au contrôle d'un robot mobile.

Dans une telle architecture, on cherche tout d'abord à modéliser les connaissances et les méthodes d'inférence associées permettant d'élaborer les actions que le robot doit entreprendre. Le principe de fonctionnement de ce type d'architecture peut être résumé par la figure 3.1. Le robot commence par traiter les données recueillies par ses capteurs. Ensuite, il identifie les objets qui sont dans son environnement proche. Puis il construit une représentation de la scène dans son ensemble, et l'utilise pour générer un plan. Après

3. Par la suite, nous considérons qu'une tâche est une mission ou une action.

quoi, il calcule au mieux une séquence de commandes vers les effecteurs pour exécuter le plan prévu.

Points forts :

- l'atout essentiel de ces architectures réside dans la possibilité d'intégrer des raisonnements de haut niveau (niveau mission, planification) qui s'appuient sur des modèles assez complets (carte par exemple) de l'environnement dans lequel évolue le robot. Cet aspect est, comme nous le verrons plus loin, absent des architectures purement réactives.

Points faibles :

- un cycle de traitement s'étend de l'acquisition des données capteurs jusqu'à l'envoi de commandes sur les effecteurs en traversant toutes les couches de l'architecture. On observe ainsi un manque de réactivité (prise en compte et réponse en temps réel) avec l'environnement dans lequel évolue le robot. Ces sytèmes, même implantés sur super calculateurs, sont de ce fait relativement lents pour un observateur extérieur (notamment pour maintenir un modèle de l'environnement).
- ces systèmes sont peu adaptatifs dans la mesure où il leur est difficile de réagir à des situations différentes de celles qu'ils connaissent. Cela exprime une lacune dans la possibilité de généraliser des situations possibles.
- ces sytèmes sont généralement peu robustes de par leur modèle centralisé. Une défaillance d'un module peut provoquer le blocage de toute l'architecture.

Exemples

Le robot SHAKEY

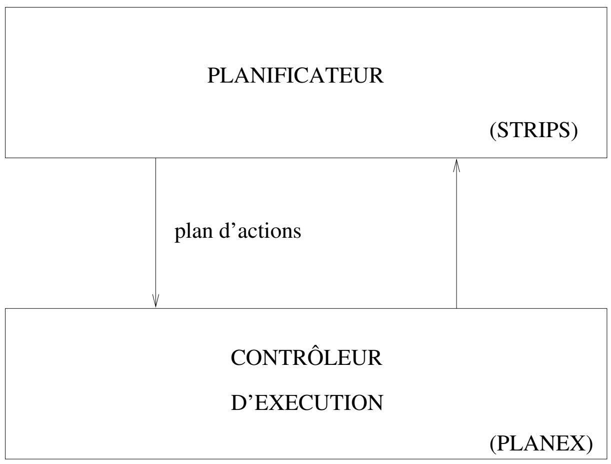


FIG. 3.2 – *L'architecture hiérarchique de SHAKEY*

Une des plus anciennes architectures hiérarchiques est celle utilisée pour contrôler le robot SHAKEY [Nilsson, 1969]. Le robot SHAKEY est un robot mobile à roues piloté par un ordinateur et doté d'une caméra TV, de capteurs télémétriques et d'une ceinture de capteurs tactiles. Il a pour mission de ranger des objets simples de l'environnement en les poussant. SHAKEY évolue dans un environnement qui lui est complètement dédié. Les obstacles sont statiques, et tous visibles par les capteurs.

Son architecture (figure 3.2) est composée de deux niveaux, un niveau de planification et un niveau contrôle d'exécution.

Le planificateur STRIPS [Fikes et Nilsson, 1971] est utilisé pour planifier les actions du robot SHAKEY. Le robot suit un plan généré par STRIPS en utilisant un contrôleur d'exécution nommé PLANEX.

PLANEX utilise le formalisme de tables triangulaires pour gérer le plan fourni par STRIPS. Il a la possibilité de réexécuter une même action ou une des actions suivantes du plan. Les cas d'échec correspondent à un état du monde qui n'est pas un état prévu par le plan. Dans ce cas STRIPS procède à une nouvelle planification (replanification) à partir du nouvel état du système.

L'architecture NASREM

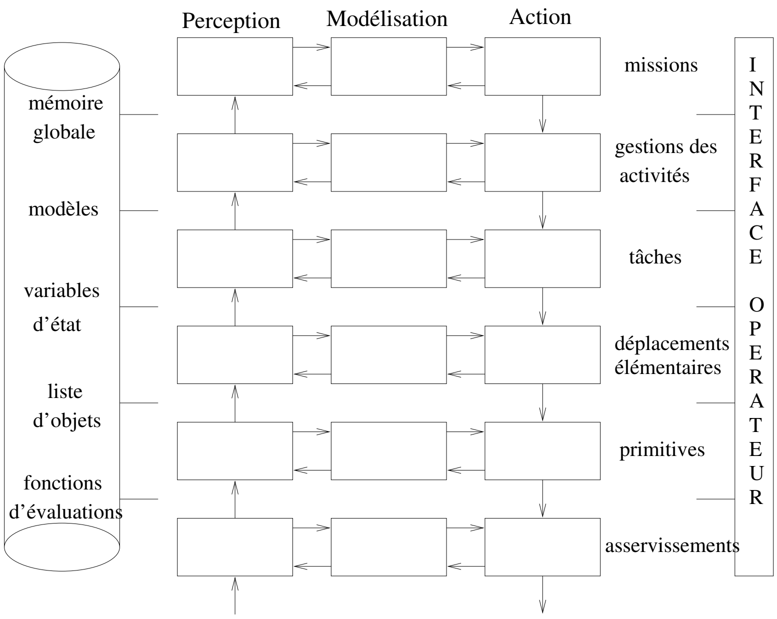


FIG. 3.3 – *L'architecture hiérarchique NASREM*

L'architecture NASREM [Albus et Lumia, 1987] a été proposée par la NASA comme un standard pour le contrôle. Elle généralise des modèles classiques d'architecture reposant sur une décomposition de l'ensemble du système de commande en couches hiérarchiques [Lumia et Wavering, 1990]. Dans cette décomposition hiérarchique, chaque tâche peut se décomposer en sous-tâches qui constituent le niveau immédiatement inférieur (figure 3.3). A chaque niveau, une transformation générique est effectuée sur l'objectif. Le principe de cette décomposition peut se voir comme une abstraction temporelle, chaque niveau traitant des données et des événements avec une fréquence plus faible que le niveau inférieur. Ainsi, aux niveaux les plus bas se trouvent les traitements rapides tels que les asservissements qui manipulent des données brutes ou peu affinées. Et aux niveaux supérieurs des processus lents de raisonnement, tel que le planificateur de tâches, utilisent des données affinées essentiellement symboliques. L'architecture complète comporte six couches qui sont elles-mêmes décomposées horizontalement en trois parties :

- La perception.
- La modélisation.
- La planification et l'action.

La partie "Planification/Action" est un planificateur de tâches à réaliser par le robot. Certaines des informations nécessaires à la réalisation de ces tâches sont fournies par la partie "Modélisation" qui correspond à l'estimation la plus robuste de l'état de l'environnement. La partie "Modélisation" se sert des données acquises et traitées par la partie "Perception". L'intérêt de cette décomposition en trois parties est de créer un découplage entre les informations nécessaires pour effectuer la décomposition en tâches et le mode d'acquisition et de traitements des données qui est spécifique au robot commandé. Le concept de mémoire globale permet à tout module de communiquer avec n'importe quel autre module du système. L'interaction entre l'opérateur humain et le système de commande s'effectue à travers des modules de l'interface opérateur.

Cette structuration très stricte a le mérite d'être claire mais sa rigidité en fait un système parfois difficile à adapter au problème considéré.

3.2.3 Les Architectures Réactives

Les architectures réactives ont pour but de concevoir et de réaliser un robot, capable d'apporter une réponse immédiate à toute nouvelle modification de l'environnement le concernant. Ces approches sont initialement fondées sur l'étude de l'interaction de l'animal avec son environnement naturel (l'éthologie) de manière à essayer de comprendre les mécanismes mis en oeuvre et de les reproduire. Elles sont fondées sur le fait que lorsqu'un être humain évolue dans un environnement, il ne passe pas son temps à modéliser son environnement, et à planifier ses actions, mais qu'au contraire il réagit à de simples stimuli qui lui permettent de se déplacer vers son but tout en faisant face aux obstacles. Ainsi, contrairement aux systèmes hiérarchiques, le comportement global du robot apparaît comme la résultante de plusieurs comportements actifs simultanément. Il n'y a pas de phase de planification ou de modélisation de l'environnement.

Points forts :

- la robustesse, du fait de la relative indépendance de chacun des comportements les uns par rapport aux autres, si l'un d'eux vient à avoir une défaillance, le robot est toujours capable de réaliser une mission, ce qui est très important lorsque l'on a à faire à un robot autonome.
- la rapidité de réponse, du fait que les comportements sont directement placés entre les capteurs et les effecteurs. De plus, comme en général, les comportements sont simples à câbler, on peut les réaliser en hardware pour une vitesse encore accrue.
- l'aspect incrémental en niveau : il suffit de mettre en place un niveau de compétence puis d'ajouter les niveaux supérieurs sans avoir, en théorie, à remettre en question les niveaux inférieurs. Cet aspect est primordial pour la mise au point et la recherche de pannes. Cette approche permet également d'accroître la complexité du compor-

tement global du robot sans pour autant accroître celle du contrôle global puisque celui-ci est complètement distribué.
– faible coût : un robot avec une telle architecture est d'un coût moindre, de part la conception et le test incrémental des niveaux, que celui d'un robot doté d'une architecture traditionnelle réalisant la même tâche.

Points faibles :

– Les architectures réactives souffrent d'une absence de capacité de modélisation et de raisonnement.

Exemple

L'architecture Subsomption de Brooks

	7	Raisonner sur les comportements des objets et modifier les plans en conséquence		
	6	Formuler et exécuter un plan qui implique un changement voulu de l'état du monde		
	5	Raisonner sur le monde en termes d'objets identifiables et de réalisation de tâches relativement à certains objets		
capteurs →	4	Remarquer les changements dans l'environnement statique	→	effecteurs
	3	Construire une carte de l'environnement et des routes d'un lieu à l'autre		
	2	Explorer le monde en repérant les endroits atteignables et chercher à les atteindre		
	1	Errer sans but		
	0	Eviter les objets mobiles ou non		

FIG. 3.4 – *La Hiérarchie de Comportements dans l'Architecture Subsomptive de Brooks*

La première et la plus célèbre des architectures réactives est l'architecture Subsomption proposée par [Brooks, 1986]. Elle peut être définie comme une hiérarchie de comportements (figure 3.4) plutôt que comme une hiérarchie fondée sur l'abstraction des données. Le principe est de faire coïncider à chaque comportement du robot un sous-système capteurs-actionneurs appelé couche. Tous les comportements s'exécutent en parallèle. Les comportements de haut niveau ont la possibilité d'inhiber temporairement des comportements de plus bas niveau. Quand le comportement de plus haut niveau n'est pas déclenché par les mesures des capteurs, il n'inhibe pas de comportements de plus bas niveau (figure 3.5). Un comportement de bas niveau prend donc le contrôle du robot. Ainsi, cette architecture est réellement parallèle, et les capteurs sont directement utilisés par toutes les couches comportementales. Il n'y a pas de modèle central et géométrique du monde.

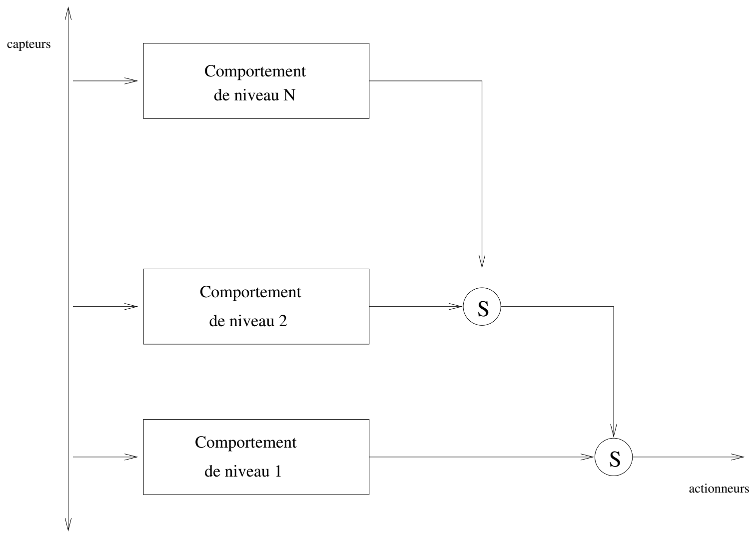


FIG. 3.5 – *Le Mécanisme d'Inhibition dans l'Architecture Subsomptive de Brooks*

3.2.4 Les Architectures Hybrides

Les approches hiérarchiques et réactives sont diamétralement opposées. Cependant, chacune présente des caractéristiques intéressantes. Pour cela, des chercheurs ont essayé de les combiner en mettant au point des architectures hybrides permettant notamment d'allier des capacités de raisonnement et de décision de haut niveau, s'appuyant sur des représentations abstraites des connaissances, avec des comportements réactifs garantissant robustesse et flexibilité. En effet, l'exécution de plans est basée sur un contrôle réactif (ce que détectent les capteurs permet de réagir immédiatement en cas de nécessité), alors que la formulation de plan(s) est assurée par un système hiérarchique pouvant reposer sur un modèle de l'environnement et des obstacles connus.

Points forts :

- Les avantages de la méthode résultent de la relative facilité d'ajouter des comportements nouveaux et de la possibilité d'effectuer différents traitements sur les données issues des capteurs pour satisfaire la demande des différents niveaux de l'architecture.

Points faibles :

- Le problème majeur de cette approche réside dans la détermination des agents experts et leur regroupement en sous-ensembles d'activation qui est totalement heuristique.

Exemples

L'architecture de contrôle du robot mobile HILARE de l'équipe RIA du LAAS/CNRS

L'architecture de contrôle du robot mobile HILARE de l'équipe Robotique et Intelligence Artificielle (figure 3.6) [Alami *et al.*, 1993] est une architecture hybride composée

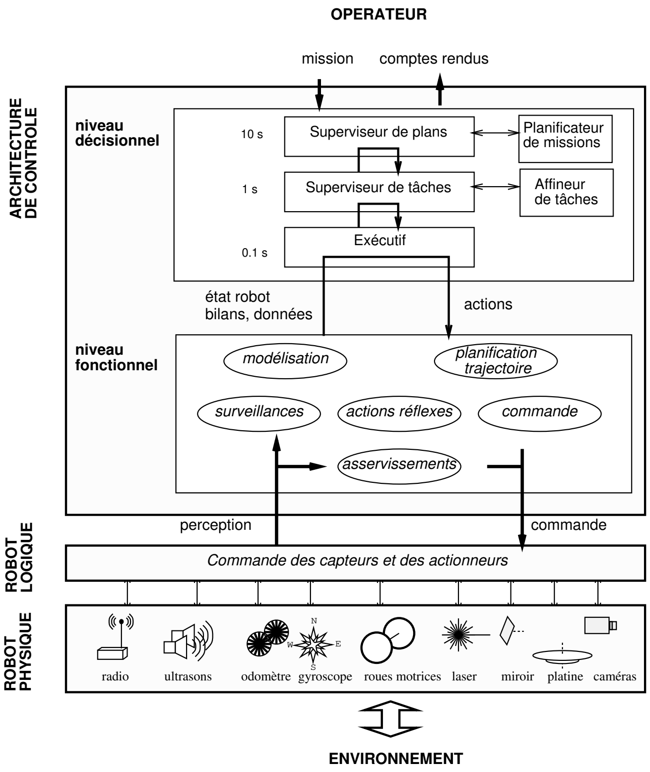


FIG. 3.6 – *L'architecture du robot mobile HILARE de l'équipe RIA du LAAS*

de quatre couches :

- Au plus bas se trouvent les capteurs et actionneurs du robot.
- Au dessus, une couche d'abstraction appelé robot logique, permet d'utiliser ces capteurs et actionneurs indépendamment du robot concerné et/ou des capteurs spécifiques utilisés.
- Ensuite, on trouve le coeur de l'architecture de contrôle [Chatila, 1994]. Ce dernier se décompose en deux parties bien distinctes, le niveau décisionnel et le niveau fonctionnel.

Le niveau décisionnel

Le niveau décisionnel joue un rôle essentiel dans l'architecture de contrôle d'HILARE. De façon générale, son rôle est de surveiller le bon déroulement des plans/actions qu'il a activés, tout en surveillant un certain nombre de conditions qui peuvent ou doivent donner lieu à des traitements particuliers. Son rôle est très central, puisque présent sur deux niveaux (figure 3.6), il interagit à la fois avec le niveau utilisateur, les planificateurs de missions et l'affineur de tâches, et l'exécutif qui est en prise directe avec les modules fonctionnels. Les raisonnements du niveau décisionnel sont, pour des raisons de réactivité, minimaux, prédictibles en temps et interruptibles.

Le niveau décisionnel assure :

- au niveau "supervision de plans":
  - l'interprétation des ordres donnés par l'utilisateur.

  - l'appel au planificateur de missions.
  - la transmission des plans produits au niveau inférieur.
  - la supervision et le contrôle de l'exécution de ces plans par le niveau fonctionnel.
- au niveau de la "supervision de tâches" où on retrouve le même type d'activité plus centré sur l'affinement de tâches:
  - l'interprétation des plans du niveau supérieur.
  - l'appel à l'affineur de tâches.
  - l'envoi des requêtes vers le niveau fonctionnel à travers l'exécutif.
  - l'analyse des résultats des requêtes et des événements asynchrones remontant de l'exécutif.

Le niveau fonctionnel

Les multiples actions que le robot est amené à exécuter pour accomplir une tâche sont assurées par le niveau fonctionnel qui apparaît au niveau décisionnel comme un ensemble de services. Le niveau fonctionnel regroupe en effet l'ensemble des fonctions opératoires qui permettent au robot de se déplacer, de percevoir et de modéliser son environnement, de calculer des trajectoires ou encore d'estimer sa position. C'est par son biais que le robot maintient une interaction permanente avec son environnement (asservissements et actions réflexes) et que des données relatives à cet environnement et à l'état du robot peuvent être remontées aux processus de raisonnement[Fleury, 1996].

Il se présente simultanément comme :

- Une interface avec le niveau décisionnel : les différents services doivent pouvoir être activés, désactivés, paramétrés et séquencés en temps réel par le niveau décisionnel en fonction de la tâche à accomplir, mais aussi de données dynamiques relatives à l'état du robot et à la perception de son environnement (ces données sont elles-mêmes rendues disponibles par la couche fonctionnelle). Cela suppose que les différents services, malgré leur diversité, soient programmables, observables, présentent des comportements cohérents et prédictibles (conflits, conditions de démarrage et d'arrêt, reprise d'erreurs), et offrent des modes d'interaction standard (contrôle et accès aux données).
- Une interface assurant une interaction en boucle fermée sur le monde réel : la dynamique du système (propre au robot mobile et induite par les interactions avec l'environnement), la mise en oeuvre de traitements synchrones et à temps d'exécution borné, la nécessaire réactivité face à des événements asynchrones, la parallélisation et la distribution des traitements, impliquent des capacités temps réel exigeantes.

L'architecture du projet SHARP de l'INRIA Rhônes/Alpes

L'architecture de contrôle du projet Sharp (Programmation Automatique et Systèmes Décisionnels en Robotique) de l'INRIA Rhône-Alpes est une architecture hiérarchique hybride (figure 3.7). Cette architecture, d'un point de vue fonctionnel, a déjà été utilisée dans le cadre du projet PROMETHEUS [Hassoun, 1994]. Elle se décompose verticalement en modules de haut niveau (missions, planification) et de bas niveau (contrôle d'exécution,

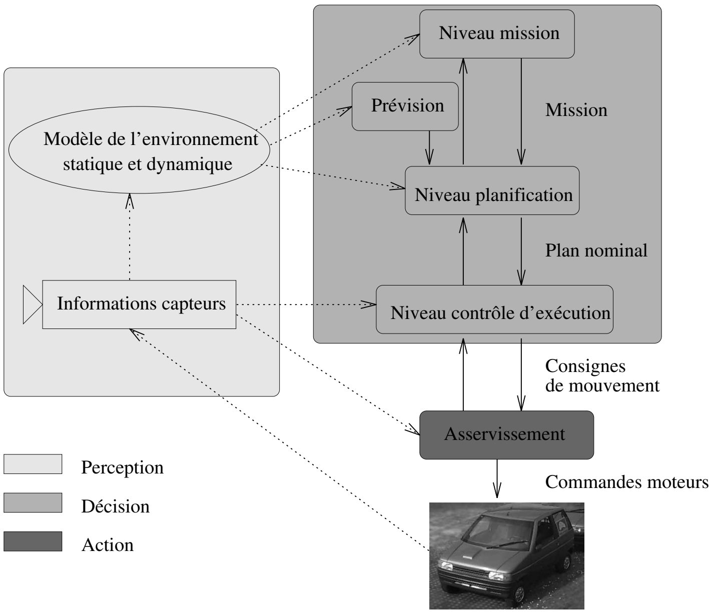


FIG. 3.7 – *L'architecture de contrôle de Sharp (Lifia/INRIA Rhône-Alpes)*

asservissements). Les modules de haut niveau sont des modules délibératifs s'appuyant sur une modélisation de l'environnement statique et dynamique; les modules de bas niveau utilisent directement les informations issues des capteurs.

Fonctionnement

Le niveau mission détermine des buts que doit atteindre le robot. A partir de ces buts, le niveau planification génère une trajectoire en prenant en compte l'environnement statique (obstacles statiques, voies de circulation, ...) ainsi que l'environnement dynamique (obstacles mobiles, ...). Pour cela, il utilise l'état instantané de l'environnement mais recourt également à une prévision de l'évolution de celui-ci sur un certain horizon temporel T. La trajectoire ainsi calculée est fournie à un contrôleur d'exécution chargé de la suivre en prenant en compte de manière réactive des modifications de l'environnement depuis l'étape de planification (par exemple obstacle non détecté ou dont le comportement ne correspond pas à celui qui a été prédit). Le contrôleur génère enfin un ensemble de consignes (commandes) au module d'asservissement du robot. Nous allons maintenant porter notre attention sur les modules de planification et de contrôle d'exécution.

Le planificateur de mouvements

Les contraintes auxquelles est soumis le robot sont de différentes natures : contraintes cinématiques et dynamiques du robot, contraintes de non-collision avec l'environnement statique et dynamique, optimisation de critères comme la distance parcourue, le temps d'exécution d'une manœuvre, ... Dans le but de réduire la complexité liée à une prise en compte globale de ces contraintes, le planificateur de mouvements [Fraichard, 1992] est composé de deux modules complémentaires :

- un “planificateur de chemins” intégrant les contraintes géométriques et cinématiques. Ce planificateur permet de trouver un chemin géométrique sans manœuvre dans un environnement statique structuré (composé de voies de circulation rectilignes pouvant s’intersecter). Ce chemin assure l’évitement d’obstacles statiques tout en intégrant les contraintes cinématiques du robot (rayon de giration borné par exemple).
- un “planificateur de trajectoires” permettant la prise en compte de l’aspect dynamique. La fonction du planificateur de trajectoires consiste donc à suivre un chemin en un temps minimal, de manière à éviter les obstacles mobiles non pris en compte jusque là et ce, en présence de contraintes dynamiques. L’évitement des obstacles sur le chemin nominal est rendu possible grâce à la notion de chemins adjacents : à tout moment, le robot peut ainsi passer d’un chemin à un autre qui lui est adjacent.

Le contrôleur d’exécution de mouvements

Le rôle du contrôleur d’exécution de mouvements est d’assurer et contrôler de manière réactive la réalisation du plan délivré par le planificateur de mouvements. En effet, celui-ci doit, dans un premier temps, envoyer aux modules d’asservissements des consignes permettant de suivre le plan nominal. Cependant, afin de prendre en compte les modifications de l’environnement dynamique depuis la phase de planification, le contrôleur est capable d’amender localement le plan nominal à travers des manœuvres d’évitement et de rattrapage de la trajectoire nominale.

3.2.5 Conclusion

La définition des différents types d’architecture s’est faite au fil du temps, et s’est faite grâce à l’analyse des points faibles de chaque type d’architecture. En effet, les premières architectures hiérarchiques résulte de l’application directe des techniques d’Intelligence Artificielle telles que la planification. [Brooks, 1986], partant du principe que le robot évolue dans un environnement dynamique, propose une approche radicalement opposée en avançant l’idée que la définition de comportements réactifs permet à elle seule de contrôler un robot mobile. Progressivement, on s’est rendu compte qu’une bonne architecture de contrôle doit à la fois permettre la décomposition du contrôle en niveaux, et intégrer des comportements réactifs. Actuellement, les architectures hiérarchiques ont quasiment disparu, et les architectures purement réactives sont peu nombreuses [Maes, 1993]. La grande majorité des architectures sont hybrides. C’est donc vers ce type d’architecture que nous nous orienterons.

3.3 Les critères retenus pour la définition d'une architecture de contrôle

3.3.1 Introduction

La classification des architectures de contrôle repose essentiellement sur deux critères : la décomposition du contrôle et l'autonomie. Au delà des différents types d'architectures, nous analysons dans cette section comment sont pris en compte ces critères dans la définition d'une architecture de contrôle. Dans le premier paragraphe de cette section, nous montrons à partir des exemples présentés pourquoi la décomposition du contrôle en deux niveaux paraît la meilleure. Dans le deuxième paragraphe de cette section, nous analysons les types de situations auxquels le robot doit pouvoir réagir de manière autonome. Cette réactivité va de la simple prise en compte d'une modification de l'environnement qui va obliger notre robot à modifier sa trajectoire initiale pour éviter un obstacle à la modification complète du contexte d'exécution de la mission assignée au robot qui dans ce cas va l'obliger à replanifier sa mission. Il faut donc pour définir une bonne architecture de contrôle tenir compte des différents niveaux de réactivité.

3.3.2 La décomposition du contrôle

L'élaboration d'une architecture de contrôle consiste à trouver les traitements nécessaires permettant, à partir des entrées fournies par les capteurs et de la mission à réaliser, de fournir les actions aux moteurs. Nous avons vu que ce traitement fait appel à des compétences diverses et nombreuses. Pour être efficaces, ces diverses compétences doivent être regroupées en fonction des données qu'elles manipulent et des traitements qu'elles effectuent. Ces regroupements amènent tout naturellement à une décomposition du contrôle.

Cette décomposition en niveaux est à la base des architectures de contrôle pour robot mobile. Elle est le seul critère retenu pour la définition des premières architectures hiérarchiques. On la retrouve dans tous les types d'architectures.

Le nombre de niveaux peut être très variable. Dans son architecture, [Nilsson, 1969] utilise deux niveaux, un niveau planifiant les missions du robot, et l'autre niveau les exécutant. On retrouve aussi ces deux niveaux dans l'architecture du projet SHARP, un niveau décisionnel composé d'un planificateur de mission et de chemins qui comprend aussi un module de contrôle d'exécution et un niveau fonctionnel qui en fonction des actions à exécuter définit les asservissements à réaliser par le robot. Bien que possédant quatre couches, on retrouve dans la couche principale de contrôle d'HILARE, deux niveaux (un niveau décisionnel et un niveau fonctionnel) qui ont les mêmes fonctions que les deux niveaux de SHAKEY.

Cette décomposition en deux niveaux paraît naturelle puisque pour contrôler un robot mobile, on a besoin de planifier ses missions puis de les exécuter. Le niveau supérieur manipule des données symboliques et génère des actions peu détaillées. Le niveau inférieur traite des données numériques bruitées et fournit aux actionneurs des commandes très simples.

3.3.3 L'autonomie du robot

Comme nous l'avons déjà dit, un robot mobile évoluant dans un environnement non spécifiquement conçu pour lui, doit être doté d'une certaine autonomie. En effet, les situations possibles dans un tel environnement sont tellement nombreuses qu'il est impossible de les prévoir toutes. Pour faire face aux modifications de son environnement, un robot mobile doit être capable de réagir à un nombre défini de situations. Cette réactivité se retrouve à chaque niveau de l'architecture de contrôle [Giralt, 1993] :

- Une réactivité réflexe qui permet au robot de faire face aux modifications directes de son environnement, par exemple lorsqu'un obstacle imprévu se trouve sur la trajectoire du robot. La réactivité réflexe a pour but de calculer les actions moteurs en fonction des mesures des capteurs.
- Une réactivité stratégique qui lui permet de faire face aux modifications importantes de son environnement et qui ont une incidence directe sur la mission à réaliser. Lorsque le robot ne peut pas exécuter une action qui lui a été assignée, il doit être capable d'intégrer cette contrainte dans sa mission et de trouver un nouveau plan qui lui permettra de réaliser sa mission.
- Entre ces deux extrémités, on trouve généralement un niveau intermédiaire de réactivité et de prise de décision : la réactivité tactique [Radford et Fang, 1993]. Ce niveau est en général chargé de déterminer l'action immédiate du robot en fonction des informations limitées et incertaines provenant des capteurs. Il se caractérise donc à la fois par une prise de décision temps réel et la gestion d'une représentation du contexte de la tâche ou du comportement induit du robot.

Dans une architecture comme celle de SHAKEY, le contrôle du robot est complètement centralisé, et est assuré par le planificateur qui en incluant les données fournies par le robot calcule à chaque instant la meilleure action à réaliser. Cette méthode permet à SHAKEY d'être autonome dans la planification de ses missions, mais il l'est assez peu au niveau de leur exécution. En effet, lorsqu'un obstacle imprévu apparaît, SHAKEY ne peut pas déclencher un simple évitement d'obstacles, mais il doit transmettre cet imprévu au planificateur qui, après analyse de la situation, va lui fournir la suite d'actions, lui permettant de l'éviter. SHAKEY ne dispose que d'une réactivité stratégique, puisqu'il adapte à chaque instant sa stratégie en fonction de l'état de l'environnement.

Contrairement à cela, l'architecture NASREM, qui est aussi hiérarchique, possède les trois niveaux de réactivité. Ceci tient à l'utilisation de modules semblables à chaque niveau. En effet, chaque niveau traite les données qu'il reçoit du niveau directement inférieur, en fonction de ses besoins, ce qui lui permet de réagir aux imprévus. Bien qu'elle fasse partie de la même catégorie d'architecture que celle du robot SHAKEY, l'architecture NASREM permet au robot d'exécuter beaucoup plus librement les actions qui lui sont fournies par son planificateur.

Il paraît évident que l'architecture subsomption possède une réactivité réflexe qui lui permet de faire face aux changements simples de l'environnement. En effet, les deux premiers niveaux de cette architecture assurent les réactions simples à l'environnement ne mettant pas en oeuvre de traitements des données capteurs. Il paraît par contre plus difficile de voir comment sans prétraitement des données, des compétences des derniers niveaux peuvent fonctionner et donc prendre en compte la réactivité stratégique et tactique.

L'organisation de l'architecture développée pour le véhicule autonome du projet SHARP gére les trois niveaux de réactivité. Le planificateur de mission et le planificateur de chemins assurent la réactivité stratégique. Ils reçoivent des compte-rendus d'exécution que renvoit le contrôleur d'exécution et les intégrent afin de vérifier le bon déroulement de la mission, ce qui peut nécessiter le cas échéant une replanification. Le contrôleur d'exécution gère la réactivité tactique puisque son rôle est de sélectionner l'action en fonction du contexte. La réactivité réflexe est prise en compte grâce à des mécanismes d'exceptions qui permettent lorsqu'un obstacle se trouve sur la trajectoire initiale de passer sur une trajectoire adjacente. On retrouve quasiment le même type d'organisation dans l'architecture d'HILARE. En effet, au niveau décisionnel, l'affineur de tâches transforme les missions en procédures composées d'actions élémentaires, et supervise l'exécution de ces procédures tout en étant réactif aux événements asynchrones (réactivité stratégique). Le niveau fonctionnel est supervisé par un contrôleur central d'exécution (réactivité tactique). Au niveau fonctionnel, on trouve un ensemble de services qui permettent au robot d'exécuter les actions (réactivité réflexe).

3.3.4 Conclusion

Cette section nous a permis de voir comment décomposer le contrôle d'un robot en niveaux, et d'examiner les types de situation auxquels un robot doit pouvoir réagir. De cette analyse, nous retenons qu'une bonne architecture doit comporter deux niveaux : un niveau cognitif qui décompose et gère les missions, et un niveau physique qui les exécute.

D'autre part, si l'on veut réaliser un robot autonome, il faut lui donner les moyens de réagir aux événements imprévus tant au niveau de la planification des missions que de leurs exécutions. Il faut pour cela que les actions soient spécifiées très sommairement, et que le robot soit muni de comportements réactifs lui permettant de comprendre ces actions et de les exécuter en étant complètement autonome dans la façon de gérer cette exécution. Il doit d'autre part être capable de détecter les situations imprévues et d'analyser celles qu'il peut résoudre facilement et celles qu'il doit soumettre au planificateur de missions. Le planificateur doit pouvoir intégrer ces imprévus et y répondre par une nouvelle suite d'actions, et dans le cas contraire, soit abandonner sa mission, soit faire appel à l'utilisateur pour l'aider. Pour cela, le robot doit posséder trois types de réactivité pour faire face aux diverses situations qu'il rencontre dans son environnement. La réactivité stratégique qui doit être gérée au niveau cognitif, puisque c'est à cet endroit que se planifie les missions. La réactivité tactique qui doit être prise en compte au plus haut du niveau physique, car c'est en fonction de la situation courante que l'on va déterminer quel module du niveau physique va exécuter l'action courante. Et chaque module d'exécution doit faire face aux imprévus et aux imperfections et posséder une réactivité réflexe.

Cette analyse nous permet de définir les caractéristiques de notre architecture. Dans la section suivante, nous présentons les différents modules qui composent chaque niveau et leur organisation les uns par rapport aux autres.

3.4 Les composantes d'une architecture de contrôle et leur organisation

3.4.1 Introduction

Nous nous sommes jusqu'à maintenant intéressés aux caractéristiques que doit posséder une architecture de contrôle, ainsi qu'aux contraintes auxquelles elle doit répondre. Dans cette section, nous analysons à travers les exemples présentés quelles sont les composantes[4] d'une architecture de contrôle, et comment sont organisées ces composantes.

Nous allons en nous appuyant sur les architectures présentées, détailler les différentes composantes d'une architecture de contrôle, le niveau auquel elles se situent, et leur organisation les unes par rapport aux autres.

3.4.2 Le planificateur de missions

Dans les architectures présentées, on trouve toujours dans le niveau décisionnel (appelé aussi niveau cognitif) un module de planification de missions. Il reçoit ses missions de l'utilisateur.

Dans l'architecture du projet SHARP, il a un rôle très limité puisqu'il détermine simplement les buts à atteindre. Ceci tient au fait que le véhicule autonome du projet SHARP réalise uniquement des déplacements. Il reçoit ses missions de l'utilisateur.

Lorsqu'on examine une architecture telle que celle du robot mobile HILARE, on retrouve au plus haut niveau un "véritable" planificateur de missions, qui décompose une mission sous forme d'une suite d'actions. Ces actions restent des actions dites de très haut-niveaux, telles que "se rendre quelquepart" ou "saisir un objet", qui ne peuvent être exécutées directement par le robot.

L'architecture de Brooks constitue un cas à part. En effet, Brooks considère que son architecture permet de planifier des missions complexes. Malheureusement, il ne parle pas de la façon dont fonctionne le sixième niveau de son architecture, celui planifie et gère l'exécution des missions. De plus, il paraît inconcevable que le planificateur de missions puisse manipuler des données issues des capteurs, sans qu'elles aient été traitées auparavant.

En conclusion, nous pouvons dire que le rôle du planificateur de missions est d'interpréter les missions données par l'utilisateur, pour les transformer en une suite d'actions non exécutables directement par le robot. Il a aussi pour rôle de superviser l'exécution des missions, et de replanifier les missions suite à un imprévu ; il gère la réactivité stratégique. Il manipule des données symboliques parfaites. Il se situe dans le niveau décisionnel (ou planificateur).

3.4.3 Le planificateur de chemins

Pour pouvoir se déplacer un robot mobile a besoin d'un planificateur de chemins. C'est en effet lui qui en fonction d'un but à atteindre (ou d'une action de haut niveau) fourni par

4. On parle aussi de modules, de comportements ou d'aptitudes

le planificateur de missions planifie les mouvements qui vont être exécutés par le robot.

On trouve généralement ce module entre le planificateur de missions et le niveau fonctionnel. Selon les cas, il se situe dans le niveau cognitif (architecture du projet SHARP) ou dans le niveau physique (architecture du robot mobile HILARE).

Son rôle peut être plus ou moins important :

- Soit il retourne une trajectoire complète à suivre par le robot. Il spécifie complètement les actions que le robot exécutera pour atteindre son but. C'est le cas par exemple pour le planificateur de mouvements de l'architecture du projet SHARP.
- Soit il fournit une liste de sous buts que doit réaliser le robot. Il se contente ici de décomposer l'action abstraite fournie par le planificateur de missions en actions élémentaires exécutables par le robot. C'est le cas par exemple du le robot SHAKEY.

La deuxième solution parait plus intéressante. En effet, elle permet une plus grande autonomie lors de l'exécution des actions puisqu'en fait elle ne spécifie que des sous-buts à atteindre pour réaliser une action. Le robot est donc complètement autonome dans la manière d'atteindre ces sous-buts.

3.4.4 Le contrôleur d'exécution

Dans les architectures présentées, on trouve un module de contrôle d'exécution. Son rôle est de gérer l'exécution des actions élémentaires. Il doit en fonction du contexte et de la tâche à réaliser déterminer quelle est la meilleure action élémentaire à exécuter, et activer le module d'exécution correspondant. Il doit de plus faire face aux imprévus que les modules d'exécution ne sont pas capables de résoudre. Son fonctionnement dépend de la tâche qui lui est assignée par le planificateur de mouvements.

En effet, dans une architecture comme celle du projet SHARP, le planificateur de mouvements fournit une trajectoire complète à suivre, et le rôle du contrôleur d'exécution est de veiller à ce que le robot reste sur cette trajectoire. Il doit donc mettre en place les asservissements adaptés. Dans le cas où un imprévu se produit, un obstacle sur la trajectoire par exemple, il est chargé de positionner le robot sur un chemin adjacent vierge d'obstacles, et de s'assurer que le robot reprend sa trajectoire initiale une fois l'obstacle évité. Dans ce cas, le contrôleur d'exécution gère à la fois la réactivité réflexe puisqu'il est chargé de mettre en place des asservissements pour éviter un obstacle et la réactivité tactique puisqu'il doit aussi en fonction du contexte et de la tâche à réaliser déterminer les asservissements à activer.

Si la trajectoire n'est pas fixée, le contrôleur d'exécution doit veiller à ce que le robot se dirige vers son but, et doit fournir une solution dans le cas où un imprévu se présente. C'est le cas par exemple pour le robot SHAKEY. Dans ce cas, ce module ne gère que la réactivité tactique.

Dans une architecture comme celle de Brooks, il n'y a pas de contrôleur d'exécution et c'est le navigateur qui gère l'exécution des déplacements.

Suite à ce que nous avons dit pour le planificateur de mouvements, il paraît plus intéressant à partir d'une liste de sous buts à atteindre, de spécifier qu'elles doivent être les actions à réaliser par le navigateur pour atteindre le but, et de contrôler la bonne exécution de ces actions. Le contrôleur se place toujours au plus haut niveau dans le niveau fonctionnel, puisque c'est lui qui à partir des décisions prises dans le niveau décisionnel, va décider quel module activer pour exécuter ces décisions.

3.4.5 Le navigateur

Le navigateur [5] joue un rôle crucial au niveau décisionnel. C'est en effet lui qui permet au robot de se déplacer. Il doit à chaque instant à partir des données bruitées des capteurs et du but à atteindre définir la meilleure action motrice à exécuter par le robot.

Dans une architecture comme celle du projet SHARP, il est composé d'un ensemble d'asservissements qui permettent au robot de suivre la trajectoire du planificateur de mouvements. Le choix de l'asservissement est déterminé par le module de contrôle d'exécution. Il doit d'autre part détecter les obstacles imprévus afin que le contrôleur d'exécution puisse mettre en place un évitement d'obstacles. Son rôle est de suivre les consignes qui lui sont fournies par le contrôleur d'exécution.

Lorsqu'il est beaucoup moins contraint dans la façon d'atteindre un but, c'est par exemple le cas dans l'architecture du robot mobile HILARE, le navigateur a une ou plusieurs stratégies pour atteindre un but, il possède en plus des réactions prédéfinies qui lui permettent de faire face aux imprévus tel qu'un obstacle sur sa trajectoire[Khatib, 1996].

Dans l'architecture de Brooks, le navigateur gère complètement les déplacements et les imprévus. Il doit être capable de définir en fonction du contexte la meilleure action à réaliser par le robot.

Le rôle du module de navigation réactive dépend directement du rôle du planificateur de mouvements et de celui du contrôleur d'exécution. Il peut aller du simple suivi de trajectoires, à l'exécution autonome complète d'une action de déplacement. Nous pensons que son rôle doit être à mi-chemin entre ces deux extrêmes. Il doit être capable de gérer des modifications simples de l'environnement qui n'entraîne pas de remise en cause de l'action à exécuter, mais ne doit pas avoir la charge de définir en fonction du contexte la meilleure action à réaliser.

3.4.6 Le localisateur

Il y a au niveau fonctionnel, un autre module qui joue un rôle très important, c'est le localisateur [6]. En effet, le simple fait de répondre à la question : "Où suis-je ?" est un véritable problème pour un robot mobile. Pourtant la réponse à cette question est primordiale, puisque c'est grâce à elle que le robot connait sa position dans son environnement et qu'il peut déterminer l'état d'avancement de l'exécution d'une action.

Son fonctionnement peut être très divers et dépend du type d'informations qu'il va devoir mettre à disposition des autres modules et de la fréquence à laquelle il va devoir fournir ces informations. On peut en effet envisager de nombreux cas :

- Dans l'architecture du projet SHARP, le robot doit connaître à chaque instant sa position précise afin de savoir s'il se trouve toujours sur la trajectoire fournie

5. appelé aussi le module de navigation.
6. appelé aussi le module de localisation.

par le planificateur de chemins. Dans ce cas, le robot doit être muni d'un système odométrique (c-à-d, un système qui lui permet en fonction de la vitesse à laquelle il se déplace de calculer une estimée de sa position). Dans l'architecture du projet SHARP, ce système est considéré comme étant suffisamment fiable pour connaitre la position précise du robot. Pourtant la majorité des localisateurs se servent de l'odométrie pour avoir une idée grossière de la position du robot et utilisent d'autres techniques pour affiner la position, telles que par exemple les filtres de Kalman.
- Il peut simplement avoir besoin de connaître de temps en temps sa position grossière afin de déterminer s'il a atteint son but ou non. Dans ce cas, il se sert de techniques de reconnaissance de formes pour reconnaître des caractéristiques de son environnement et s'en servir pour se localiser.

Nous pensons qu'il est préférable de localiser le robot de manière grossière de temps en temps afin de vérifier l'état d'exécution d'un déplacement. Pour cela, nous utilisons des techniques de reconnaissance de formes afin d'extraire les caractéristiques de l'environnement qui correspondent à des lieux distincts de l'environnement. Cette idée est à la base de notre travail sur l'apprentissage et la reconnaissance de balises naturelles et son application à la réalisation d'un module de localisation et de contrôle d'exécution d'actions.

3.4.7 Les autres composantes

Il y a d'autres composantes dans une architecture de contrôle.

Nous avons par exemple vu que le niveau décisionnel travaille avec des données abstraites et non bruitées, alors que le niveau fonctionnel utilise les données bruitées des capteurs. Il faut donc définir une abstraction de ces données capteurs afin de les rendre exploitables par le niveau décisionnel. C'est le rôle d'un module d'interprétation des données ou de construction de cartes. Ce module se trouve dans le niveau fonctionnel. Dans l'architecture du projet SHARP, à chaque instant un modèle de l'environnement statique et dynamique est mis à jour. D'une manière radicalement opposée, Brooks ne construit pas de carte de l'environnement, en considérant que son modèle de l'environnement est l'environnement lui-même. Nous considérons pour notre part qu'il vaut mieux interpréter les données des capteurs afin d'en extraire des informations pertinentes sur les changements de l'environnement.

Nous avons jusqu'à présent parlé uniquement du navigateur qui permet au robot de se déplacer. De nombreux robots sont capables d'autres actions physiques telles que saisir un objet par exemple pour les robots munis d'un bras. Il faut dans ce cas définir des modules qui permettent d'utiliser ces organes tout en gérant la réactivité réflexe.

Au niveau décisionnel, nous avons des modules qui permettent de gérer des contraintes temporelles ou des priorités sur des missions à réaliser.

3.4.8 Conclusion

La conception d'une architecture de contrôle nécessite des compétences diverses et nombreuses. Nous avons dans cette section défini les modules indispensables pour réaliser notre architecture de contrôle, ainsi que leurs rôles et organisations.

Dans le niveau cognitif, on trouve au minimum un planificateur de missions. Son rôle est d'interpréter les missions données par l'utilisateur, pour les transformer en une suite d'actions non exécutables directement par le robot. Il a aussi pour rôle de superviser l'exécution des missions, et de replanifier les missions suite à un imprévu, puisque c'est lui qui gère la réactivité stratégique. Il manipule des données symboliques parfaites.

Dans le niveau physique, on trouve au moins quatre modules :

- Un planificateur de mouvements. De son fonctionnement va dépendre le fonctionnement de tous les autres modules du niveau physique actifs lors d'un déplacement du robot. Il peut soit planifier une trajectoire complète à suivre par le robot, soit spécifier une suite de buts à atteindre. Dans le premier cas, le robot doit s'asservir pour suivre la trajectoire, tout en mettant en oeuvre des mécanismes d'exception pour prendre une trajectoire parallèle libre de tous obstacles et des mécanismes de rattrapage de la trajectoire initiale lorsque l'obstacle est évité. Dans le second cas, il spécifie une liste de sous-buts, et détermine à chaque instant la meilleure action à réaliser pour se rapprocher de son but, tout en évitant les obstacles. C'est clairement ce genre d'approche que nous retenons pour notre architecture. En effet, cette approche est plus simple à mettre en oeuvre, comme nous le verrons lors de la description des modules de notre architecture. D'autre part, cette méthode laisse plus de liberté au robot dans l'exécution de ces actions.
- Un contrôleur d'exécution d'actions qui va choisir à chaque instant en fonction du contexte et de la tâche à réaliser la meilleure action à exécuter par le robot et active le module d'exécution correspondant. Il se place toujours au plus haut dans le niveau fonctionnel, puisque c'est lui qui à partir des décisions prises dans le niveau décisionnel décide quel module doit exécuter ces décisions. Il gère la réactivité tactique.
- Un navigateur qui permet au robot de se déplacer et d'atteindre un but tout en évitant les obstacles.
- Un localisateur qui permet au robot de connaître sa position dans son environnement, et détermine si un but est atteint ou non. Pour cela, il utilise des balises naturelles de l'environnement qu'il apprend à reconnaître.

3.5 Conclusion

Un système autonome et intelligent tel qu'un robot mobile évoluant dans un environnement incertain, doit être capable de s'adapter aux changements de celui-ci. Il doit être en effet capable de modifier son comportement en fonction de stimuli par le biais de comportements réflexes, en fonction de sa situation par le biais de réactions tactiques, de choisir la meilleure action à réaliser en fonction du contexte et du but à atteindre, tout en maintenant un haut niveau de planification susceptible de replanifier une mission suite à de fortes modifications de l'environnement. Toutes les architectures présentées ne respectent pas ces trois niveaux de réactivité, c'est-à-dire de prise en compte de l'information capteur. Ceci ne tient pas au type de l'architecture, mais plutôt à sa conception. Par exemple, l'architecture du robot SHAKEY ne permet qu'une prise en compte de la réactivité stratégique, alors que l'architecture NASREM permet la prise en compte des trois niveaux de réactivité. La plupart des architectures hybrides actuelles répondent généralement, aux trois niveaux de réactivité (réflexe, stratégique et tactique). Nous avons vu que pour définir une architecture de contrôle répondant à ces trois niveaux, il faut que le planificateur de mission soit capable de replanifier suite à de fortes modifications de l'environnement, que le contrôleur d'exécution soit capable de définir la meilleure action à réaliser par le robot en fonction du contexte et du but à atteindre, et que les modules d'exécutions soient capables d'intégrer des imperfections et de modifier leur comportement en conséquence.

D'autre part, la définition d'une bonne architecture de contrôle repose sur une décomposition du contrôle en niveau. Un niveau est généralement composé d'un ensemble de comportements ou d'aptitudes qui lui permettent de réaliser la tâche à laquelle le niveau est dédié. Le contrôle de chaque niveau est géré par un mécanisme d'arbitrage qui décide en fonction du contexte et de la tâche à accomplir, quel comportement est le plus adapté à la situation courante les activations/désactivations des comportements. Chaque niveau est contrôlé ou supervisé par le niveau supérieur. On parle souvent d'architectures comportementales [Tigli, 1996]. Nous avons remarqué que l'ensemble des facultés que doit avoir un robot mobile pour être autonome se divise en deux grandes catégories qui constituent les deux niveaux minimums d'une architecture de contrôle :

- des facultés décisionnelles (ou cognitives) qui permettent d'interpréter de comprendre et de décomposer les missions. Dans ce niveau, on doit au moins trouver un planificateur de missions.
- des facultés fonctionnelles (ou physiques) qui permettent d'exécuter les missions du robot tout en faisant face aux imprévus. Dans ce niveau, nous avons vu qu'on doit au moins trouver un contrôleur d'exécution qui gère le niveau et décide en fonction du contexte et de l'action à réaliser quels modules doivent être activés. Pour pouvoir se déplacer, le robot détermine en fonction du but à atteindre une liste de sous-buts. Il a pour cela besoin d'un planificateur de mouvements. Pour se rendre en chacun des sous buts, il faut disposer d'un navigateur afin que le robot soit capable de se déplacer. Ce navigateur doit être autonome dans la façon d'atteindre les sous buts afin d'éviter les obstacles sur son trajet. D'autre part, il faut que le robot ait les moyens de savoir quand il a atteint un sous but. Pour cela, il doit disposer de capacités qui, à partir des données bruitées des capteurs, lui permettent de déterminer l'état de l'environnement ainsi que sa position dans l'environnement. Pour terminer, si le robot possède d'autres capacités physiques, il faut définir des modules qui permettent de gérer ces capacités physiques.

Une architecture de contrôle hybride à deux niveaux, un niveau cognitif planifiant et décomposant les missions et un niveau physique les exécutant est la solution que nous avons retenue. Dans le chapitre suivant, nous présentons l'architecture AIBOT qui remplit les conditions que nous avons mentionnées tout au long de ce chapitre, et expliquons comment nous nous en sommes inspirés pour définir la structure de notre architecture de contrôle.

4

Notre architecture de contrôle

4.1 Introduction

Ce chapitre est consacré à la réalisation de notre architecture de contrôle [Morignot *et al.*, 1997]. Dans un premier temps, nous présentons le robot mobile Nomad200 sur lequel nous avons travaillé. Nous examinons en particulier son architecture matérielle et l'ensemble de ses capteurs. Nous présentons ensuite l'architecture Aibot dont nous nous sommes inspirés. Cette architecture répond en effet tout à fait aux besoins et exigences que nous avons mis en évidence dans le chapitre précédent. D'autre part, la décomposition du contrôle est basée sur le concept de niveau générique. Les différents niveaux de l'architecture ont une structure identique mais une fonction différente.

A partir de là, nous définissons notre architecture de contrôle. Nous commençons tout d'abord par la spécification de notre modèle de niveau. Chaque niveau comporte des composantes communes telles qu'une mémoire partagée qui gère tous les événements postés par les différents comportements, un module de planification qui permet de décomposer la tâche à résoudre par le niveau, et un contrôleur d'exécution qui en fonction du contexte et de la tâche à réaliser active les comportements appropriés. Afin de résoudre la tâche à laquelle il est dédié, chaque niveau comporte, en plus, des comportements spécifiques. Nous détaillons ensuite les deux niveaux instances du modèle qui constituent notre architecture, ainsi que les modules qui les composent. Un niveau cognitif qui planifie et décompose les missions tout en gèrant la réactivité stratégique. Ce niveau ne comporte pas de modules spécifiques, et le planificateur est ici un planificateur de missions. Un niveau physique qui exécute les actions constituant la mission. Pour cela, ce niveau comporte trois comportements spécifiques :

- un navigateur qui permet au robot de se déplacer.
- un localisateur afin que le robot connaisse sa position.
- un module permettant au robot de saisir et de rendre des objets.

Le planificateur sert dans ce niveau à déterminer une liste de sous-buts à atteindre pour réaliser une action de déplacement vers un but. Pour terminer, nous illustrons le fonctionnement de notre architecture par un exemple de mission de manipulations d'objets.

4.2 Description de notre robot mobile

FIG. 4.1 – *Notre robot mobile Gaston*

Le robot (figure 4.1) est un Nomad200 commercialisé par [Nomadics, 1996]. Il est constitué d'une base et d'une tourelle. La base est formée de trois roues motrices et directrices, et de capteurs tactiles. La tourelle est un polygone uniforme à seize faces. Sur chaque face, il y a un capteur infrarouge et un capteur ultrason. La tourelle peut tourner indépendamment de la base. Un bras manipulateur (une pince mobile verticalement) est monté à l'arrière de la tourelle. Il avance à une vitesse maximale d'environ soixante centimètres par seconde et tourne à une vitesse maximale de quarante cinq degrés par seconde. Il mesure environ cinquante cinq centimètres de large et environ quatre-vingts centimètres de hauteur.

Le système de contrôle est un système multi-processeurs à mémoire partagée. Le processeur maître est un Pentium 120 utilisant le système d'exploitation Linux. Le système utilise aussi de nombreux coprocesseurs intelligents situés sur le bus qui partagent une mémoire avec le Pentium. Le robot est relié au réseau local par une liaison radio sans fil. Il est alimenté par cinq batteries de 12V, dont trois alimentent le circuit de contrôle du robot et les deux autres alimentent ces moteurs. Les batteries donnent au robot une autonomie d'environ six heures. Un environnement de développement permet le développement et le test des programmes de contrôle du robot.

4.2.1 Les Capteurs ultrasons

Les capteurs ultrasons (figure 4.2) au nombre de seize donnent une mesure entre dix sept et deux cent cinquante cinq pouces avec une erreur de 1% environ. La distance

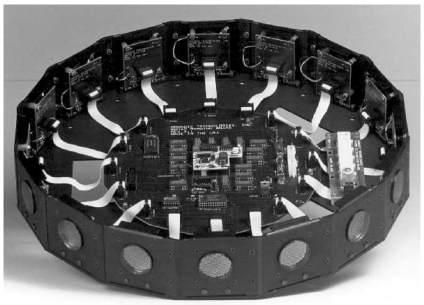


FIG. 4.2 – *Les Capteurs Ultrasons*

entre le robot et un objet est déterminée par le temps de vol d'un signal acoustique envoyé par un transducteur et réfléchi par l'objet. Les capteurs ultrasons effectuent une mesure les uns après les autres, afin d'éviter les phénomènes de croisement entre signaux acoustiques. L'angle entre deux capteurs ultrasons est de 22,5 degré, et le signal émis par chaque capteur ultrason a une largeur approximative de 23,6 degré. En examinant les seize capteurs ultrasons, nous pouvons obtenir rapidement une vue complète de l'environnement local du robot.

4.2.2 Les Capteurs infrarouges

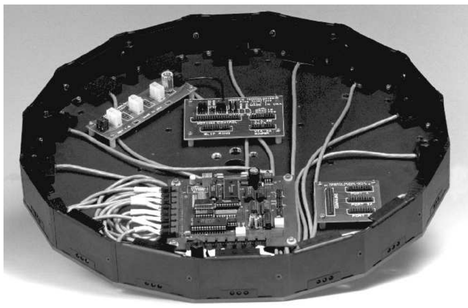


FIG. 4.3 – *Les Capteurs Infrarouges*

Les capteurs infrarouges (figure 4.3) mesurent la différence de luminosité entre la lumière émise et la lumière réfléchie. Ils sont très sensibles à la lumière ambiante, la couleur et l'orientation de l'objet. Nous supposons que pour de courtes distances, la mesure est acceptable, nous utilisons les capteurs infrarouges pour des mesures de moins de 17 pouces, là où les capteurs ultrasons sont inutilisables.

4.2.3 Les Capteurs tactiles

Une ceinture de vingt capteurs tactiles entoure la base. Ils détectent les contacts avec les objets. Ils sont seulement utilisés dans les cas d'urgence. Ils sont associés aux réflexes de bas niveau comme l'arrêt d'urgence ou la marche arrière.

4.2.4 Le Bras manipulateur

FIG. 4.4 – *Le Bras manipulateur*

Avec sa pince, le bras (figure 4.4) situé à l'arrière de la tourelle est utilisé pour manipuler des objets. La tourelle permet les déplacements dans le plan pour s'approcher d'un objet, et le bras permet de se positionner à la hauteur de l'objet. Le bras peut s'élever jusqu'à une hauteur d'environ 1,20 mètres. La pince est équipée de deux doigts. Les deux doigts sont utilisés pour saisir et rendre des objets.

4.2.5 Le système odométrique

Ce système intègre les translations et les rotations du robot, et met à jour la position et l'orientation du robot. Comme tout système odométrique, il accumule les erreurs au fur et à mesure des déplacements, dûes aux glissements et aux frottements sur le sol. Nous l'utilisons uniquement pour avoir une idée grossière de la position et de l'orientation du robot.

4.2.6 Le système de stéréovision

Ce système comporte deux cameras CCD pouvant acquérir des images en 256 niveaux de gris aux formats NTSC et CCIR. Deux résolutions sont disponibles, une haute (512 x 512) et une basse (256 x 256). Les images peuvent être acquises de façon périodique ou sur commande.

4.2.7 Le télémètre laser

Il comporte une caméra CCD et un laser (figure 4.5). Il a une portée théorique de 12 à 120 pouces. La diode laser est utilisée comme source lumineuse et la caméra pour capter l'image. La diode émet un faisceau lumineux, dans un plan horizontal. A partir du faisceau lumineux extrait de l'image, le système calcule par triangulation les coordonnées des points du plan horizontal.

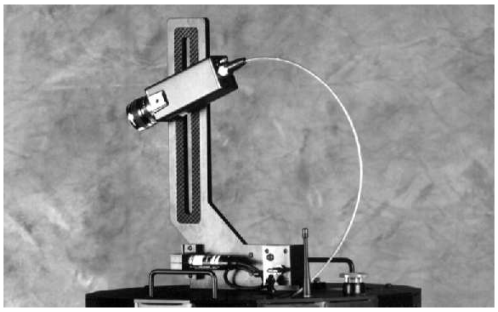


FIG. 4.5 – *Le télémètre laser*

4.3 L'architecture Aibot

4.3.1 Description

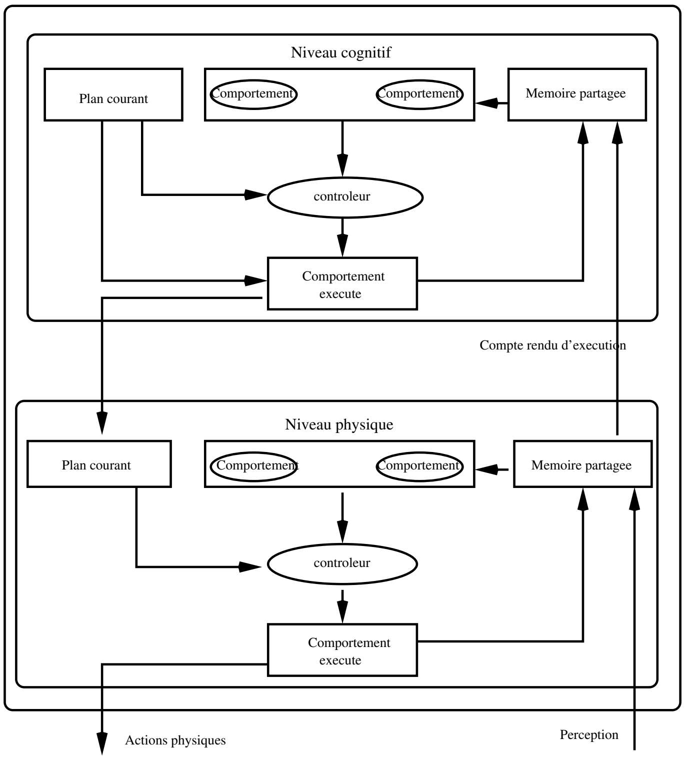


FIG. 4.6 – *L'architecture de l'équipe KSL*

L'architecture de contrôle Aibot développée au sein du Knowledge System Laboratory de l'Université de Stanford [Hayes-Roth *et al.*, 1993] est composée de deux niveaux (figure 4.6) :

– Un niveau cognitif qui intègre et gère les connaissances que le robot a de son environnement. Il planifie et supervise les missions du robot, grâce à un ensemble de comportements cognitifs du robot telles que la planification de tâches ou la résolution de problèmes. Ce niveau reçoit des requêtes de l'opérateur et lui renvoie des comptes-rendus d'exécution. Il envoie au niveau physique des descriptions d'actions, et des qualificatifs sur cette description et reçoit des comptes-rendus d'exécution des actions qu'il analyse pour poursuivre ou replanifier la mission qui lui est donnée.

- Un niveau physique qui permet au robot d'interagir avec son environnement grâce à des comportements physiques qui permettent au robot d'agir (se déplacer, saisir des objets...) et de percevoir l'environnement. Il reçoit du niveau cognitif des descriptions d'actions à effectuer, et renvoie des comptes-rendus d'exécution.

La structure de chaque niveau basée sur BB1 [Hayes-Roth, 1985] est identique, seul son contenu diffère :

- Une mémoire commune qui permet de stocker les informations sur l'état de l'environnement.
- Un ensemble de comportements qui définissent ce que le robot est capable de faire. Chaque comportement est composé de deux parties, une qui définit les conditions d'activation du comportement, et l'autre qui contient le corps de son exécution.
- Un contrôleur qui décide quelle action doit être exécutée en fonction de la mémoire commune, du plan à exécuter et des conditions d'activation des différentes aptitudes.

Cette architecture est utilisée pour le contrôle d'un robot mobile Nomad200 [Nomadics, 1996] dans des missions de surveillance[Hayes-Roth *et al.*, 1993]. Les comportements utilisés sont :

- Au niveau cognitif :
  - Un comportement qui permet de générer de nouvelles missions à partir des observations.
  - Un système de contrôle qui permet de sélectionner et de gérer les différents buts, non seulement en fonction de leur importance, mais aussi en fonction des motivations et des besoins du robot lui-même.
  - Un générateur de plans qui à partir d'une mission genère la liste des actions à effectuer.
  - Un exécuteur de plans qui réalise et supervise l'éxecution des actions, et qui prend en compte des contraintes de temps sur les actions.
- Au niveau physique :
  - Deux comportements de navigation réactive :
    - Un basé sur des champs de potentiels amélioré.
    - L'autre basé sur des suivi de murs.
  - Un comportement de suivi de balises qui permet au robot de suivre des balises artificielles au plafond pour qu'il se localise.
  - Un planificateur de chemins qui calcule un chemin entre la position courante du robot et son but. Ce planificateur [Zhu et Latombe, 1989] est basé sur une décomposition de l'espace en rectangle qui ne contiennent pas d'obstacles. La recherche s'effectue en cherchant un chemin dans l'espace des rectangles qui aille de la position initiale du robot, jusqu'à son but.

4.3.2 Analyse

L'architecture Aibot remplit l'ensemble des conditions que nous avons déterminées. C'est une architecture hybride composée de deux niveaux. Chaque niveau est spécialisé dans la résolution d'un type précis de tâche et regroupe les différentes fonctions opératoires lui permettant de résoudre ce type de tâches. A l'intérieur de chaque niveau, les fonctions opératoires coopèrent ou sont en compétition pour contrôler le niveau. Le contrôleur d'exécution de chaque niveau gère la réactivité du niveau. En effet, c'est lui qui détermine en fonction du contexte et de la tâche à réaliser quels modules doivent être activés. Les niveaux sont organisés de manière hiérarchique. Les niveaux en bas de la hiérarchie manipulent des informations numériques et agissent directement sur l'environnement. Les niveaux en haut de hiérarchie manipulent des données symboliques, et contrôlent les bas niveaux. La circulation des informations entre deux niveaux consécutifs se fait de manière ascendante et descendante.

4.4 Notre architecture

Nous nous sommes attachés à développer une architecture de contrôle la plus générique possible [Morignot *et al.*, 1997]. Ce travail a été réalisé en collaboration avec Philippe Morignot. Nous avons choisi d'utiliser une structure proche de celle d'Aibot (section 4.3), car elle répond bien aux besoins et exigences énoncés dans le chapitre précédent. De plus, elle a déjà été utilisée sur des applications réelles [Hayes-Roth *et al.*, 1993]. La définition claire et générique de niveau, et l'intégration aisée de comportements fournit un cadre de développement et de tests parfaits pour une application de robotique mobile réelle.

4.4.1 Le modèle de niveau de notre architecture

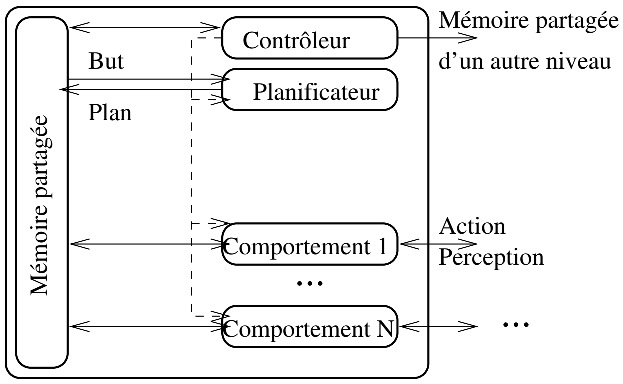


FIG. 4.7 – *Le modèle de niveau de notre architecture*

Chaque niveau comporte :

- Une mémoire partagée [Hayes-Roth, 1985] composée d'événements. Ces événements sont soit des missions ou des actions à réaliser, soit des comptes-rendus d'exécution de missions ou d'actions, soit des états d'avancement de l'exécution de missions ou d'actions, soit des informations pertinentes perçues pendant l'exécution.

- Un mécanisme de contrôle (*contrôleur*) qui choisit, en fonction des actions et/ou des missions à réaliser qui se trouvent dans la mémoire partagée du niveau, quelle est la tâche à résoudre en premier, et active le module spécialisé dans la résolution de cette tâche. Le contrôleur est à partir de ce moment-là en attente d'un événement posté par un comportement. Cet événement est généralement un compte-rendu d'exécution. Une fois l'événement attendu posté, le contrôleur définit la nouvelle tâche à résoudre, et active le comportement approprié. Ce processus se répète tant qu'il y a dans la mémoire partagée des tâches à résoudre.

  Il gère aussi la circulation d'informations entre les niveaux. Le contrôleur de chaque niveau est en relation avec la mémoire partagée d'un autre niveau afin de pouvoir lui transmettre des tâches à résoudre, ou des actions à exécuter qui sont de sa compétence. Une fois la tâche réalisée ou l'action exécutée, le contrôleur du niveau qui a résolu cette tâche poste un compte-rendu de l'exécution de cette action dans la mémoire partagée du niveau qui lui a demandé de résoudre cette tâche.
- Un planificateur qui permet de décomposer le problème à résoudre par le niveau afin qu'il soit directement exécutable par les comportements du niveau ou qu'il soit posté dans la mémoire d'un autre niveau par le contrôleur.
- Des modules indépendants (*comportements*) qui vont permettre au niveau de résoudre une tâche spécifique. En fonction d'une tâche à réaliser dans laquelle il est spécialisé, un comportement va exécuter cette tâche. Il est pour cela activé par le contrôleur. Au cours de l'exécution, il poste dans la mémoire partagée différents types d'événements qui vont correspondre soit à l'état d'avancement de la tâche, soit au résultat de l'exécution, soit à des événements perçus pendant l'exécution. Un comportement est soit désactivé par le contrôleur soit s'arrête automatiquement une fois qu'il a résolu la tâche pour laquelle il a été activée.

Les caractéristiques que nous venons de définir pour notre architecture reprennent la plupart des caractéristiques des architectures présentées dans le chapitre précédent, et en particulier Aibot (section 4.3). Notre architecture en est en effet très proche. Sa structure est identique, et celle du modèle de niveau est très proche. Les différences sont en effet mineures :

- Notre modèle de niveau comporte toujours au moins un planificateur (planificateurs de missions ou de chemins) qui planifie les prochaines actions du niveau. Ce comportement nous paraît nécessaire dans toute application de robotique mobile.
- D'autre part, le fonctionnement du contrôleur et d'un niveau en général est nettement plus simple que dans Aibot. Dans notre modèle de niveau, c'est le contrôle qui détermine quel est le prochain comportement à activer. Il n'y a pas de condition d'activation des comportements du niveau qui permettent de définir quel comportement le contrôleur peut activer.

4.4.2 Application au contrôle de notre robot mobile

Au vu de la description de notre robot mobile, nous considérons qu'il doit être capable de se déplacer dans son environnement, et d'y saisir et rendre des objets. Il peut donc

être utilisé pour deux types de tâches :
- des tâches de manipulation d'objets.
- des tâches de surveillance.

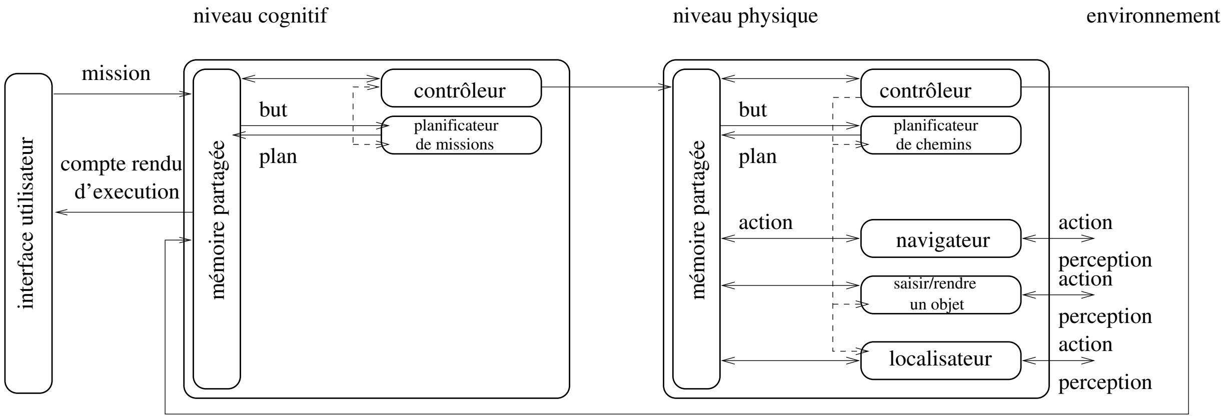


FIG. 4.8 – *Notre architecture*

Notre architecture (figure 4.8) est donc composée de deux niveaux :
- Un niveau cognitif (section 4.4.3) qui décompose les missions du robot et envoie les actions à exécuter au niveau physique. Il reçoit des compte-rendus sur l'exécution des actions et des informations perceptuelles (porte ouverte sur la droite, robot perdu, obstacles inattendus...). Ce niveau aura pour but de traiter, d'intégrer et de raisonner sur les informations données à priori au robot, et celles découvertes lorsque le robot évolue dans son environnement. Le planificateur est ici, un planificateur de tâches qui sert à décomposer les tâches du robot en actions de haut niveau non directement exécutables par le robot. Le niveau cognitif ne comporte actuellement pas de comportements spécifiques.
- Un niveau physique qui est en contact avec l'environnement par l'intermédiaire des capteurs et des moteurs. Il permet au robot d'agir et de percevoir. Les résultats de ces actions et les interprétations de ces perceptions sont communiqués au niveau cognitif. Il est composé des modules suivants :
  - Le planificateur est ici un planificateur de mouvements qui sert à planifier les mouvements du robot avant qu'il se déplace.
  - Un navigateur qui sert à déplacer le robot et à faire face aux imprévus lors de ses déplacements.
  - Un localisateur qui sert au robot à se localiser, et à obtenir de nouvelles informations sur son environnement.
  - Un manipulateur d'objets qui permet au robot de saisir et de rendre des objets.

Pour utiliser notre architecture, un utilisateur spécifie une tâche de haut niveau au robot. Cette demande est postée dans la mémoire partagée du niveau cognitif. Le contrôleur du niveau cognitif se rend compte de la présence d'une tâche non résolue. Il active donc le planificateur qui décompose la tâche sous forme de descriptions d'actions de haut niveau et les poste dans la mémoire partagée du niveau physique. Dès qu'une description d'action de haut niveau non exécutée se trouve dans la mémoire partagée, le contrôleur poste la description de cette action à effectuer dans la mémoire partagée du niveau physique. A ce moment-là, le contrôleur du niveau cognitif attend le compte-rendu d'exécution de cette action. Ce compte-rendu sera posté par le contrôleur du niveau physique, une fois l'action exécutée. Indépendamment, le planificateur de missions continue à décomposer la tâche et à poster les actions de hauts niveaux à exécuter dans la mémoire partagée. L'événement posté dans la mémoire partagée du niveau physique déclenche automatiquement le contrôleur du niveau physique. Si l'action à exécuter est un déplacement alors le contrôleur active le planificateur qui retourne une liste de sous-buts à atteindre. Le contrôleur active le comportement qui permet de réaliser l'action (saisir/rendre un objet ou se déplacer et se localiser). Le contrôleur poste le résultat de l'action dans la mémoire partagée du niveau cognitif, et des informations sur l'interprétation des perceptions. Selon le résultat de l'action exécutée, le contrôleur du niveau cognitif peut continuer l'exécution de la tâche, l'abandonner ou la replanifier.

4.4.3 Le niveau cognitif

Dans notre application, le niveau cognitif ne contient pas de comportements spécifiques. Nous décrivons donc uniquement comment fonctionne le contrôleur du niveau et le planificateur de missions.

Le contrôleur

Le rôle du contrôleur est de superviser l'exécution des actions constituant une mission. Il est actif dès qu'une mission se trouvant dans la mémoire partagée contient des actions non exécutées. A ce moment-là, il active le planificateur de missions. Dès que la première action à exécuter est postée dans la mémoire partagée par le planificateur de missions, il la poste dans la mémoire partagée du niveau physique et attend le compte-rendu d'exécution de cette action. Si l'action s'est bien éxecutée alors il poste l'action suivante dans la mémoire partagée du niveau physique, et dans le cas contraire, il active à nouveau le planificateur de missions. Le contrôleur répète ce processus jusqu'à ce qu'il n'y ait plus d'actions non exécutées dans la mémoire partagée. Ce module gère la réactivité stratégique puisqu'en fonction des comptes-rendus d'exécution des actions, il décide de poursuivre, de replanifier ou d'abandonner la mission en cours.

Le planificateur de missions

Le planificateur est à ce niveau un planificateur de tâches qui sert à trouver une séquence d'actions (un *plan*) qui transforme la description d'une situation initiale en une autre dans laquelle la tâche est résolue.

La description de la situation initiale est typiquement la situation initiale du robot, et le but représente la situation dans laquelle se trouvera le robot quand la tâche sera exécutée. Il utilise un ensemble de descriptions des actions que le robot peut réaliser physiquement (figure 4.9). Une action est représentée par une pré-, post-condition [Fikes et Nilsson, 1971] et des conditions de préservation (i.e., des conditions qui doivent être vérifiées tout au long de l'exécution) [Pednault, 1986].

```
(deftemplate (rendre ?robot ?objet ?position)
   :preservations ((robot ?robot)
                   (height ?object ?height)
                   (at ?robot ?location)
                   (clamp-height ?height))
   :preconditions ((at ?object ?robot)
                   (:not (is-clamp-open)))
   :postconditions ((at ?object ?location)
                    (:not (at ?object ?robot))
                    (:not (is-clamp-open)))
   :location ?location)
```

FIG. 4.9 – *Description de l'action "rendre un objet"*

La description de la situation initiale et du but forment le plan initial. Notre planificateur [Morignot, 1994] manipule des plans partiellement ordonnés et partiellement instanciés : deux actions exécutables en parallèle signifient qu'elles ne sont pas ordonnées. Une variable instanciée signifie qu'elle peut être instanciée à plus d'une constante (le domaine d'une variable est élagué pendant la planification). La planification de tâches est une recherche dans un espace partiellement ordonné et partiellement instancié [Chapman, 1987]. Une recherche par meilleur d'abord, par exemple, assure la complétude. Un plan est une solution à un problème donné (soit un plan initial plus une description des actions) si et seulement si il ne contient plus de préconditions ou de conditions de préservations insatisfaites. La satisfaction d'une précondition (ou, similairement, d'une préservation condition) est calculée en un temps polynomial avec un critère de vérité modale (MTC[1] [Chapman, 1987]) [Kambhampati et Nau, 1996]. Ce critère assure l'exactitude des plans générés.

Ce module poste dans la mémoire partagée les actions à exécuter qui composent le plan au fur et mesure qu'il les planifie. Une fois, sa tâche réalisée, il s'arrête automatiquement. Dans le cas, où le contrôleur lui demande de planifier une autre mission avant qu'il ait terminé de planifier la précédente, il abandonne automatiquement la planification de l'ancienne mission.

4.4.4 Le niveau physique

En plus du contrôleur et du planificateur de chemins, le niveau physique est composé de trois modules spécifiques :

- un module de navigation qui permet au robot de se déplacer dans les couloirs du laboratoire.
- un module de saisie/rendu d'objets qui permet au robot de saisir avec son bras manipulateur et de rendre des objets.
- un module de localisation qui permet au robot de reconnaître les balises naturelles devant lesquelles il passe, et ainsi de déterminer sa position dans l'environnement.

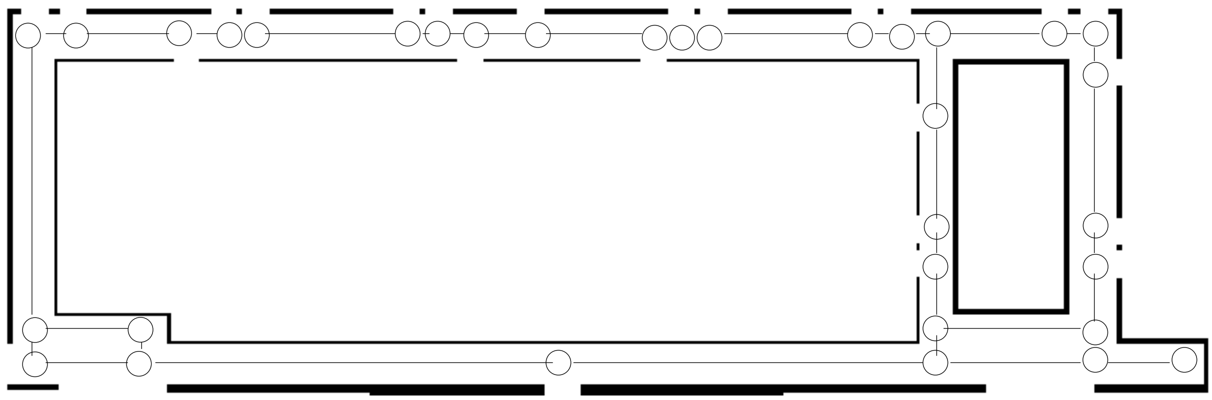


FIG. 4.10 – *La carte topologique et topographique du laboratoire*

Le planificateur de chemins

Le planificateur est ici un planificateur de chemins qui sert à trouver une séquence de sous-buts (un *plan*) qui permet à partir de la position initiale du robot d'atteindre un but donné. Ce comportement utilise une carte topographique et une carte topologique de l'environnement (figure 4.10), stockées dans la mémoire partagée du niveau physique. La carte topographique contient les obstacles statiques tels que les murs, les tables, les chaises ou les étagères. Un nœud de la carte topologique correspond à une balise de la carte topographique. Une balise est reconnue de manière unique par les capteurs du robot. Quand un chemin entre deux nœuds est unique, un arc est ajouté entre ces deux nœuds. Actuellement, la carte topologique est extraite manuellement. La planification de chemins implique :

- trouver la balise la plus proche de la position actuelle (en utilisant la carte topologique).
- trouver la balise la plus proche du but (en utilisant la carte topologique).
- réaliser une recherche dans la carte topologique entre la première et la dernière balise. Nous utilisons pour cela l'algorithme A^* qui nous assure la complétude.

Au fur et à mesure que les sous-buts sont déterminés, seuls les sous-buts correspondant à un changement de direction du robot sont postés dans la mémoire partagée du niveau physique. Dans le cas, où le contrôleur lui demande de planifier un autre chemin avant qu'il ait terminé de planifier le précédent, il abandonne automatiquement la planification de l'ancien chemin.

Le module de saisie et de rendu d'un objet

Ce comportement a pour rôle de saisir et de rendre des objets. Le plan pour saisir un objet est le suivant :

- Monter la pince verticalement pour atteindre la hauteur d'un objet.
- Ouvrir la pince.
- Fermer la pince jusqu'à ce qu'une résistance soit détectée.

- Si la pince est fermée et qu'aucune résistance n'a été détectée, réouvrir la pince. Recommencer l'opération.
- Descendre la pince verticalement afin de la remettre dans sa position de départ.

On suppose que les objets transportés ont une taille inférieure à la largeur de la pince. *A priori* les deux premières actions ne sont pas ordonnées.

Le plan pour rendre un objet est le suivant :

- Monter la pince verticalement pour atteindre la hauteur d'un objet.
- Ouvrir la pince.
- Descendre la pince verticalement afin de la remettre dans sa position de départ.
- Fermer la pince.

Pour des raisons de sécurité, le robot se déplace avec la pince dans une position basse. *A priori* les deux dernières actions ne sont pas ordonnées.

Une fois sa tâche réalisée, il s'arrête automatiquement, et poste dans la mémoire partagée le compte-rendu d'exécution de l'action de saisie ou de rendu d'objets. On considère à l'heure actuelle que ces deux actions ne peuvent pas échouer. Dans le cas, où le contrôleur lui demande d'exécuter une action de saisie/rendu d'objets, alors que le bras manipulateur est déjà en action, il abandonne automatiquement la planification de l'ancien chemin.

Le module de navigation

Son rôle est de permettre au robot de se déplacer de sa position actuelle vers le but qui lui a été assigné. Cette tâche est réalisée en deux étapes :

- Il se place tout d'abord dans la direction du but.
- Il se déplace réactivement vers son but.

C'est lui qui gère la réactivité réflexe lors des déplacements. Son fonctionnement sera détaillé dans le chapitre 6.

Au fur et à mesure que le robot se déplace, et découvre les obstacles, il poste la position des obstacles découverts dans la mémoire partagée du niveau physique. Ce comportement ne s'arrête pas, mais est arrêté par le contrôleur lorsque celui-ci le juge nécessaire. Il ne poste donc pas de compte-rendu d'exécution. Dans le cas, où le but à atteindre change en cours d'exécution, le navigateur place le robot dans la direction du nouveau but à moins qu'il n'y soit déjà, et le robot recommence à se déplacer automatiquement vers le nouveau but.

Le module de localisation

Son rôle est de permettre de connaître la position du robot dans son environnement au fur et à mesure de ses déplacements, et déterminer si le robot a atteint le prochain sous-but. Pour cela, il interprète les mesures des capteurs afin de reconnaître les balises de son environnement et de se localiser par rapport aux balises qu'il a reconnues. Au fur et à mesure que le robot se déplace, et reconnaît des balises, il poste l'état des balises qu'il reconnaît, afin que ces informations soient transmises au niveau cognitif par l'intermédiaire du contrôleur. En même temps que l'état de ces balises, il poste la position actuelle du robot. Il faut noter que la localisation du robot n'est réalisée qu'à chaque fois que le robot reconnait et interprète une balise de l'environnement. La position du robot est postée périodiquement dans la mémoire partagée du niveau physique. A chaque instant il compare la position du robot avec la position du but à atteindre. Au moment où le but est atteint, il poste un événement permettant au contrôleur d'exécution de savoir que le but est atteint, puis s'arrête automatiquement. Son fonctionnement sera détaillé dans le chapitre 10.

Le contrôleur d'exécution

Le rôle du contrôleur est de superviser l'exécution d'une action. Il est actif dès qu'une action se trouvant dans la mémoire partagée du niveau physique n'a pas été exécutée. Ce module gère la réactivité tactique puisqu'en fonction du contexte (c-à-d les événements présents dans la mémoire partagée) et de l'action à réaliser, il active les comportements appropriés. Selon les actions :

- Si l'action est un déplacement, à ce moment-là, il active le planificateur de chemins. Dès que le premier sous-but à atteindre est posté dans la mémoire partagée par le planificateur de chemins, il active le navigateur afin d'atteindre le sous-but et le localisateur afin de détecter quand le sous-but est atteint. A chaque fois que la position du robot est postée par le localisateur dans la mémoire partagée, il demande au navigateur de continuer à atteindre le but. En effet, il se peut que le robot atteigne et dépasse le but sans le détecter. Ceci peut arriver comme nous le verrons dans le chapitre 11 dans le cas où l'interprétation des balises reconnues est mauvaise ou dans le cas où la position initiale du robot est fausse et le localisateur pense ne toujours pas avoir atteint son but. Dans ce cas, en continuant à demander au navigateur d'atteindre le but, le navigateur va se replacer dans la direction du but, et à nouveau déplacer le robot vers le but. C'est ainsi que la prise en compte de la réactivité tactique est assurée. Tant que le but n'est pas atteint à chaque fois que le localisateur poste la position du robot, le contrôleur demande au navigateur de vérifier que le robot se trouve toujours dans la direction du but, et dans le cas contraire, le navigateur va replacer le robot face à son but. Une fois le but final atteint, il poste le compte rendu d'exécution à la mémoire partagée du niveau cognitif. Dans le cas, où un des sous-buts ne peut être atteint, il va tout d'abord replanifier un chemin pour atteindre le but final. Si lors du déplacement du robot, ce chemin ne lui permet pas d'atteindre son but, alors il poste le compte-rendu d'exécution au niveau cognitif afin que celui-ci replanifie la mission. Au cours de l'exécution d'un déplacement, il poste dans la mémoire partagée du niveau cognitif les informations perceptuelles sur l'état de l'environnement postées dans la mémoire partagée du niveau physique par le navigateur ou le localisateur. Ces informations sont de deux types et peuvent avoir une influence sur la mission, elles doivent donc être transmises au niveau cognitif. Lorsque le robot découvre un obstacle inconnu, le navigateur poste la position de cet obstacle dans la mémoire partagée. Au fur et à mesure que le robot se déplace et interprète les données des capteurs, et reconnait les balises correspondantes. L'état de ces balises (visibles ou non visibles) peut avoir une influence sur l'exécution de la mission. Le localisateur poste donc l'état des balises qu'il reconnait dans la mémoire partagée du niveau physique et le contrôleur se charge de les transmettre au niveau cognitif. Nous reviendrons sur la façon dont le contrôleur d'exécution d'actions gère l'exécution d'un déplacement dans le chapitre 11 une fois que nous aurons expliqué le fonctionnement du module de localisation.

– Si l'action est une saisie ou un rendu d'objets, à ce moment-là, il active le module de saisie/rendu d'objets et attend son compte-rendu d'exécution qu'il poste dans la mémoire partagée du niveau cognitif.

4.4.5 Exemple

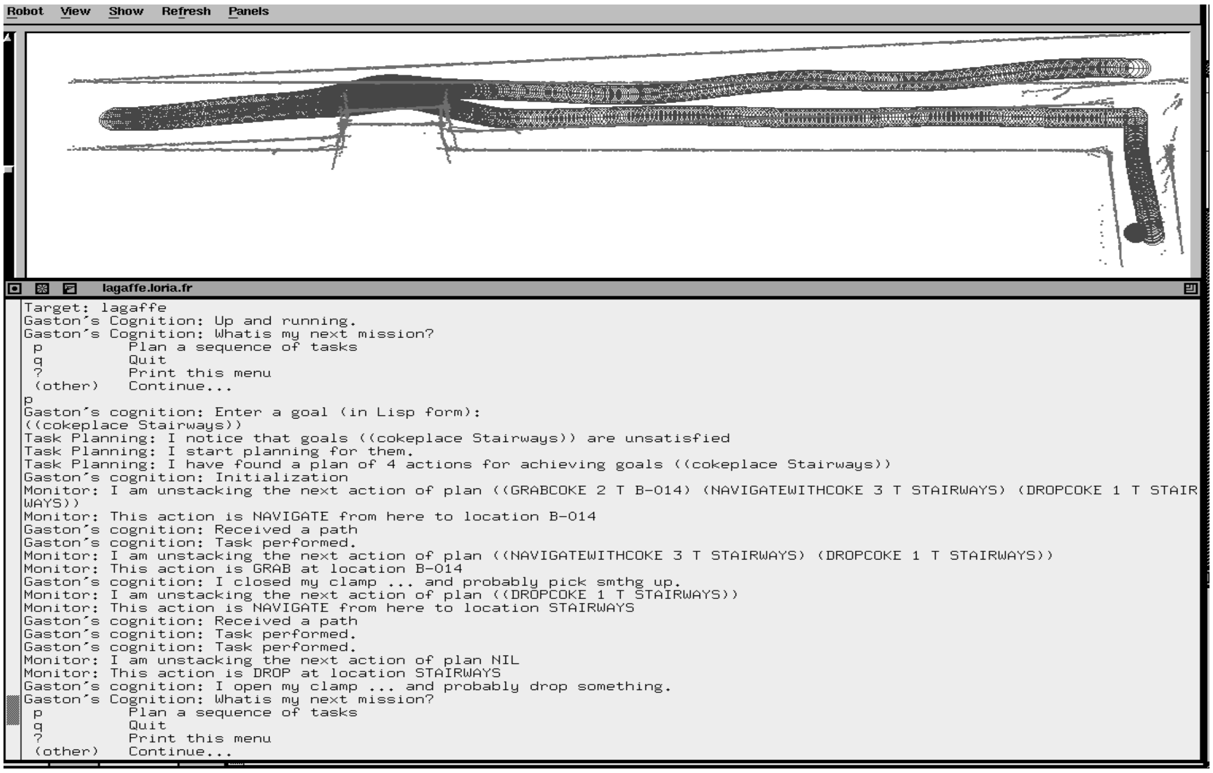


FIG. 4.11 – *Trace d'exécution du déplacement d'un objet*

Nous avons testé notre architecture à deux niveaux dans des scénarios de manipulations d'objets au premier étage de notre batiment (figure 4.11). Le robot évolue dans des conditions réelles et peut rencontrer des objets *non* mentionnés dans la carte tels que des étagères, des extincteurs ou des personnes. Une mission est postée par l'utilisateur dans la mémoire partagée du niveau cognitif : typiquement, apporter un objet à un endroit précis, la position de l'objet étant connu. Cet événement provoque l'activation du contrôleur du niveau cognitif, qui lui-même active le planificateur de tâches. Dès que la première action de la mission est planifiée (c-à-d se déplacer vers l'endroit où se trouve l'objet), le contrôleur poste cette action dans la mémoire partagée du niveau physique. Les trois autres actions (c-à-d prendre l'objet, se déplacer vers le but puis rendre l'objet) composant le plan seront postées dans la mémoire partagée du niveau cognitif au fur et à mesure qu'elles sont planifiées par le planificateur de missions. Le contrôleur d'exécution des actions du niveau physique active le planificateur de chemins pour trouver un chemin permettant au robot d'atteindre son but. Lorsque le premier but est trouvé, le contrôleur active le navigateur et le localisateur, pour superviser et exécuter le déplacement du robot. En cours de déplacement, le robot poste dans la mémoire partagée des événements relatifs à l'interprétation qu'il fait de son environnement, tels que : porte ouverte sur la droite, ou obstacle imprévu devant. Une fois, le but atteint et donc l'action exécutée et son résultat posté dans la mémoire partagée du niveau physique par le localisateur, le résultat est posté dans la mémoire partagée du niveau cognitif par le moniteur du niveau physique. Le moniteur décide en fonction de la réponse postée de la suite de la mission, et relance le processus jusqu'à ce que les trois autres actions composant la mission soient exécutées.

4.5 Conclusion

Dans ce chapitre, nous avons défini un modèle d'architecture basée sur Aibot. La décomposition du contrôle sous forme de niveaux permet de regrouper les diverses fonctions opératoires en fonction du type de données qu'elles manipulent. Chaque niveau possède sa propre autonomie, et est réactif aux nouveaux événements qu'il reçoit, ce qui permet un développement indépendant de chaque niveau. De plus, chaque niveau a une structure identique : un contrôleur, un planificateur, une mémoire partagée ainsi qu'un ensemble de modules spécifiques pour réaliser les tâches auxquelles le niveau est dédié. La hiérarchie des niveaux est construite de manière incrémentale, en commençant par des niveaux gérant la réactivité réflexe jusqu'à des niveaux gérant la réactivité stratégique. Chaque niveau correspond à un raffinement de la tâche : les niveaux du haut de la hiérarchie planifient la mission et travaillent sur des données symboliques parfaites et les niveaux les plus bas exécutent des actions, sont en contact avec l'environnement et manipulent des données bruitées. La circulation des informations se fait de manière ascendante et descendante entre deux niveaux consécutifs. Le niveau le plus haut dans la hiérarchie envoie des actions à exécuter au niveau juste inférieur, et le niveau inférieur renvoie le résultat de l'exécution de cette action.

Dans notre application, nous avons utilisé deux niveaux :

- Un niveau cognitif qui manipule des données symboliques. Le planificateur de ce niveau est un planificateur de missions, qui interprète et décompose en actions élémentaires les missions du robot. Ce niveau envoie des descriptions des actions à effectuer au niveau inférieur.
- Un niveau physique qui manipule des données numériques. Le planificateur de ce niveau est un planificateur de chemins, qui fournit au robot une liste de sous-buts à atteindre pour se déplacer de sa position initiale vers un but donné. Ce niveau comprend un module de navigation (détaillé dans le dernier chapitre du mémoire), qui à partir des perceptions du robot, renvoie une action motrice visant à le rapprocher de son but tout en évitant les obstacles. Il comprend aussi un module de localisation (détaillé dans la deuxième partie du mémoire) qui permet au robot de se repérer dans son environnement au fur et à mesure de ses déplacements et de tenir compte des modifications de l'environnement. Ce module est une application

directe de notre approche pour apprendre et reconnaître des balises naturelles.

Grâce à la construction incrémentale des niveaux, nous pourrions ajouter un troisième niveau qui serait chargé de la coordination entre plusieurs robots. Dans ce niveau, le planificateur serait un planificateur de missions, qui permettrait d'interpréter et de décomposer en actions élémentaires les missions des robots, et de les distribuer à chacun d'eux. Le contrôleur permettrait la gestion des résultats des actions effectuées ainsi que la gestion des conflits entre robots.

5

La navigation

5.1 Introduction

Dans les deux derniers chapitres de cette partie, nous nous intéressons au module de navigation, composante essentielle du niveau physique. En effet, pour pouvoir réaliser des missions dans son environnement, un robot mobile doit être capable de s'y déplacer. La navigation permet à un robot mobile ayant une trajectoire à suivre ou un ensemble de sous-buts à atteindre de se déplacer d'un lieu à un autre dans son environnement, tout en évitant les obstacles imprévus. Le but des recherches en navigation est de trouver un modèle qui possède le but à atteindre et les données des capteurs, délivre la meilleure action sur les moteurs. La conception d'un tel modèle pose de nombreux problèmes :

- L'environnement est vaste et dynamique. Les obstacles (ou les personnes) peuvent bouger, apparaître ou disparaître. L'ensemble des situations possibles ne peut être calculé. Le robot mobile est équipé de capteurs (caméras, ultrasons ou infrarouges) pour avoir des informations sur son environnement local et agir avec ces informations. Malheureusement, les mesures de ces capteurs sont bruitées et dépendent de l'humidité ou de la température de l'air, de la forme et du type d'objets inclus dans l'environnement.
- A cause du type de sol et du glissement des roues, les actions du robot ne sont pas complètement fiables.
- Etant donné que le robot bouge dans son environnement et que son environnement évolue, il doit réagir en un temps fini (souvent court) à un nouvel événement. Un module de navigation doit donc être le plus simple possible et obéir à un cycle d'exécution très court du type : Perception, Décision et Action. Il n'a pas le temps de construire un modèle de l'environnement pour définir la meilleure action à effectuer. Il raisonne uniquement sur les perceptions au temps t.

Ce chapitre décrit les trois techniques habituellement utilisées pour réaliser un module de navigation réactive :

- Les approches à base de champ de potentiel. Dans ces approches, les objectifs de navigation (atteindre le but, tout en évitant les obstacles) sont codés dans une fonction.
- Les approches à base de réseau de neurones. Dans ces approches, le système de

navigation doit apprendre lui-même comment éviter les obstacles et atteindre un but prédéfini.
- Le contrôle flou. Dans ces approches, on définit la manière de contrôler le robot, sous forme de règles.

Chaque approche est illustrée par des exemples d'application en robotique mobile.

Cette description nous permet de mettre en avant les critères pour réaliser un module de navigation réactive et de spécifier notre approche.

5.2 Les approches à base de champs de potentiels

Le principe général de la commande par potentiels a été formulé à l'origine par [Khatib, 1986] pour le contrôle d'un bras manipulateur. Le but de cette approche est de coder les objectifs de navigation dans une fonction :
- atteindre le but.
- éviter les obstacles.

Cette fonction, dont la valeur dépend de la configuration du robot, doit décroître lorsque l'on se rapproche de l'objectif et croître si le comportement du robot est contraire aux spécifications. Le principe de la navigation par potentiel consiste à créer un contrôleur dont les commandes auront pour but d'amener le robot dans une configuration minimisant une fonction. Cette minimisation assure la réussite de la tâche à accomplir. Le problème est donc un problème d'optimisation de fonction. En fait, on considère que le robot est un point dans l'environnement évoluant sous l'influence d'un champ de potentiel artificiel U dont les variations locales reflètent la structure de l'espace libre[7]. La fonction de potentiel est souvent (mais pas obligatoirement) définie comme la somme d'un potentiel attractif attirant le robot vers son but et d'un potentiel répulsif repoussant le robot loin des obstacles. A chaque instant, la force artificielle $\vec{F}(q)$ = -$\vec{\nabla}$U(q) induite par la fonction de potentiel permet de trouver le déplacement le plus approprié, et le calcul de cette force artificielle en chaque point conduit le robot vers son but. Afin que le robot soit attiré par son but, et repoussé par les obstacles, U est construite comme la somme de deux fonctions de potentiels élémentaires :
U(q) = $U_{att}(q) + U_{rep}(q)$
où U_{att} est le potentiel attractif associé au but q_{goal} et U_{rep} est le potentiel répulsif associé aux obstacles. U_{att} est indépendant des obstacles et U_{rep} est indépendant du but. Avec ces notations, $\vec{F}$ est la somme des deux vecteurs :
$\vec{F_{att}} = -\vec{\nabla} U_{att}$ et $\vec{F}_{rep} = -\vec{\nabla} U_{rep}$
qui sont appelés respectivement les forces attractives et répulsives.

5.2.1 Le potentiel attractif

Le champ de potentiel attractif U_{att} est défini par une fonction parabolique :
$U_{att} = \frac{1}{2}\epsilon\rho_{goal}^2$ (q)

7. Nous définissons l'espace libre par opposition à l'espace encombré, comme la partie de l'espace qui n'est pas encombrée d'obstacles

où ϵ est un facteur d'échelle positif et $\rho_{goal}(q)$ représente la distance euclidienne ||q - q_{goal}||. La fonction U_{att} est positive ou nulle, et atteint son minimum à q_{goal}, où $U_{att}(q_{goal}) = 0$. La fonction ρ_{goal} est dérivable en chaque point de l'environnement. A chaque configuration q, la force artificielle attractive dérivée de U_{att} est :

$$\begin{aligned}\vec{F}_{att}(q) &= -\vec{\nabla} U_{att}(q)\\ &= -\epsilon\rho_{goal}(q)\vec{\nabla}\rho_{goal}(q)\\ &= -\epsilon(q - q_{goal})\end{aligned}$$

5.2.2 Le potentiel répulsif

Le champ de potentiel répulsif est défini par :

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho(q)} - \frac{1}{\rho_0})^2 & \text{si } \rho(q) \leq \rho_0 \\ 0 & \text{si } \rho(q) > \rho_0 \end{cases}$$

où η est un facteur d'échelle positif, $\rho(q)$ représente la distance entre q et l'obstacle le plus proche, et ρ_0 est une constante positive appelée distance d'influence des obstacles. La fonction U_{rep} est positive ou nulle, et tend vers l'infini quand q est proche d'un obstacle, et est nulle quand la distance entre q et l'obstacle le plus proche est plus grande que ρ_0. La fonction U_{rep} est dérivable en chaque point de l'espace libre. La force artificielle de répulsion dérivée de U_{rep} est :

$$\vec{F}_{rep}(q) = -\vec{\nabla} U_{rep}(q) = \begin{cases} \eta(\frac{1}{\rho(q)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q)}\vec{\nabla}\rho(q) & \text{si } \rho(q) \leq \rho_0 \\ 0 & \text{si } \rho(q) > \rho_0 \end{cases}$$

5.2.3 Exemple

La figure 5.1 montre un champ de potentiel attractif (b), un champ de potentiel répulsif (c) et la somme des deux (d) dans un espace a deux dimensions contenant deux obstacles (a). La figure (e) montre à la fois des contours équipotentiels du potentiel total, et un chemin généré en suivant le gradient négatif de cette fonction. La figure (f) montre une matrice des orientations de vecteurs de gradient négatif sur l'espace libre.

5.2.4 Utilisation pour la navigation réactive d'un robot mobile

Le problème principal d'une telle approche réside dans l'existence de minima locaux. Ce problème peut être abordé de deux manières différentes :

- en construisant une fonction de potentiel dont le seul minimum est le but à atteindre.
- en utilisant une fonction de potentiel possédant des minima locaux mais en mettant en place un ensemble de mécanismes permettant au robot de les éviter ou de repartir s'il en est prisonnier [Latombe, 1991].

Koditschek [Koditschek, 1987] a montré qu'en général une fonction de navigation globale, c'est-à-dire une fonction de potentiel possédant comme seul minimum le but à atteindre, n'existe pas. Pour pallier ce problème diverses solutions ont été proposées.

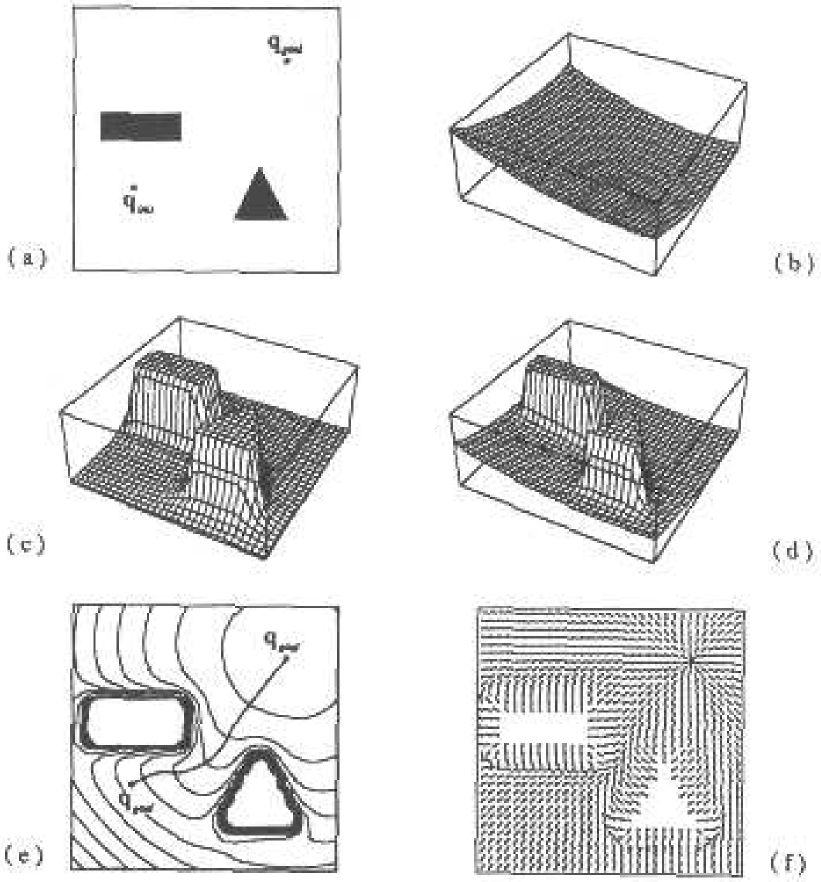


FIG. 5.1 – *Recherche d'un chemin grâce à la méthode des champs de potentiels [Latombe, 1991]*

L'approche proposée par [Krogh et Thorpe, 1986] ne nécessite pas l'utilisation d'une telle fonction de navigation. Elle se base plutôt sur la constatation qu'un environnement simple est moins susceptible de produire des minimums locaux qu'un environnement complexe. Le principe consiste à faciliter le travail du système réactif en décomposant le chemin global en un ensemble de sous-buts proches à rejoindre. Ces sous-buts sont calculés grâce à un planificateur global [Thorpe, 1984] à partir d'un modèle de l'environnement. Ce modèle peut être incomplet, le rôle du système réactif étant bien entendu de gérer ces imperfections. L'originalité de cette approche réside dans la fonction de potentiel choisie. Cette fonction, afin de prendre en compte la dynamique du système, dépend non seulement de la configuration du robot mais également de sa vitesse.

Dans une méthode faisant un lien direct entre perception et action, [Borenstein et Koren, 1989] utilise les champs de forces virtuels. Le but est de permettre à un robot mobile “rapide” de contourner un obstacle imprévu détecté par ses capteurs. Les mesures des différents capteurs sont intégrées dans une grille appelé *grille histogramme*. La différence avec [Elfes, 1989] est que seule la cellule correspondant à la mesure d'un capteur a son degré d'activation incrémenté de manière à obtenir une mise à jour plus rapide.

La grille ainsi calculée représente une carte globale dans un repère absolu de l'environnement. Lorsque le robot se déplace de manière à limiter son champ de vision, seuls les échos capteurs situés dans son environnement proche (délimité par une fenêtre de taille fixe) sont considérés. Chaque cellule de la grille exerce sur le véhicule une force dirigée vers le robot et dont l'amplitude est fonction du degré d'activation de la cellule et de la distance la séparant du véhicule. Simultanément à cette force de répulsion, le robot est également soumis à une force d'attraction provenant du but.

La force totale est la somme de toutes les forces de répulsion et de la force d'attraction. Sa direction indique la direction devant être suivie par le robot. Sa vitesse linéaire est fonction de l'angle entre le vecteur vitesse courant et le vecteur somme des forces de répulsion.

5.2.5 Conclusion

Les trois approches principales du concept de champ de potentiel sont : la méthode classique [Khatib, 1986], la méthode généralisée [Krogh et Thorpe, 1986] et la méthode des champs de force [Borenstein et Koren, 1989].

Les principaux avantages des méthodes à base de champ de potentiels sont :

- Une mise en place de l'algorithme relativement simple.
- Une réactivité du robot importante.
- Une force résultante continue.
- Un traitement facile des obstacles mobiles et immobiles.

Son principal inconvénient est l'existence de minimums locaux où les forces de répulsion et d'attraction s'annulent. Cette situation se produit généralement dans deux situations : lorsque le robot doit passer dans un passage étroit où lorsqu'il y a entre le robot et le but un obstacle en forme de U. Un autre inconvénient est l'oscillation de la trajectoire du robot due aux rapides changements de direction de la force appliquée au robot dans des zones où les forces de chaque côté du robot s'équilibrent.

On constate aussi que cette méthode peut s'appliquer autant à la planification de chemins qu'à la navigation réactive.

5.3 Les approches à base de réseaux de neurones

Comme nous l'avons vu précédemment, il est difficile de trouver une fonction qui permette d'associer aux observations du robot la meilleure action possible. Il semble donc intéressant et prometteur de confier au système de navigation le soin d'apprendre lui-même comment réaliser au mieux cette tâche. Les réseaux de neurones ont de fortes capacités d'apprentissage et de classification. Ils peuvent donc être appliqués pour la réalisation d'une fonction de navigation.

5.3.1 Le neurone formel

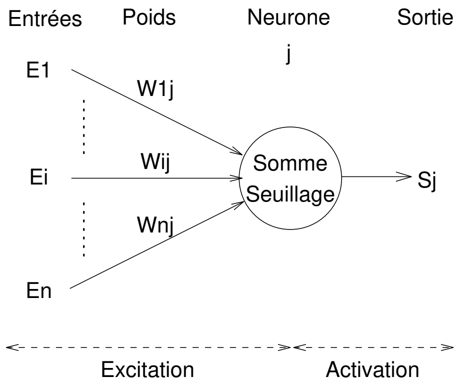


FIG. 5.2 – *Le neurone formel*

Dans un neurone formel (voir Figure 5.2), la sortie S_j est définie par la formule:

$$S_j = F_j(-\theta_j + \sum_{i=1}^{n} W_{ij}.E_i) \tag{5.1}$$

où n est le nombre d'entrées, E_i la i^{eme} entrée, W_{ij} le poids de l'entrée i, θ_j le seuil d'activation, et F_j la fonction d'activation du neurone j. On peut intégrer le seuil θ_j dans la somme, comme poids supplémentaire correspondant à une "entrée" fictive toujours à 1.

Les fonctions d'activation (ou "de seuillage") les plus courantes sont celles indiquées sur la Figure 5.3, où x est l'activité brute (ou excitation) $\sum_{i=1}^{n} W_{ij}.E_i$ et $F(x)$ l'activité après seuillage (ou sortie).

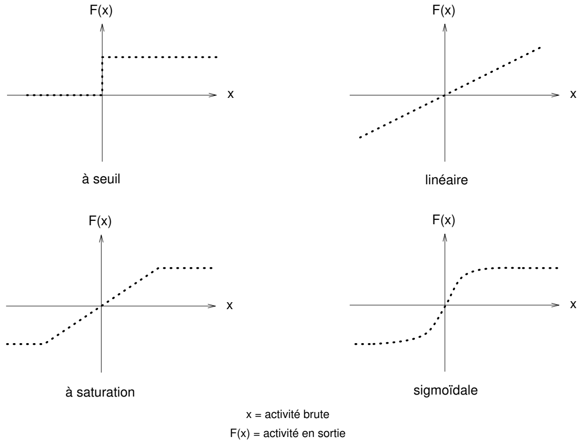


FIG. 5.3 – *Types de fonctions d'activation*

5.3.2 Les techniques d'apprentissage et leur utilisation en Robotique mobile

Apprentissage supervisé

L'apprentissage supervisé consiste à trouver une fonction f qui à partir d'un ensemble $(\vec{x_1}, \vec{y_1})$,..., $(\vec{x_n}, \vec{y_n})$ d'exemples de couples (entrée, sortie) vérifie les propriétés suivantes :

- f satisfait les exemples : $\forall i \in \{1,...,n\}$ f$(\vec{x_i}) = \vec{y_i}$
- f possède de "bonnes" propriétés de généralisation :
  $\forall x, \forall i, \vec{x} \neq \vec{x_i}, \vec{y} = f(\vec{x})$
  $\vec{y}$ étant une commande permettant de satisfaire l'objectif imposé par la tâche de navigation. Cette propriété de généralisation est capitale dans le type de problème que nous traitons car l'espace est vaste et seule une petite sous-partie pourra être présentée sous forme d'exemples.

L'apprentissage supervisé est un apprentissage généralement exécuté hors-ligne. Cet apprentissage est peu utilisé en robotique. En effet, il nécessite un corpus d'apprentissage qui, même s'il ne représente qu'une petite partie de l'(énorme) ensemble des couples possibles, reste très important, et donc difficile à constituer. On citera quand même le système ALVINN[8] développé par [Pomerleau, 1992] dans le cadre de la conduite automatique d'un véhicule de type voiture sur une route.

Apprentissage par renforcement

C'est une forme particulière d'apprentissage supervisé, où le rôle du "professeur" est minimisé. Au lieu de devoir disposer d'un ensemble de couples (entrée, sortie désirée), il suffit d'avoir une fonction de renforcement, ou fonction de coût. Elle va agir en tant que "critique" de la sortie obtenue, en évaluant dynamiquement la qualité de cette dernière,

8. Autonomous Land Vehicule In a Neural Network

sans toutefois connaître la sortie désirée exacte. Les phases d'apprentissage et d'exploitation sont confondues, la qualité de l'exploitation augmentant au fur et à mesure de l'apprentissage.

Cet apprentissage, quoique toujours supervisé par le critique, est cependant autonome une fois ce dernier implanté. De plus, la plausibilité comportementale [Sutton et Barto, 1981] de cette technique est très supérieure à celle de l'apprentissage supervisé pur. Enfin, elle peut permettre un apprentissage plus complet que l'apprentissage supervisé (cf. l'étude [Tesauro, 1992] sur un réseau entraîné à jouer au backgammon).

Cette technique d'apprentissage en ligne est donc très appréciée en robotique mobile et de nombreux systèmes l'utilisent, avec un succès certain.

Les systèmes réalisés avec cette technique ayant eu les résultats les plus intéressants avaient des objectifs divers, tels que pousser des caisses [Mahadevan et Connell, 1991], aller se brancher à une prise de courant [Lin, 1993], ou suivre un chemin en se dirigeant de sous-but en sous-but [Chang et Hsu, 1994].

Apprentissage par différence temporelle

Il s'agit d'une technique particulière d'apprentissage par renforcement (voir l'article fondateur [Sutton, 1988]), dans laquelle les informations nécessaires au "critique" ne sont pas connues à l'instant t où l'entrée se présente, mais plus tard, à l'instant $t+d$, avec $d \geq 1$. Ainsi, le renforcement attribué à une étape $t + d$ agit sur toutes les étapes précédentes (t à $t + d$), ce qui permet une prise en compte temporelle, sur la suite complète des choix du réseau.

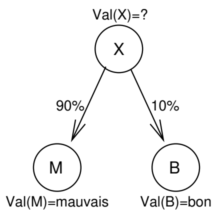


FIG. 5.4 – *Utilisation d'une différence temporelle*

Sur la figure 5.4, cet apprentissage permet d'associer la valeur "mauvais" à l'état X, qui mène la plupart du temps a l'état mauvais M, alors qu'un apprentissage sans différence temporelle ne prendrait pas en compte M.

L'intérêt de cette technique est qu'elle permet d'apprendre les poids correspondant à des séquences de sorties liées temporellement [Tesauro, 1992]. Ceci est évidemment très intéressant pour le guidage en robotique, car seules des séries d'actions consécutives peuvent permettre d'atteindre un objectif. Cependant, cette voie semble avoir été encore assez peu explorée.

Apprentissage non supervisé

Il consiste à extraire des informations à partir de données non étiquetées, de façon totalement autonome, afin d'apprendre les poids adéquats. Le réseau doit donc effectuer des regroupements sur ses entrées, trouver des similarités locales, à partir de critères prédéfinis. Les phases d'apprentissage et d'exploitation du réseau sont donc consécutives.

Les systèmes employant cette technique sont ceux basés sur des architectures à base de cartes topologiques auto-adaptatives de type Kohonen, tels que ceux réalisés dans [Zimmer, 1995b], [Zimmer, 1995a] et [Gaussier, 1994].

5.3.3 Conclusion

Les techniques à base de réseau de neurones sont basées sur l'apprentissage de la fonction de navigation. Nous avons présenté dans cette section l'ensemble des techniques d'apprentissage et leurs applications en robotique. L'apprentissage peut être réalisé de façon supervisé ou non supervisé. L'apprentissage supervisé donne de bons résultats, mais doit être exécutée hors ligne, et est donc assez peu utilisé en robotique. Les méthodes par renforcement et les méthodes non supervisées donne de bons résultats. Les travaux sur l'apprentissage par différence temporelle sont très rares et donnent des résultats très modestes.

L'utilisation de réseau de neurones pour réaliser un module de navigation est très récente. Cette technique n'a pas encore été utilisée autant que les approches à base de champ de potentiel ou le contrôle flou, et de nombreuses voies n'ont pas encore été explorées. Il est donc difficile de porter un jugement. Bien que les premiers résultats soient prometteurs ils restent encore à l'heure actuelle en dessous de ceux des deux autres types d'approches.

5.4 La logique floue et le contrôle flou

Les deux approches précédentes (section 5.2 et section 5.3) cherchent à définir une fonction qui possédant les données des capteurs, retourne une action motrice. Il existe une troisième approche qui cherche à définir la manière de contrôler le robot : le contrôle flou. Les données des capteurs étant bruitées et les actions motrices pas complétement fiables, le contrôle flou est une façon appropriée de contrôler un robot mobile. En effet, le contrôle flou [Lee, 1990] est une application de la logique floue au contrôle de systèmes dynamiques. [Zadeh, 1965] définit la logique floue comme une extension de la logique classique pour la représentation et le raisonnement sur des données approximatives. L'idée de Zadeh, à travers le nouveau concept ensembliste d'appartenance graduelle d'un élément à un ensemble, a été de définir une logique multivaluée permettant de modéliser ces imperfections, c'est-à-dire prendre en compte les états intermédiaires entre le tout et le rien.

Un contrôleur flou va décrire la façon de contrôler un processus, tout comme le ferait un expert humain, à travers des règles de la forme "**si** *prémisse* **alors** *conclusion*" intégrant naturellement imprécisions et incertitudes.

De nombreuses applications en robotique mobile sont fondées sur le contrôle flou. Par exemple, [Reignier, 1994][Song et Tai, 1992] utilisent le contrôle flou pour contrôler un robot mobile dans un environnement intérieur structuré inconnu.

Dans cette section, nous présentons la logique floue et le contrôle flou, et leurs applications en robotique mobile.

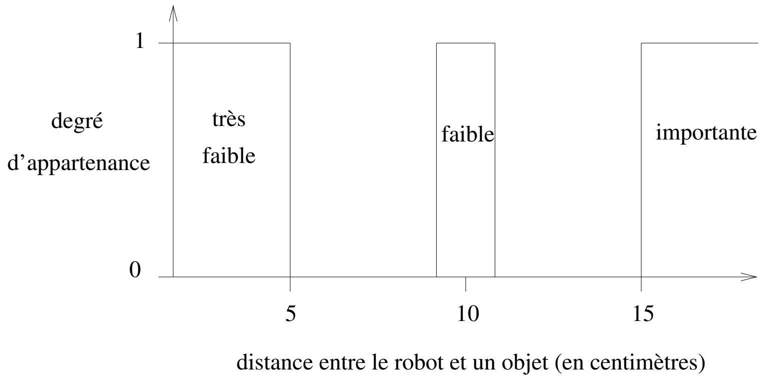


FIG. 5.5 – *Exemple de définition d'ensembles sur un univers de discours en logique binaire*

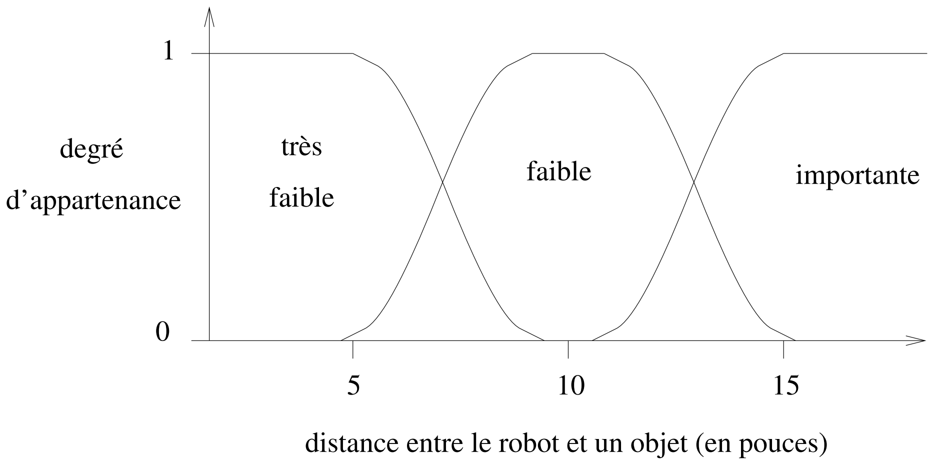


FIG. 5.6 – *Exemple de définition d'ensembles sur un univers de discours en logique floue*

5.4.1 La logique floue

Les sous-ensembles flous

La théorie des sous-ensembles flous[9] (et par extension, la logique floue), s'appuie sur la notion de *degré d'appartenance* d'un élément à un sous-ensemble flou. Tandis que les ensembles traditionnels sont caractérisés par une *fonction d'appartenance*, notée χ, (également appelée fonction caractéristique) définie sur $\{0,1\}$, les sous-ensembles flous sont, eux, caractérisés par une fonction d'appartenance, notée μ, définie sur $[0, 1]$. En d'autres termes, dans le langage ensembliste classique, un élément appartient ou n'appartient pas à un ensemble tandis qu'un élément appartient à un sous-ensemble flou avec un certain

9. Le terme de sous-ensemble flou provient du fait que celui-ci est considéré comme une partie d'un univers de discours U. Dans la littérature, on peut trouver parfois *ensemble flou*, qui constitue un abus de langage.

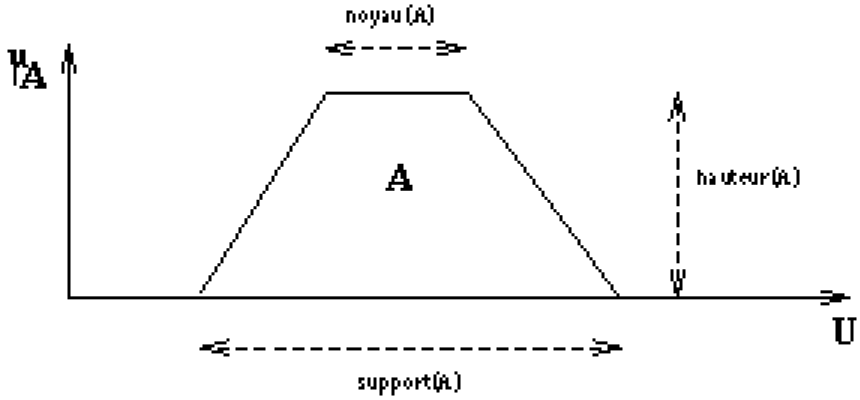


FIG. 5.7 – *Représentation d'un sous-ensemble flou et principales caractéristiques*

degré (éventuellement nul). En résumé, pour un sous-ensemble A défini sur un univers de discours U, on peut écrire :

$$\begin{array}{lcll} A\ sous-ensemble\ classique & : & fonction\ caractéristique & \chi_A : U \rightarrow \ \{0,1\} \\ A\ sous-ensemble\ flou & : & fonction\ d'appartenance & \mu_A : U \rightarrow \ [0,1] \end{array}$$

Par extension, ce nouveau concept définit une logique multivaluée qui apparaît comme une généralisation de la logique binaire. Prenons un exemple : en logique classique, il est difficile de définir "une distance très faible entre le robot et un obstacle", "une distance faible entre le robot et un obstacle" ou "une distance importante entre le robot et un obstacle". En effet, on peut considérer qu'une distance de moins de cinq pouces entre le robot et un obstacle est très faible, une distance de plus de neuf pouces et de moins d'onze pouces entre le robot et un obstacle est faible, et qu'une distance de plus de quinze pouces entre le robot et un obstacle est importante. Mais qu'en est-il d'un obstacle situé à six pouces, huit pouces ou quatorze pouces ? Une réponse intuitive consiste à affirmer qu'ils appartiennent à une ou deux des catégories précédentes avec des niveaux (normalisés i.e. définis sur $[0,1]$[10]) différents. On évite ainsi des transitions rigides entre différentes catégories, comme cela est le cas en logique binaire (cf.figure 5.5 et 5.6). Ces figures montrent que dans certains cas une logique binaire est trop restrictive, et qu'il est nécessaire d'utiliser une logique multivaluée. La distance entre le robot et un obstacle sera ainsi considérée comme très faible, faible ou importante avec un degré d'appartenance différent selon la distance entre le robot et cet obstacle.
Nous venons de voir ce que l'on entend par sous-ensemble flou, d'un point de vue formel. Un sous-ensemble flou A sur un univers de discours U, est représenté comme dans la figure 5.7 à travers sa fonction caractéristique μ_A.

Variable linguistique

Une variable linguistique est définie par un triplet (V,U,T_V) où V représente une variable classique (âge, température,...) définie sur l'univers de discours U. T_V est l'ensemble

10. Les bornes de l'intervalle correspondent respectivement à *n'appartient pas* et *appartient totalement.*

des instanciations possibles de la variable V : il s'agit de sous-ensembles flous repérés par leur étiquette A_i : on écrit ainsi $T_V = \{A_1, A_2, \ldots, A_n\}$. Par exemple, la figure 5.6 représente la variable linguistique "distance entre le robot et un objet" où "distance entre le robot et un objet" représente V, U représente l'ensemble des distances possibles ([0, 255] dans notre cas), et $T_V = \{$Très faible, Faible, Importante$\}$.

Propositions floues

Une *proposition floue élémentaire* est définie à partir d'une variable linguistique (V, U, T_V) par la qualification "V est A", avec A appartenant à T_V. Par exemple, "La distance entre le robot et un obstacle est Faible" est une proposition élémentaire définie à partir de la variable linguistique ("Distance entre le robot et un obstacle", [0, 255], $\{\ldots$,Faible,$\ldots\}$). La valeur de vérité d'une proposition élémentaire "V est A" est égale à $\mu_A(v)$ où v correspond à la valeur numérique exacte de V.
Une *proposition floue générale* est définie à partir de propositions élémentaires et d'opérateurs logiques binaires (**et**, **ou**, **implique**) ou unaire (**non**). Il existe plusieurs méthodes pour calculer la valeur de vérité de telles propositions. Nous ne donnons ici que les plus communément utilisées.

- conjonction : (V_1 est A_1) **et** (V_2 est A_2)
  $\min(\mu_{A_1}(v_1), \mu_{A_2}(v_2))$ (Logique de Zadeh)
  $\max(\mu_{A_1}(v_1) + \mu_{A_2}(v_2) - 1, 0)$ (Logique de Lukasiewicz)
  $\mu_{A_1}(v_1).\mu_{A_2}(v_2)$ (Logique probabiliste)
- disjonction : (V_1 est A_1) **ou** (V_2 est A_2)
  $\max(\mu_{A_1}(v_1), \mu_{A_2}(v_2))$ (Logique de Zadeh)
  $\min(\mu_{A_1}(v_1) + \mu_{A_2}(v_2), 1)$ (Logique de Lukasiewicz)
  $\mu_{A_1}(v_1) + \mu_{A_2}(v_2) - \mu_{A_1}(v_1).\mu_{A_2}(v_2)$ (Logique probabiliste)
- implication : (V_1 est A_1) **implique** (V_2 est A_2)
  $\min(1 - \mu_{A_1}(v_1) + \mu_{A_2}(v_2), 1)$ (Lukasiewicz)
  $\min(\mu_{A_1}(v_1), \mu_{A_2}(v_2))$ (Mamdani)
  $\mu_{A_1}(v_1).\mu_{A_2}(v_2)$ (Larsen)
- complémentation : **non** (V est A)
  $1 - \mu_A(v)$

Dans ce qui précède, v, v_1 et v_2 correspondent à des instanciations numériques réelles des variables V, V_1 et V_2.

Règles floues

Une *règle floue* est une proposition floue générale utilisant une implication entre deux propositions floues quelconques. Par exemple :

(V_1 est A_1) **et** (V_2 est A_2) **implique** (V_3 est A_3)

ou sous une forme plus linguistique :

si (V_1 est A_1) et (V_2 est A_2) **alors** (V_3 est A_3)

est une règle floue. La partie **(V_1 est A_1) et (V_2 est A_2)** est appelée *prémisse* de la règle

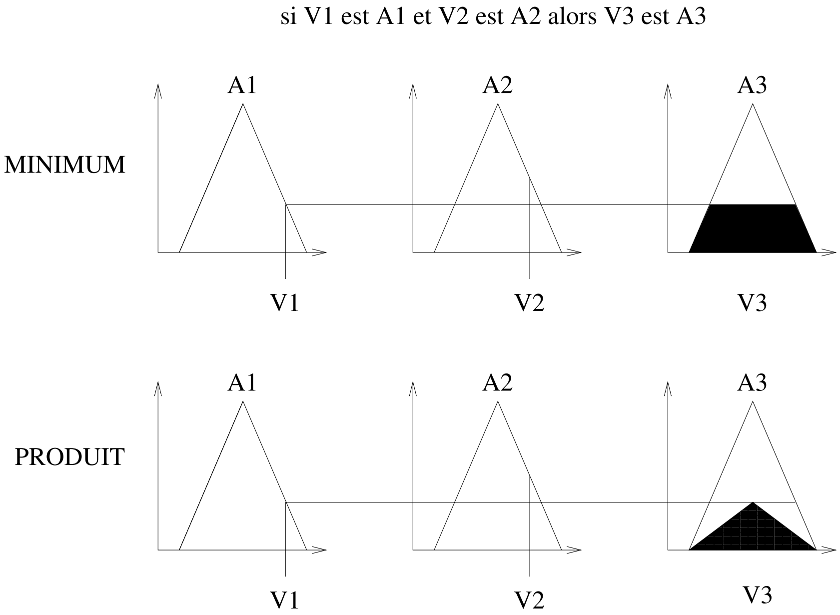


FIG. 5.8 – *Inférence "MINIMUM" et "PRODUIT"*

et (V_3 **est** A_3) *conclusion*. Les conclusions de règles peuvent être de deux sortes [Berenji, 1990] :

- les règles de type *Mamdani* où les conclusions, comme les prémisses, sont des propositions floues :

  $$\textbf{si } V_1 \textbf{ est } A_1 \textbf{ et } V_2 \textbf{ est } A_2 \textbf{ alors } V_3 \textbf{ est } A_3$$

- les règles de type *Sugeno* où dans les conclusions, une sortie est fonction des entrées :

  $$\textbf{si } V_1 \textbf{ est } A_1 \textbf{ et } V_2 \textbf{ est } A_2 \textbf{ alors } V_3 = f(V_1,V_2)$$

Une règle ne peut être activée et intervenir dans le processus d'inférence que lorsque la valeur de vérité de la proposition floue constituant sa prémisse est non nulle. Lorsqu'une règle est activée, on détermine la valeur $\mu_{A_3}(v_3)$ de la variable V_3. Les deux méthodes les plus couramment utilisées sont le MINIMUM et le PRODUIT (figure5.8). L'inférence MINIMUM [11] consiste à tronquer à la valeur $\mu_{prémisse}$ la caractérisation floue associée à une variable de sortie, alors que l'inférence PRODUIT [12] revient à affecter à celle-ci un facteur d'échelle correspondant à $\mu_{prémisse}$.

5.4.2 Architecture classique d'un contrôleur flou (FLC)

L'architecture classique d'un contrôleur flou (FLC [13]), proposée par Mamdani [Mamdani, 1976], est illustrée dans la figure 5.9 tirée de [Berenji, 1990].

Comme nous pouvons le voir dans la figure 5.9, un contrôleur flou est composé de quatre parties :

1. ***la fuzzification*** [14] : les valeurs délivrées par les capteurs du robot, décrivant l'état

11. Correspond à l'implication de Mamdani (cf.précédemment).
12. Correspond à l'implication de Larsen (cf. précédemment).
13. Acronyme de Fuzzy Logic Controller.
14. Faute de trouver une traduction correcte, nous garderons cet anglicisme que nous préférons à flouage

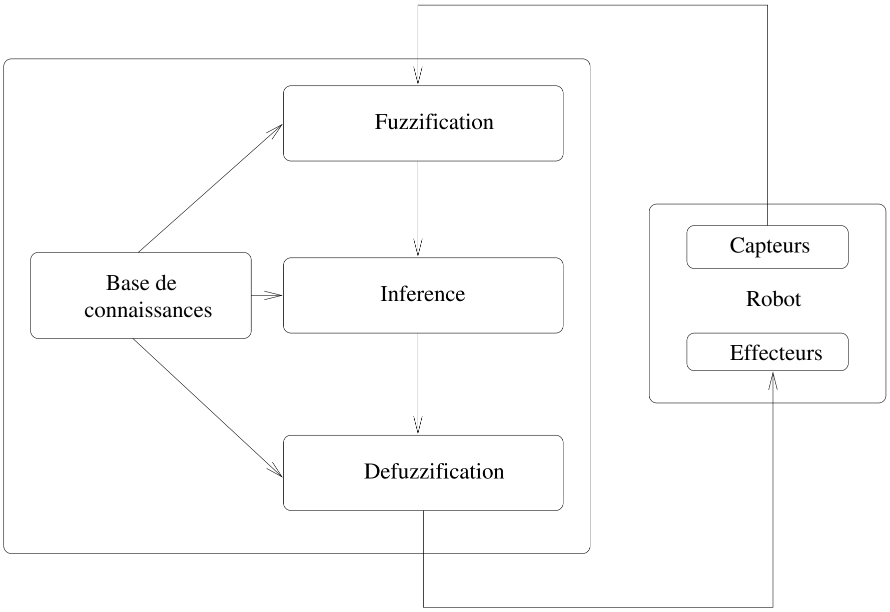


FIG. 5.9 – *Architecture d'un contrôleur flou*

du système contrôlé (en l'occurrence le robot), sont traduites en étiquettes de sous-ensembles flous caractérisant les variables linguistiques associées.

2. ***la base de connaissances***: celle-ci est composée de:
   - *variables linguistiques* permettant de caractériser les états associés au processus contrôlé;
   - *règles linguistiques* codant la connaissance que l'on a sur le contrôle de ce processus.
3. ***l'inférence sur les règles***: cette étape représente le raisonnement du contrôleur effectué par le moteur d'inférences.
4. ***la défuzzification***[13] : il s'agit de la partie duale de la fuzzification, consistant à combiner les sorties des règles calculées à l'étape précédente, dans le but de déterminer les commandes effectives du robot.

Nous allons maintenant reprendre plus en détail ces différentes parties, afin de montrer le rôle de chacune dans l'architecture du contrôleur flou.

Fuzzification

L'étape de fuzzification assure la transformation permettant de passer:
- d'un niveau discret à un niveau continu;
- d'un niveau numérique/quantitatif à un niveau symbolique/qualitatif [Bouchon-Meunier, 1994; Berenji, 1990].

Chaque instanciation d'une variable d'entrée réelle du contrôleur va être traduite en une étiquette caractérisant la variable linguistique associée. Par exemple, une distance de 2,5

et déflouage.

pouces sera traduite en très faible, étiquette de la variable linguistique “distance en centimètre entre le robot et l'objet” {Très faible, Faible, Important}). Ainsi une information ne sera plus caractérisée par une valeur numérique, mais par un terme linguistique comme Très faible, si l'on considère une distance en centimètre par exemple. Cependant, le degré d'appartenance $\mu_A(v)$ d'une valeur réelle v à un sous-ensemble flou A doit être mémorisé car celui-ci interviendra dans la détermination des valeurs de vérité des propositions utilisant A.

La figure 5.10 illustre le mécanisme de fuzzification pour deux éléments appartenant au domaine des distances entre le robot et un objet.

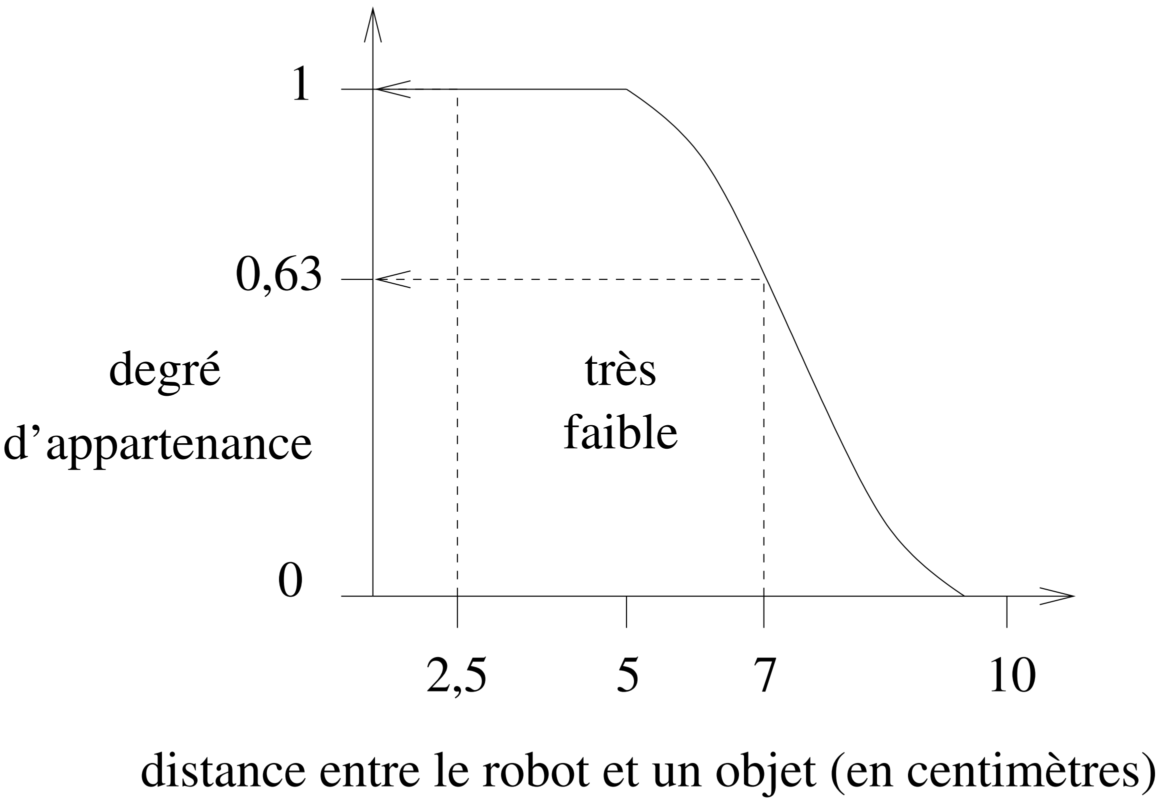


FIG. 5.10 – *Mécanisme de fuzzification*

Ainsi deux distances entre le robot et un objet respectivement de 2,5 et 7 centimètres appartiennent toutes deux à la catégorie “Très faible” avec cependant des degrés d'appartenance différents : $\mu_{Distance}(2{,}5) = 1$ et $\mu_{Distance}(7) = 0.63$.

Base de connaissances

Variables linguistiques

Une des premières tâches qui incombent aux concepteurs de systèmes flous est, après avoir déterminé les variables liées au système à contrôler, de définir les variables linguistiques associées. Ainsi, pour chaque variable, il s'agit de discrétiser le domaine de variation en sous-ensembles flous.

Règles linguistiques

Les règles linguistiques sont des règles floues dont la forme générale est la suivante :

"**si** *prémisse* **alors** *conclusion*"

où les *prémisses* portent sur les entrées du contrôleur flou et les *conclusions* sur les sorties. Comme nous l'avons vu précédemment, ces règles sont en fait des propositions floues incluant un opérateur d'implication.

Inférence

Le mécanisme d'inférence consiste à déterminer les règles floues activées (i.e. les règles dont le degré d'activation $\mu_{prémisse}$ issu des prémisses est non nul). Au niveau de chacune de ces règles, ce degré va permettre de déterminer une valeur floue pour la(les) variable(s) de sortie apparaissant dans la partie "conclusion". Ce degré d'activation est calculé en utilisant les formules décrites précédemment sur la détermination des valeurs de vérité des propositions floues.

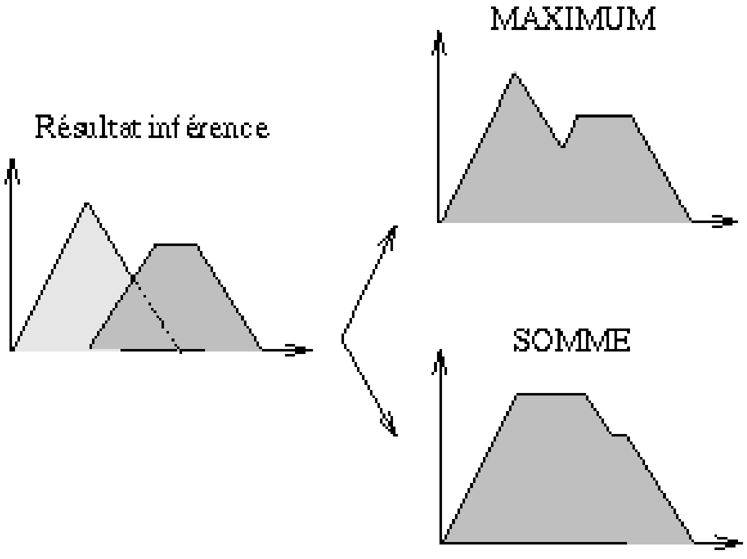


FIG. 5.11 – *Compositions de valeurs floues issues de l'inférence*

Défuzzification

a) Composition

Une fois la phase d'inférence terminée, pour chaque variable linguistique de sortie du contrôleur, il s'agit de regrouper les caractérisations floues issues de l'inférence pour en obtenir une seule par variable. Comme méthodes de composition, on peut citer en particulier les compositions MAXIMUM (en général couplée avec l'inférence MINIMUM) et SOMME (en général couplée avec l'inférence PRODUIT)[16]. La première consiste à prendre le maximum de chacune des caractérisations floues impliquées dans la phase d'inférence pour une variable linguistique déterminée. La seconde consiste à faire la somme des caractérisations floues issues de l'inférence (figure 5.11). On notera que la seconde méthode ne garantit pas une normalisation de la caractérisation floue résultant de la composition.

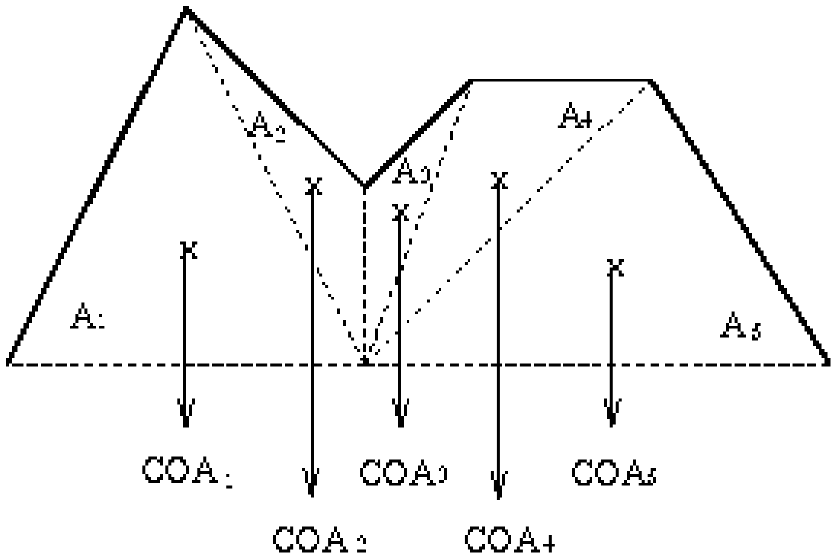


FIG. 5.12 – *Calcul du centre de gravité dans le cas de fonctions d'appartenance simples*

b) Passage symbolique $\rightarrow$ numérique

Une fois la phase de composition réalisée, il s'agit de déterminer la valeur numérique de chaque variable de sortie du système flou, à partir de la caractérisation floue résultant de cette phase. Il s'agit là de la phase de défuzzification proprement dite, permettant de générer les commandes qui peuvent être appliquées au système réel contrôlé. Il existe plusieurs méthodes de défuzzification (au moins une trentaine); les plus communément usitées sont :

– la méthode du centre de gravité[17]
  Dans le cas où la fonction d'appartenance associée à la caractérisation floue résultant de la phase de composition est de la forme $f(v)$, la valeur numérique délivrée est donnée par la formule :

$$valeur = \frac{\int_U v f(v) dv}{\int_U f(v) dv}$$

16. Dans la littérature, on regroupe souvent les phases d'inférence et de composition sous le vocable générique d'inférence. Les qualificatifs MIN-MAX ou PRODUIT-SOMME (ou vice-versa) caractérisent les opérations retenues pour ces deux phases.

17. Cette méthode est appelée COA en anglais, pour Center Of Area.

avec *valeur* définie sur l'univers de discours U de v où $\int_U vf(v)dv$ représente le moment de $f(v)$. Cette formule est issue de celle appliquée dans le cas discret (moyenne pondérée), à savoir :

$$valeur = \frac{\sum^{i} v_i \ \mu_A(v_i)}{\sum^{i} \mu_A(v_i)}$$

où les v_i représentent les niveaux de discrétisation de la variable de sortie v et μ_A la fonction caractéristique de la caractérisation floue de sortie A associée à v.
Bien que les fonctions d'appartenance des caractérisations floues puissent être quelconques, force est de constater que l'on recourt souvent à des fonctions trapézoïdéales, et surtout triangulaires. Cela s'explique certainement par la facilité des opérations pouvant être réalisées sur ces fonctions, notamment dans les phases d'inférence et de défuzzification. En décomposant la caractérisation floue résultant de la phase de composition en polygones convexes élémentaires (triangles par exemple), on peut déterminer la valeur numérique en prenant le barycentre de l'ensemble des polygones pondérés par leurs aires respectives (cf. figure 5.12). Dans ce cas, la valeur numérique sera obtenue de la manière suivante :

$$valeur = \frac{\sum^{i} COA_i \times A_i}{\sum^{i} A_i}$$

où les COA_i et A_i représentent respectivement les barycentres et les aires des triangles composant la caractérisation floue résultant de la phase de composition.

- la méthode du maximum
  Pour une variable de sortie donnée, si la fonction d'appartenance associée à la caractérisation floue de sortie ne possède qu'un maximum, la valeur numérique renvoyée est égale à ce maximum (figure 5.13).
- la méthode de la moyenne des maxima[18]
  Pour une variable de sortie donnée, la valeur numérique délivrée est calculée comme la moyenne des valeurs appartenant à la caractérisation floue de sortie et ayant comme degré d'appartenance le maximum de la fonction caractéristique de cette caractérisation floue (figure 5.14).

Une étude comparative de ces différentes approches peut être trouvée dans [Lee, 1990]. Il en ressort que la méthode de la moyenne des maxima assure de meilleures performances de transitions alors que la méthode du centre de gravité est plus régulière dans ses sorties. Les performances de la première méthode sont comparables à celles d'un système de relais à plusieurs niveaux. Les performances de la seconde méthode sont elles comparables à celles d'un contrôleur PI. Un contrôleur utilisant la méthode du centre de gravité garantit une erreur quadratique moyenne inférieure à celle d'un contrôleur utilisant la méthode de

18. Cette méthode est appelée MOM en anglais, pour Mean Of Maximum.

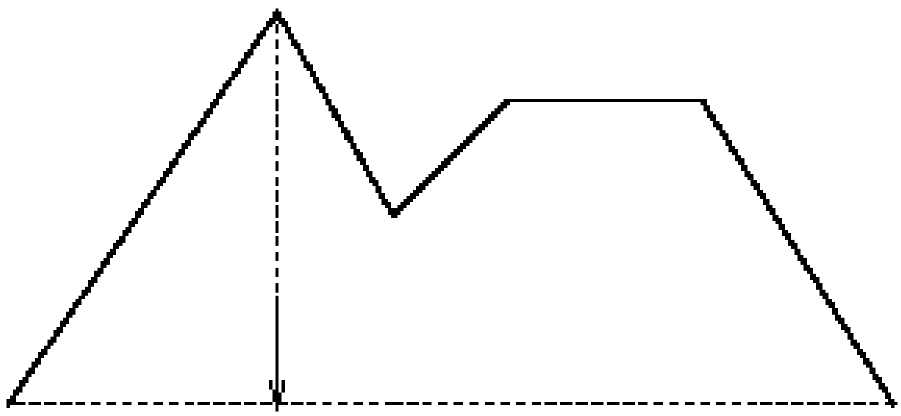


FIG. 5.13 – *Défuzzification par la méthode du maximum*

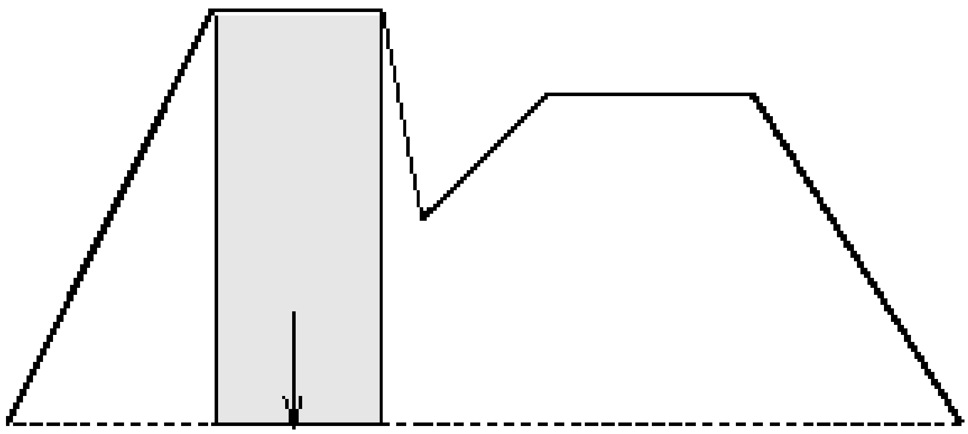


FIG. 5.14 – *Défuzzification par la méthode de la moyenne des maxima*

la moyenne des maxima. Cette dernière est cependant plus performante que la méthode du maximum.

5.4.3 Applications en robotique mobile

Un certain nombre d'applications utilisant la logique floue en robotique mobile existent dans la littérature. Nous donnons ci-après un aperçu de ces applications.

Le robot mobile de l'équipe PRIMA du LIFIA

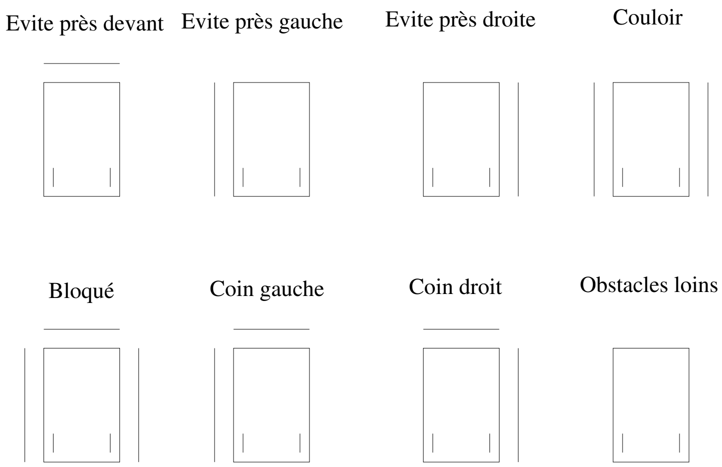


FIG. 5.15 – *Représentation des 8 situations prédéfinies*

Le robot mobile ROBUTER de l'équipe PRIMA du LIFIA est capable de se déplacer dans un environnement structuré intérieur [Reignier, 1994]. Le système de navigation admet en entrée les données relatives aux capteurs ultrasons ainsi que les données relatives au but. Dans le cas du but, les données retenues sont tout naturellement l'angle entre la direction du robot et la droite reliant le centre du robot au but ainsi que la distance séparant le robot du but. Pour l'angle, les données linguistiques sont : Derrière Droite, Droite, Devant Droite, Devant, Devant Gauche, Gauche, Derrière Gauche, et pour la distance : Près, Normal, Loin. Le robot mobile ROBUTER comprend vingt quatre capteurs ultrasons regroupés en quatre groupes : Devant, Derrière, à Gauche et à Droite. Lors de la phase d'acquisition, seule la mesure la plus faible de chaque groupe est retenue. Pour chaque zone, on utilise deux données linguistiques : Près, Loin. En sortie, le contrôleur flou délivre une vitesse linéaire et une vitesse angulaire. Les données linguistiques correspondates sont : Nulle, Lente, Normale, Rapide. Les règles du contrôleur flou sont déterminées de manière à reproduire les actes d'un opérateur humain téléopérant le système. Elles correspondent aux huit situations prédéfinies (figure 5.15) que peut rencontrer le robot. On ne présente ici que les huit situations prédéfinies correspondant à la donnée linguistique *près*. Les réactions associées à ces situations sont les suivantes :

- **Evite près devant** : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.

- **Evite près gauche** : on ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente vers la droite de manière à s'éloigner de l'obstacle.
- **Evite près droite** : de même, on ne fournit pas de consigne pour la vitesse de translation et on demande une rotation lente sur la gauche.
- **Couloir étroit** : pas de consigne pour la vitesse linéaire. On demande une vitesse angulaire nulle pour rester dans l'axe du couloir.
- **Blocage** : on arrête le mouvement de translation du robot et on demande une rotation lente vers la droite (de manière soit à réaliser un demi-tour, soit à apercevoir une issue par changement de point de vue résultant de la rotation).
- **Coin gauche** : on arrête le mouvement de translation du robot et on demande une rotation lente à droite.
- **Coin droit** : on arrête le mouvement de translation du robot et on demande une rotation lente à gauche.
- **Obstacles loins** : on place le robot face à son but. La vitesse est inversement proportionnelle à la distance au but.

Le véhicule du projet SHARP de l'INRIA Rhône-Alpes

La véhicule expérimental autonome du projet SHARP de l'INRIA Rhône-Alpes est un petit véhicule de type "voiture sans permis" de marque Ligier. Ce véhicule est capable de se déplacer dans un environnement dynamique, d'y éviter les obstacles, de suivre d'autres véhicules ou de se stationner. Le système de navigation [Garnier, 1995] doit remplir quatre conditions :

- Le suivi de trajectoires générées par le planificateur de chemins.
- La prise en compte de l'environnement statique.
- La prise en compte de l'environnement dynamique.
- L'activation du planificateur en cas d'échec du contrôleur.

Un contrôleur flou gère les conditions (2) et (3) du système de navigation. Pour cela, il admet en entrée les données relatives aux capteurs ultrasons (quatorze capteurs regroupés en huit zones). Pour chaque zone, les données linguistiques sont : Très proche, Proche, Assez proche, Loin. Les sorties sont une vitesse linéaire et une vitesse angulaire, dont les données linguistiques sont : Lente, Normale, Rapide. L'originalité principale de ce contrôleur flou est qu'il introduit une pondération des règles en fonction de leur importance. En effet, lors de la mise au point d'un contrôleur flou, bien que l'écriture des règles floues constituant la base soit relativement aisée, il est difficile de faire cohabiter plusieurs comportements dans la même base. En effet, suivant la situation courante, certains comportements doivent avoir une importance qui n'apparaît pas dans les règles. La pondération des règles permet de résoudre ce problème. La valeur de ces pondérations est déterminée par une méthode d'apprentissage supervisée, grâce à l'algorithme du simplexe [Garnier, 1995].

Autres travaux

Li [Li, 1994] et [Saffioti *et al.*, 93] définissent des comportements de haut niveau tels que l'évitement d'obstacles ou le suivi de murs en groupant des règles, et en associant un niveau d'activation à chaque comportement. Un algorithme définit un comportement global du robot en fonction du niveau d'activation de chaque comportement. Par exemple, [Li, 1994] utilise en entrée de son contrôleur flou les quinze capteurs ultrasons du robot mobile THMR-II regroupés en trois zones : Gauche, Devant, Droit, ainsi que l'angle entre la direction du robot et la droite reliant le centre du robot au but. A chaque groupe, on associe les données linguistiques suivantes : Proche, Normal, Loin. Pour l'angle, les données linguistiques sont : Droite, Devant, Gauche. A la sortie, le contrôleur flou rend une vitesse linéaire et une vitesse angulaire. Pour atteindre un but, trois comportements définis avec des règles flous sont utilisés : éviter les obstacles et ralentir dans les passages étroits, suivre un mur, se placer face au but.

Chaque comportement est défini au moyen de quatre règles, correspondant intuitivement au comportement d'un humain simulant ces trois comportements. La vitesse linéaire et la vitesse angulaire délivrée sont calculées par une pondération (selon le contexte) des trois comportements, grâce à l'algorithme d'inférence Min-Max et à la méthode du centre de gravité pour la défuzzification.

5.4.4 Conclusion

La logique floue, de par la quantité des travaux de recherche et des réalisations existantes, présente un certain nombre de points forts. En premier lieu, elle simplifie la conception d'un contrôleur dans la mesure où l'on décrit directement le fonctionnement de celui-ci sur le système contrôlé. Dans le cas de systèmes complexes, cela représente un avantage indéniable par rapport à la plupart des autres méthodes qui s'attachent à modéliser les systèmes. De plus, un contrôleur flou est relativement ouvert à travers l'utilisation de règles linguistiques écrites en langage naturel; il est possible de gérer de manière assez souple ces règles. Par contre, les contrôleurs flous sont quelquefois difficiles à concevoir, à cause :

- Du choix de données d'entrées et de sorties appropriées.
- De la définition de règles, et la possibilité de donner plus d'importance à certaines règles, ou de regrouper plusieurs règles.
- De la façon de défuzzifier les sorties du contrôleur flou.

De nombreux travaux ont été réalisés pour concevoir des contrôleurs flous [Garnier, 1995], [Saffioti *et al.*, 93] ou [Li, 1994]. Dans le chapitre suivant, nous proposons une nouvelle méthode pour concevoir des contrôleurs flous.

5.5 Conclusion

Nous avons présenté dans ce chapitre les trois techniques couramment utilisés pour réaliser un module de navigation réactive :

- Les techniques à base de potentiels dont le but est d'utiliser une fonction continue afin de coder les objectifs de navigation. Hormis les avantages et les inconvénients que nous avons déjà mentionnés, nous pensons que le gros avantage des champs de potentiel est qu'ils proposent une intégration facile de l'ensemble des forces résultantes des valeurs de chaque capteur et de la force exercée par le but à atteindre afin de déterminer la force globale à appliquer au robot. La réaction globale du robot est donc vue comme une résultante des réactions de chaque partie physique du robot. Cette idée parait très séduisante. Par contre, ils permettent moins facilement de privilégier les forces exercées par certains capteurs ou même de ne considérer que les forces exercées par certains capteurs que le contrôle flou.

- Les techniques à base de réseaux de neurones qui cherchent à apprendre la fonction de navigation. Nous ne revenons pas sur ces techniques car comme nous l'avons déjà dit, leur utilisation en navigation réactive n'est pas encore aussi répandu que les deux autres types d'approche, ce qui ne permet pas de concevoir un module de navigation aussi performant qu'avec les deux autres types d'approche.
- Le contrôle flou dont le but est de définir la manière de contrôler le robot, sous forme de règles. Cette idée est très séduisante. La définition de sous-ensembles flous associés aux valeurs permet la prise en compte du bruit. De plus, on peut regrouper les capteurs et disposer de plusieurs possibilités afin de prendre en compte les valeurs des capteurs du groupe, comme par exemple la valeur minimum ou la moyenne de l'ensemble des capteurs. L'écriture de règles portant sur une zone permet de définir des réactions différentes sur chaque zone. Par contre, les règles portant sur plusieurs zones et répondant à des situations plus complexes deviennent vite difficiles à écrire.

Comme nous l'avons vu chaque technique a des avantages et des inconvénients. Pour réaliser notre module de navigation, nous excluons d'utiliser des réseaux de neurones car ils sont à l'heure actuelle moins performants que les deux autres types d'approches. Le contrôle flou paraît une solution intéressante puisqu'elle permet une construction incrémentale et intuitive des règles régissant le comportement du module de navigation. Le plus gros inconvénient du contrôle flou est que la définition de règles portant sur plusieurs zones devient vite compliqué. Ce problème peut être évité grâce à l'utilisation de techniques comme dans [Garnier, 1995] ou de comportements comme dans [Saffioti *et al.*, 93]. Nous préférons utiliser une autre technique : partant du fait qu'en utilisant un contrôle flou, les règles portant sur un ensemble de capteurs sont faciles à écrire et permettent de donner une réaction différente à chaque ensemble de capteurs. Nous utilisons un contrôleur flou pour définir des réactions appropriées à chaque ensemble de capteurs. D'autre part, l'intégration des réactions de chaque capteur est relativement bien réalisée dans une approche à base de potentiels, nous utilisons cette méthode pour intégrer les réactions de chaque ensemble de capteurs.

6

Notre module de navigation

6.1 Introduction

Le rôle de notre module de navigation [Aycard *et al.*, 1997b] est de permettre au robot de se déplacer dans les couloirs du laboratoire et d'atteindre un but donné. Pour cela, le robot se place premièrement face à son but et deuxièmement se dirige vers son but. La première phase consiste simplement à faire une rotation de la base et de la tourelle de façon à ce que le robot se retrouve face à son but. Dans la deuxième phase, le module de navigation détermine à chaque instant l'action motrice qui le rapproche de son but et évite les obstacles qui se trouvent sur son chemin. Parallèlement à cela, à chaque fois qu'il découvre un obstacle, il poste dans la mémoire partagée du niveau physique la position de cet obstacle. De plus, il doit intégrer une autre contraire très forte : éviter que la tourelle fasse des rotations trop importantes, et ceci pour deux raisons :

- Le module de localisation interprète les données des capteurs afin de reconnaître les balises que le robot a "vues". Pour cela, il se sert de la variation de certains capteurs dans le temps. Pour observer cette variation dans le temps, il faut que le robot ne fasse pas de rotations trop importantes afin que chaque capteur observe toujours la même partie de l'environnement.
- Le bras manipulateur est situé à l'arrière de la tourelle. Pour éviter qu'il touche les obstacles, il doit toujours rester à l'arrière de la tourelle.

Nous avons vu dans le chapitre précédent, les trois possibilités les plus connus pour qu'un robot se déplace réactivement vers un but. Etant donné que les données sont bruitées et les actions du robot ne sont pas complétement fiables, le contrôle flou est une solution appropriée pour déplacer réactivement un robot vers un but. D'autre part, il permet une construction plus incrémentale des réactions du robot. On peut en effet, partir de quelques règles pour faire face à des situations simples, et au fur et à mesure les enrichir pour arriver à un module qui gère de plus en plus de situations. Ces règles permettent de définir des comportements de manière souple, et de les modifier très facilement. Nous avons vu que ces règles deviennent compliquées, et difficile à gérer lorsqu'elles sont nombreuses et portent sur plusieurs zones physiques du robot. Pour éviter cela, nous mettons en place un ensemble minimum de règles simples qui définissent la réaction associée à un ensemble de capteurs. Ces règles permettent de définir des comportements différents pour chaque ensemble de capteurs. Pour intégrer ces comportements, nous utilisons une technique similaire à celle des champs de potentiels, qui permet l'intégration facile des réactions de l'ensemble des parties physiques du robot.

Intuitivement, notre idée se justifie en considérant que la réaction globale du robot est vue comme une intégration de l'ensemble des réactions des parties physiques du robot. Le comportement global du robot émerge de l'intégration des comportements locaux du robot. Ce principe d'émergence est bien connu des systèmes multi-agents. Notre approche est à l'opposé d'une approche comme celle de [Saffioti *et al.*, 93] qui définit un ensemble de comportements de haut niveau tels que l'évitement d'obstacles ou le suivi de murs qui sont en compétition pour contrôler le robot, et où chaque comportement est un ensemble de règles s'appliquant sur tous les capteurs du robot. Nous allons en effet utiliser des ensembles de règles (ie, des comportements) qui vont définir la réaction des capteurs situés à droite (resp. gauche, resp. devant) du robot dans toutes les situations. Ceci nous permet de tester indépendamment les réactions des parties du robot, et de vérifier qu'elles répondent bien aux contraintes fixées, puis d'intégrer ces réactions afin de vérifier que notre robot a bien le comportement global que nous désirons. Nous pouvons définir notre approche comme un contrôleur flou à deux niveaux [Aycard *et al.*, 1997b]. Dans le premier niveau, nous avons pour chaque partie physique du robot un contrôleur flou classique qui à partir des valeurs des capteurs de chaque zone retourne une action motrice pour la zone, et au deuxième niveau un mécanisme d'intégration qui à partir de l'ensemble des actions motrices calcule l'action motrice globale du robot.

Dans la section suivante, nous expliquons comment nous plaçons le robot face à son but, avant qu'il commence à se déplacer. La section 6.3 est consacrée à la mise en place de notre contrôleur flou à deux niveaux pour qu'il évite les obstacles dans l'environnement. Les quatre premiers paragraphes de cette section détaillent le premier niveau du contrôleur : la définition des zones physiques ainsi que le choix de la valeur à fuzzifier, la fuzzification, la définition des règles pour chaque zone physique et la défuzzification. Dans le cinquième paragraphe, nous expliquons comment sont intégrés les différents comportements afin de définir le comportement global du robot, c'est-à-dire comment fonctionne le deuxième niveau de notre contrôleur flou. Pour terminer cette section, nous présentons quelques expériences pour évaluer notre approche telles que l'évitement d'un ou plusieurs obstacles, ou le passage d'une porte. La dernière section de ce chapitre est consacrée à l'ajout d'un comportement afin de prendre en compte un but donné à atteindre tout au long du déplacement du robot.

6.2 Le placement face au but

Le placement face au but revient à faire tourner le robot pour qu'il se retrouve face à son but. Cette opération pose un seul problème. En effet, à cause de l'encombrement du bras, la tourelle ne peut pas toujours immédiatement se placer face au but. Cette opération se déroule en deux étapes :

- Premièrement, la base se place face au but, puis le robot commence à se déplacer tout doucement vers le but.
- Deuxièmement, à chaque instant, nous vérifions si la tourelle peut se placer face au

but sans que le bras touche un obstacle ou un mur pendant la rotation. Pour cela, nous calculons la rotation que doit faire la tourelle pour se placer face au but, puis nous examinons à l'aide des capteurs si tout au long de la rotation, la distance entre le robot et le ou les objet(s) le(s) plus proche(s) est supérieur à la taille du bras. A chaque instant, nous essayons de placer la tourelle face au but en effectuant une rotation dans le sens trigonométrique ou dans le sens des aiguilles d'une montre. Si les deux sens sont possibles alors la rotation la moins importante est réalisée. Si un seul sens de rotation est possible alors la tourelle se place face au but en effectuant la rotation dans ce sens. Si aucun sens n'est possible alors aucune rotation n'est effectuée. La base se déplaçant tout doucement vers le but, la tourelle pourra tôt ou tard dans quasiment tous les cas se placer face au but.

Une fois le placement face au but terminé, le robot commence à se déplacer réactivement vers son but.

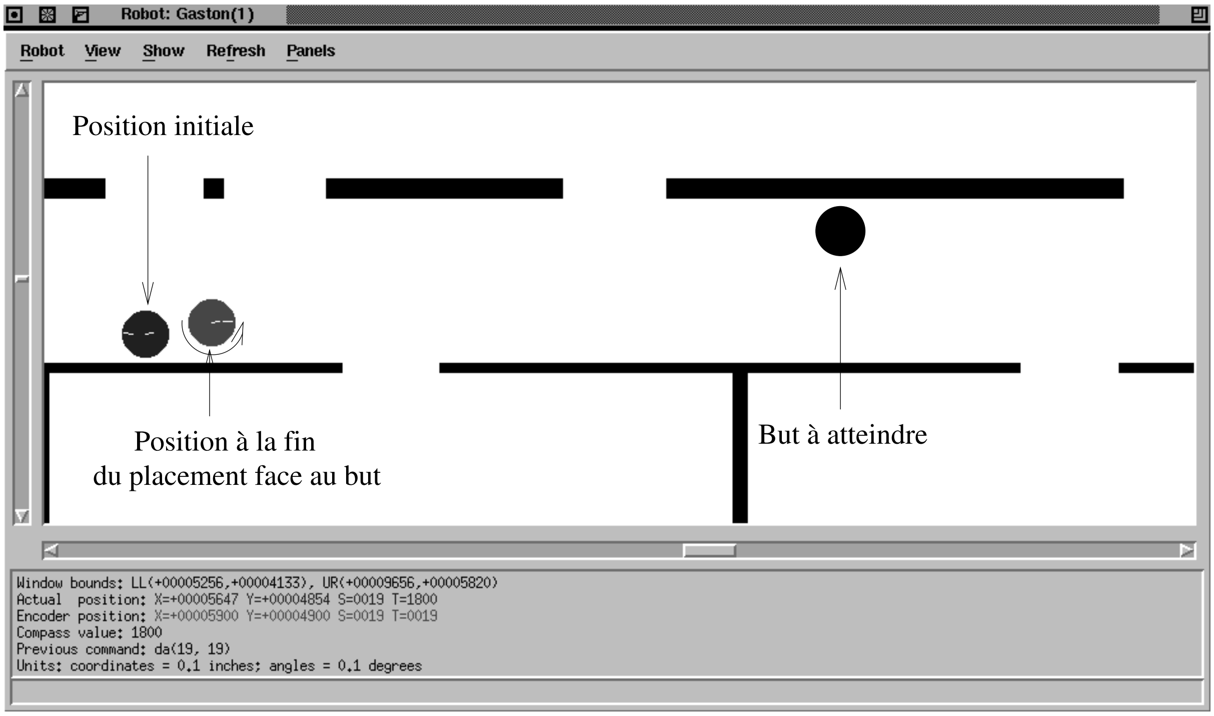


FIG. 6.1 – *Placement face au but*

La figure 6.1 donne un exemple de placement face au but. La tourelle est représentée par le trait à l'extérieur du robot, et la base par le trait à l'intérieur. La position initiale que nous présentons se situe à la fin de la première étape. Nous voyons clairement qu'à la position initiale du robot la tourelle et la base ne sont pas alignées. La tourelle n'a pas encore tourné et se trouve toujours dans l'orientation initiale du robot. La base par contre est déjà face au but. Le robot peut donc commencer à avancer tout doucement vers le but. Une fois que la tourelle peut tourner, car il n'y a plus de collision possible en effectuant la rotation dans le sens trigonométrique, la tourelle vient se réaligner à la base. Le robot peut commencer à se déplacer vers son but.

Nous avons dit dans la présentation de notre architecture de contrôle (chapitre 4) qu'à chaque fois que le module de localisation poste la position du robot, le contrôleur d'exécution demande à nouveau au navigateur d'atteindre cette position. Ceci afin de prendre en compte la réactivité tactique et qu'à chaque instant le contrôleur fasse exécuter au navigateur le déplacement qui rapproche le robot de son but. Si le robot est déjà face au but, il ne se place pas à nouveau face au but et continue à se déplacer. Dans le cas contraire le robot s'arrête et se place face au but, et recommence à se déplacer vers le but. Pour détecter si le robot doit se replacer ou non face à son but, nous avons mis au point une technique très simple. Nous savons que le robot se déplace dans un environnement intérieur structuré composé de couloirs qui sont soit perpendiculaires soit parallèles (figure 4.10). Au moment de placer le robot face à son but, connaissant sa position initiale et la position du but, on détermine s'il va emprunter un couloir vertical et dans quel sens il va l'emprunter (ie, en allant de bas en haut ou de haut en bas) ou s'il va emprunter un couloir horizontal et dans quel sens il va l'emprunter (ie, en allant de gauche à droite ou de droite à gauche). Donc au moment où le contrôleur d'exécution demande au navigateur d'atteindre un but alors que le robot se déplace déjà, il va vérifier si à partir de sa position actuelle (postée par le localisateur dans la mémoire partagée), il doit toujours emprunter le même couloir et surtout s'il est toujours dans le bon sens pour atteindre ce but.

6.3 Notre contrôleur flou à deux niveaux

6.3.1 Introduction

Dans un premier temps, nous avons conçu notre contrôleur flou [Aycard *et al.*, 1997b] pour faire naviguer notre robot mobile de manière réactive dans les couloirs du laboratoire tout en évitant les obstacles. Nous détaillons dans cette section son fonctionnement, et donnons les résultats d'expériences que nous avons réalisées. Pour le définir (figure 6.2), nous utilisons les seize capteurs infrarouges/ultrasons comme entrée, et nous générons une action motrice pour la base et la tourelle en sortie. Cette génération est faite en deux niveaux :

– Dans le premier niveau, nous définissons l'ensemble des comportements locaux. Cette définition fait l'objet des quatre premiers paragraphes de ce niveau. Dans le premier paragraphe, les capteurs adjacents sont stratégiquement regroupés pour former une zone physique. Puis dans le second paragraphe pour chaque zone, nous choisissons un critère (moyenne, minimum) pour définir la/les donnée/données capteurs utilisées pour la fuzzification. La conception des règles pour chaque zone physique afin de déterminer une action motrice locale pour la base et la tourelle correspondant au comportement désiré pour la zone physique en question fait l'objet du troisième paragraphe. Comme les règles sont écrites pour chaque zone, et ne s'appliquent que sur les capteurs de cette zone, nous n'avons pas de conjonction dans les prémisses de chaque règle, ce qui rend les règles plus faciles à écrire. Nous expliquons comment les actions moteurs correspondantes sont calculées par défuzzification en utilisant la méthode classique du centre de gravité. En fait, chaque comportement physique

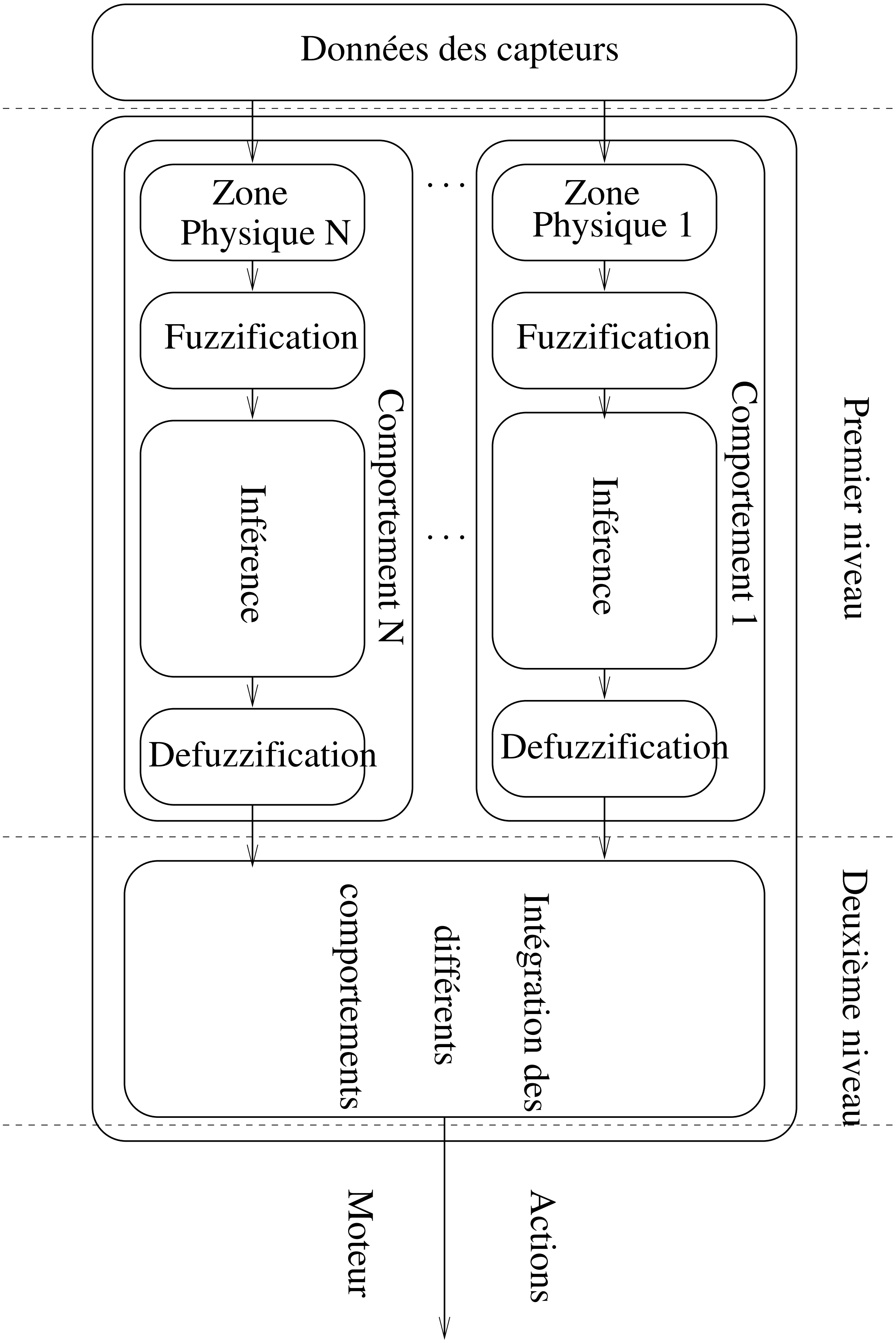


FIG. 6.2 – *Notre contrôleur flou à deux niveaux*

peut être vu comme un simple contrôleur flou local. A la fin du premier niveau, nous possédons une action motrice (pour la base et pour la tourelle) correspondant à chaque zone physique.

- Dans le deuxième niveau, nous définissons un mécanisme pour intégrer tous les comportements locaux et calculer un comportement global du robot. Nous utilisons pour cela, deux mécanismes pour définir le comportement global du robot :
  - Un mécanisme de fusion comme [Arkin, 1987], calculant une moyenne de toutes les actions motrices locales.
  - Un mécanisme d'inhibition comme [Brooks, 1986], où chaque comportement local a un niveau de priorité et le comportement activé (un comportement est activé s'il génère une action motrice non nulle) ayant le niveau de priorité le plus elévé contrôle le robot mobile.

Une combinaison de ces deux mécanismes est aussi possible. La mise en place de ce mécanisme et de l'intégration des comportements locaux du premier niveau fait l'objet du cinquième paragraphe. Dans le dernier paragraphe de cette partie, nous décrivons un certain nombre d'expérimentations que nous avons réalisées avec notre contrôleur flou. Les premières expérimentations sont des évitement d'obstacles dans un couloir qui sont les situations types que va devoir affronter notre robot. Puis nous donnons les résultats d'autres expérimentations qui sortent un peu du cadre pour lequel notre contrôleur flou est conçu. Ces expérimentations ont pour but de montrer les points forts et les limites de notre approche.

6.3.2 Définition des zones physiques

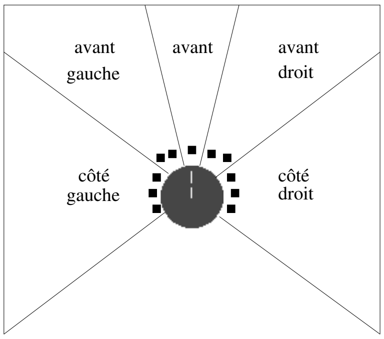


Fig. 6.3 – *Les 5 zones physiques*

Sur les seize capteurs infrarouges/ultrasons, nous utilisons onze capteurs regroupés en cinq zones physiques (figure 6.3) appelées :

- La zone Avant composée d'un capteur.
- La zone Avant Droite composée de deux capteurs.
- La zone Avant Gauche composée de deux capteurs.
- La zone Droite composée de trois capteurs.
- La zone Gauche composée de trois capteurs.

Les cinq capteurs arrière ne sont pas utilisés, puisque nous ne traitons que les obstacles situés sur les côtés et à l'avant du robot mobile, étant donné qu'à l'heure actuelle nous ne prévoyons pas que le robot mobile aura la possibilité d'effectuer des marches arrières.

6.3.3 La fuzzification

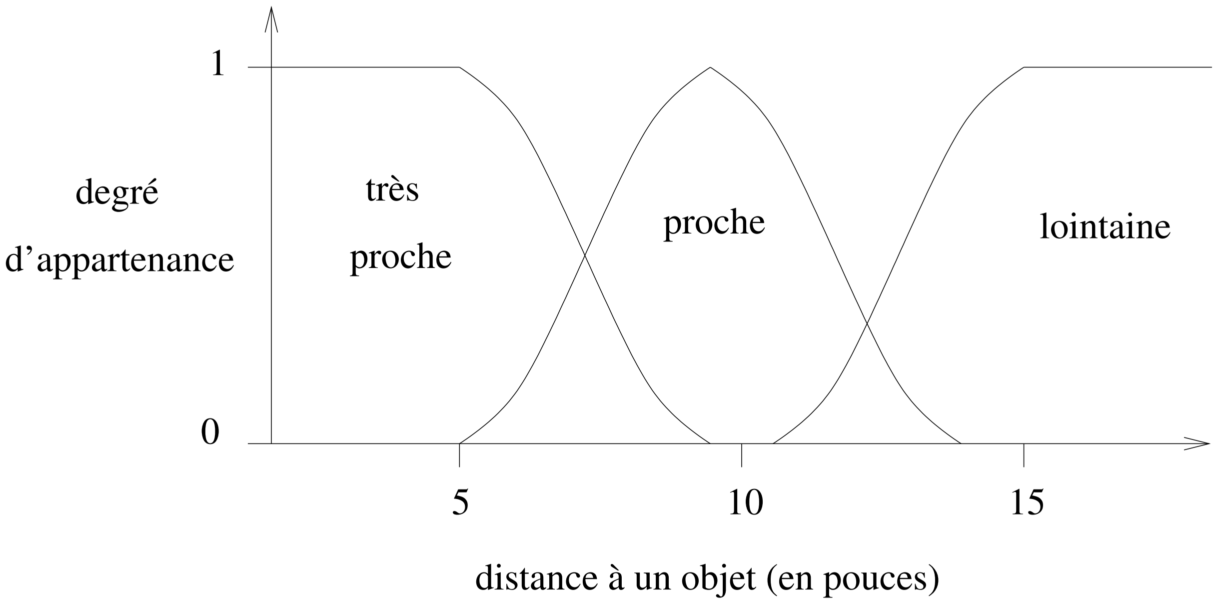


FIG. 6.4 – *Données linguistiques associées aux zones physiques*

Pour chaque zone physique, nous choisissons le capteur dont la valeur est minimum comme entrée pour chaque comportement. Cette valeur est associée à une donnée linguistique (figure 6.4) : Très proche, Proche, Loin.

6.3.4 Définition des comportements physiques

A partir de ces cinq zones physiques, nous définissons cinq comportements physiques : côté gauche, avant gauche, avant, avant droit, côté droit. Chaque comportement servant à éviter les obstacles situés dans la zone physique qui lui correspond. On associe aux actions motrices de rotation des données linguistiques : Nulle, Lente, Rapide (figure 6.5). Les règles définissant chaque comportement sont intuitives et correspondent aux réactions d'un opérateur humain placé dans une telle situation.

Pour le côté gauche, nous désirons le comportement suivant (figure 6.6) :
- Les obstacles très proches causent une rotation rapide à droite pour le robot mobile.
- Les obstacles proches causent une rotation lente à droite pour le robot mobile.
- Les obstacles lointains ne causent pas de rotation pour le robot mobile.

Donc, les règles sont :

```
Si côté gauche est très proche alors tourner rapidement à droite
```

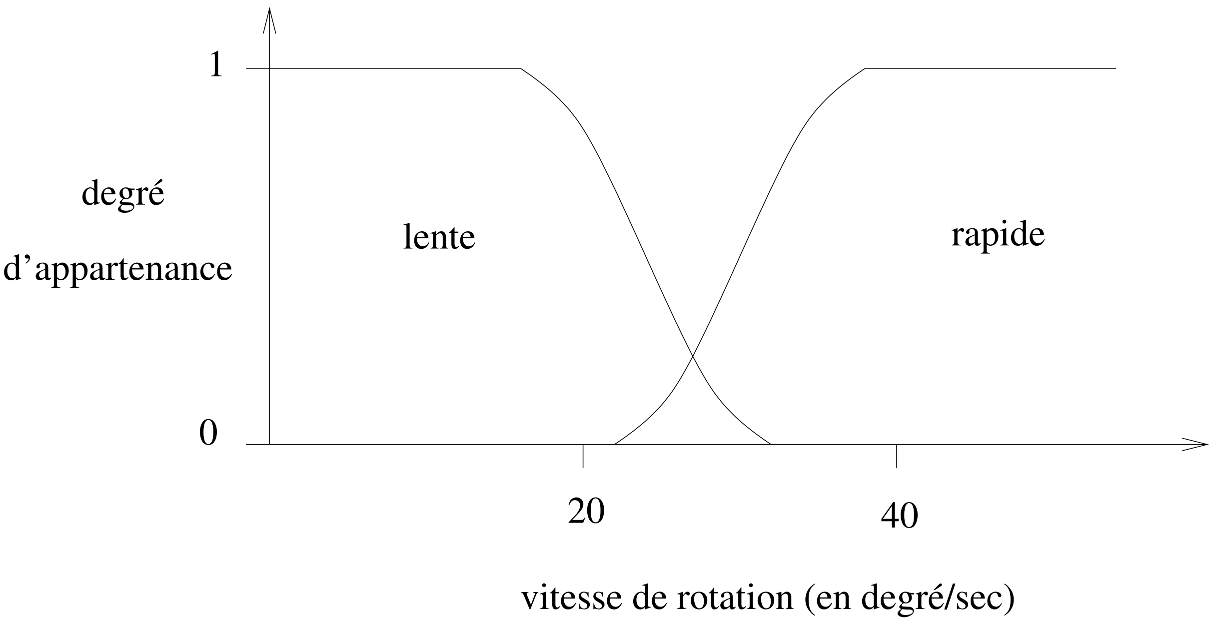


FIG. 6.5 – *Données linguistiques associées aux actions motrices de rotation*

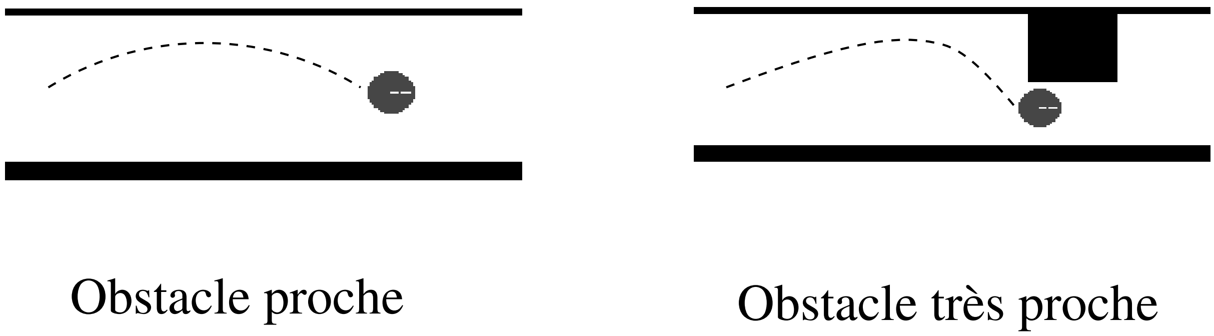


FIG. 6.6 – *Comportement désiré pour le côté gauche*

`Si côté gauche est proche alors tourner lentement à droite`
`Si côté gauche est loin alors ne pas tourner`

Pour l'Avant gauche, nous désirons le comportement suivant :

- Les obstacles proches ou très proches causent une rotation rapide à droite pour le robot mobile.
- Les obstacles lointains ne causent pas de rotation pour le robot mobile.

Donc, les règles sont :

```
Si Avant gauche est très proche alors tourner rapidement à droite
Si Avant gauche est proche alors tourner rapidement à droite
Si Avant gauche est loin alors ne pas tourner
```

Pour l'Avant droit, nous désirons le comportement suivant :

- Les obstacles proches ou très proches causent une rotation rapide à gauche pour le robot mobile.
- Les obstacles lointains ne causent pas de rotation pour le robot mobile.

Donc, les règles sont :

```
Si Avant droit est très proche alors tourner rapidement à gauche
Si Avant droit est proche alors tourner rapidement à gauche
Si Avant droit est loin alors ne pas tourner
```

Pour le côté droit, nous désirons le comportement suivant :

- Les obstacles très proches causent une rotation rapide à gauche pour le robot mobile.
- Les obstacles proches causent une rotation lente à gauche pour le robot mobile.
- Les obstacles lointains ne causent pas de rotation pour le robot mobile.

Donc, les règles sont :

```
Si côté droit est très proche alors tourner rapidement à gauche
Si côté droit est proche alors tourner lentement à gauche
Si côté droit est loin alors ne pas tourner
```

Pour l'avant, étant donné que les rotations ne sont généralement pas suffisantes pour éviter les obstacles frontaux, et comme notre robot mobile peut tourner sa base indépendamment de sa tourelle, notre idée est qu'en présence d'un obstacle frontal, le robot va naviguer en "crabe". Ce qui signifie que la tourelle restera face à l'obstacle mais que la base se décalera pour permettre au robot de longer l'obstacle jusqu'à ce qu'il trouve un passage pour éviter l'obstacle. Pour ce comportement, nous utilisons deux données linguistiques supplémentaires décrites dans la figure 6.7. Un décalage n'est pas une vitesse de rotation, mais un angle de rotation.

Pour l'avant, nous désirons le comportement suivant (figure 6.8):

- Les obstacles très proches causent un décalage important de la base par rapport à la tourelle.
- Les obstacles proches causent un décalage moyen de la base par rapport à la tourelle.
- Les obstacles lointains ne causent pas de décalage de la base par rapport à la tourelle.

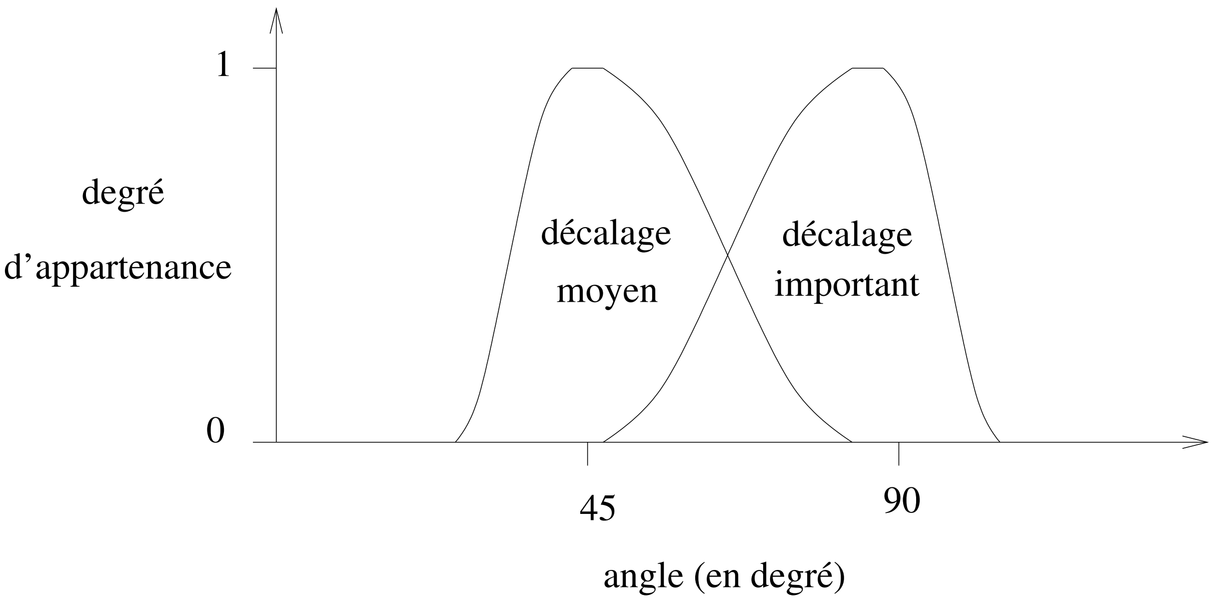


FIG. 6.7 – *Données linguistiques utilisées pour le décalage de la base*

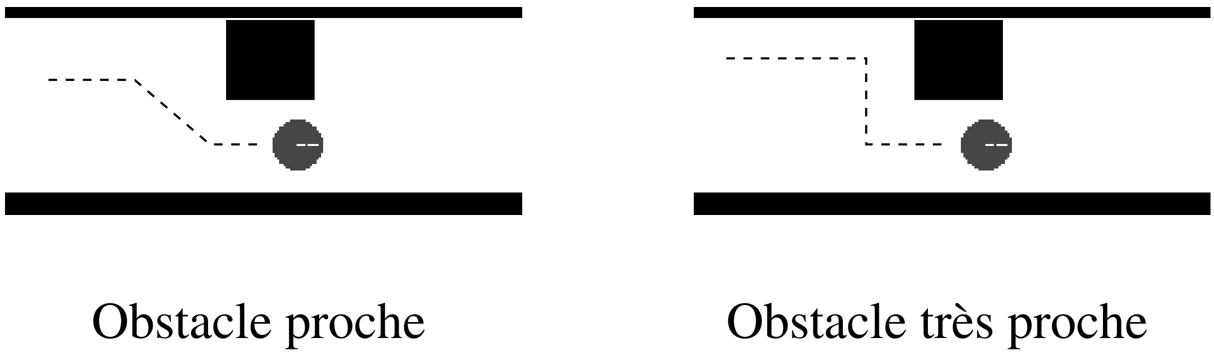


FIG. 6.8 – *Comportement désiré pour l'avant*

Donc, les règles sont :
`Si Avant est très proche alors procéder à un décalage important de la base par rapport à la tourelle.`
`Si Avant est proche alors procéder à un décalage moyen de la base par rapport à la tourelle.`
`Si Avant est loin alors ne pas procéder à un décalage de la base par rapport à la tourelle.`
Dans le premier niveau, nous ne définissons pas la direction de décalage (gauche ou droite). Ceci sera fait dans la phase d'intégration des comportements locaux. Il faut aussi noter que le décalage de la base est réalisée par rapport à la tourelle, ce qui veut dire que même si une règle est active pendant un certain temps et demande le décalage de la base par rapport à la tourelle, ce décalage n'est effectué que s'il est différent du décalage actuel entre la base et la tourelle. Ainsi le décalage maximum qu'il peut y avoir entre la base et la tourelle est de quatre vingt dix degrés au maximum. Ceci évite que le robot fasse marche arrière, ou que la tourelle tourne beaucoup et donc que les capteurs n'observent plus la même partie de l'environnement.

D'autre part à chaque fois que le robot procède à un décalage entre la base et la tourelle alors que juste auparavant elles étaient alignées, il poste dans la mémoire partagée du niveau physique la position de l'obstacle, qu'il calcule à l'aide du capteur avant.

6.3.5 La défuzzification

Pour délivrer une action moteur locale en sortie de chaque comportement physique, nous utilisons la méthode du centre de gravité pour défuzzifier.

6.3.6 Intégration des différents comportements physiques

Tous les comportements physiques fournissent une rotation locale pour la base et la tourelle, mis à part le comportement avant qui fournit un angle de rotation de la base. Pour définir un comportement global du robot mobile, nous devons intégrer toutes ces actions motrices de manière à trouver la meilleure action moteur pour le robot mobile.

Tous les comportements sont utilisés à même fin qui est d'éviter des obstacles, tous les comportements locaux ont la même priorité et sont fusionnés pour donner le comportement global du robot. L'action moteur globale sera naturellement une fusion de toutes les actions moteurs locales. Comme nous utilisons le repère local du robot pour définir les angles et les vitesses de rotation, les rotations à gauche provoquées par les comportements côté droit et avant droit sont positives et les rotations à droite provoquées par les comportements côté gauche et avant gauche sont négatives. Pour calculer la vitesse globale de rotation du robot, nous additionnons simplement les vitesses de rotation générées par le côté gauche, l'Avant gauche, l'Avant droit, et le côté droit. Cette vitesse globale de rotation est la même pour la base et la tourelle.

Si un décalage est demandé, alors le sens du décalage est défini par le signe de la vitesse de rotation. La base se décalera à gauche quand la vitesse globale de rotation sera positive et à droite quand la vitesse globale de rotation sera négative. Ceci s'explique intuitivement par le fait que le décalage doit être fait du côté où les obstacles sont le plus loin.

Pour calculer la vitesse de translation du robot, nous soustrayons simplement la valeur absolue de la vitesse de rotation à la vitesse de translation maximum. Ce qui signifie que plus la vitesse de rotation est importante plus la vitesse de translation est faible, et vice versa.

6.3.7 Expérimentations et résultats

Pour évaluer un module de navigation, il existe deux possibilités :

- On peut définir un certain nombre de critères tels que la sécurité de la trajectoire empruntée ou la distance par rapport aux obstacles, et évaluer *à postériori* la qualité de la trajectoire qu'a parcouru le robot pendant son déplacement [Fabiani, 1997].
- On peut définir un certain nombre de situations types et vérifier que le robot est capable d'y faire face.

Etant donné que le but de notre module de navigation réactive n'est pas de suivre la meilleure trajectoire mais simplement d'éviter les obstacles et de veiller à ce que la tourelle ne fasse pas de rotations trop importantes, nous l'avons testé dans diverses situations. Nous présentons tout d'abord des situations d'évitement d'obstacles dans des couloirs, puis diverses situations telles que le passage d'une porte, le passage d'un rétrécissement de couloir ou l'évitement d'obstacles dans une pièce. Nous commentons dans ce paragraphe les divers résultats obtenus. Dans chacune des figures présentées, le robot se déplace de la gauche vers la droite. Son but est de traverser le couloir.

Evitement d'obstacles dans un couloir

Nous avons dans un premier temps testé notre approche pour éviter des obstacles dans un couloir. Le robot est toujours placé au milieu du couloir, et dans une direction parallèle aux deux murs. Son but est de traverser le couloir.

Lorsque le robot évite un seul obstacle dans le couloir (figure 6.9), les zones avant (gauche ou droite, selon les cas) détectent l'obstacle d'assez loin, ce qui permet au robot de commencer à se décaler pour éviter l'obstacle. Les actions motrices locales des zones avant (gauche ou droite) permettent généralement au robot de se décaler suffisamment pour éviter l'obstacle sans que l'évitement de la zone frontale provoque un décalage des roues. Lorsque le robot se trouve entre l'obstacle et le mur, la résultante des actions motrices des zones de côté est quasiment nulle, ce qui permet au robot de longer l'obstacle sans toucher ni le mur, ni l'obstacle. Une fois l'obstacle évité, le robot se décale vers le milieu du couloir, puisque les actions motrices générées sont plus fortes d'un côté que de l'autre. Il va naviguer d'un côté à l'autre du couloir, pour progressivement se retrouver au milieu du couloir.

Lorsque le robot évite deux obstacles dans un couloir (figure 6.10), ses réactions par rapport au premier obstacle sont quasiment les mêmes que dans l'expérience précédente. Une fois le premier obstacle évité, le robot cherche à se décaler vers le milieu du couloir. Dès que le deuxième obstacle est détecté par les zones avant (gauche ou droite, selon les cas), le robot longe le mur pour éviter le deuxième obstacle. Une fois le deuxième obstacle évité, le robot se décale vers le milieu du couloir, puisque les actions motrices générées

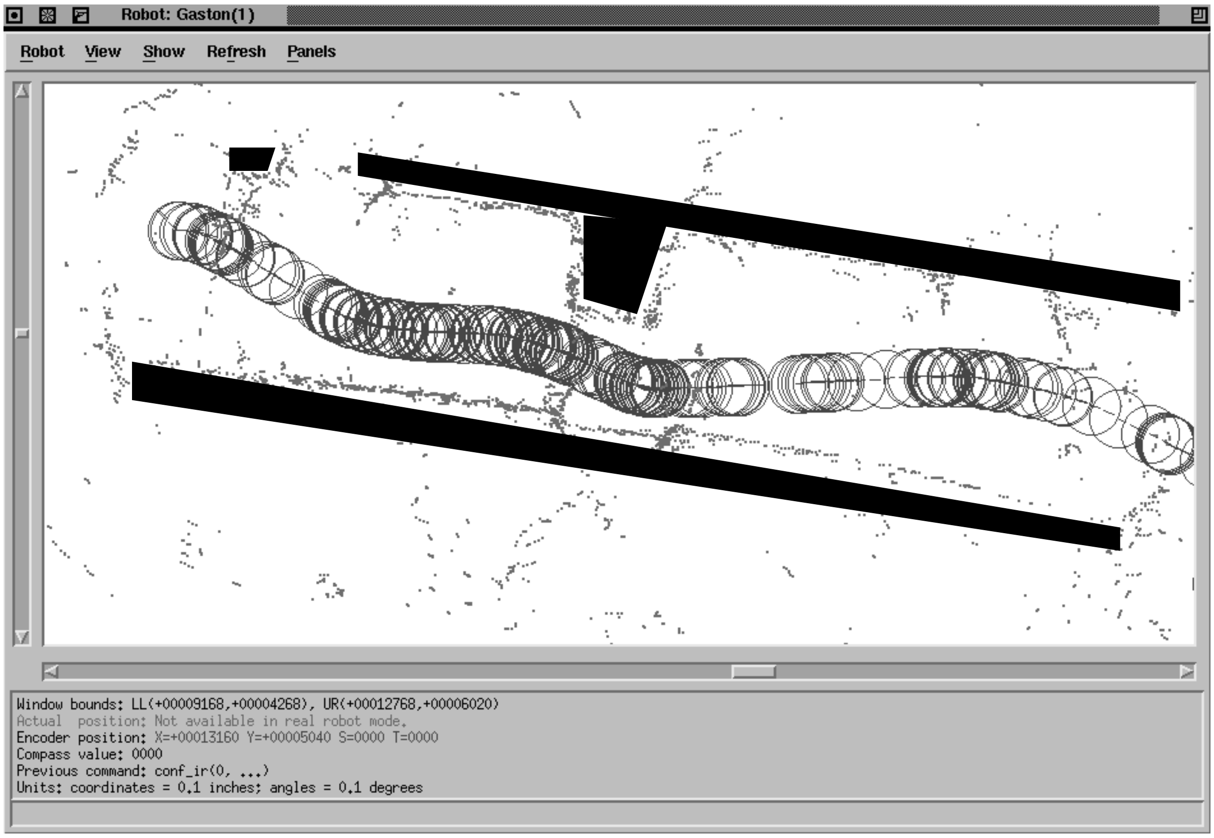


FIG. 6.9 – *Evitement d'un obstacle*

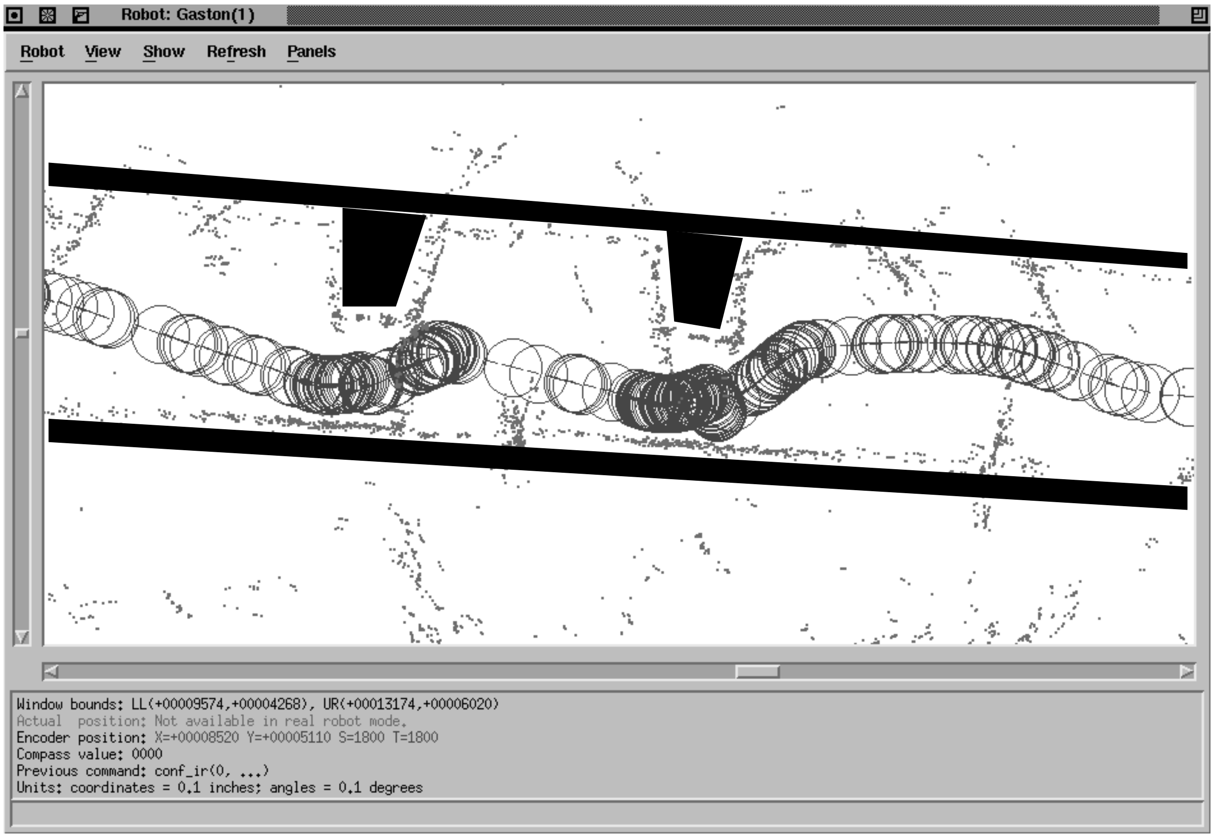


FIG. 6.10 – *Evitement de deux obstacles*

sont plus fortes d'un côté que de l'autre. Il va naviguer d'un côté à l'autre du couloir, pour progressivement se retrouver au milieu du couloir.

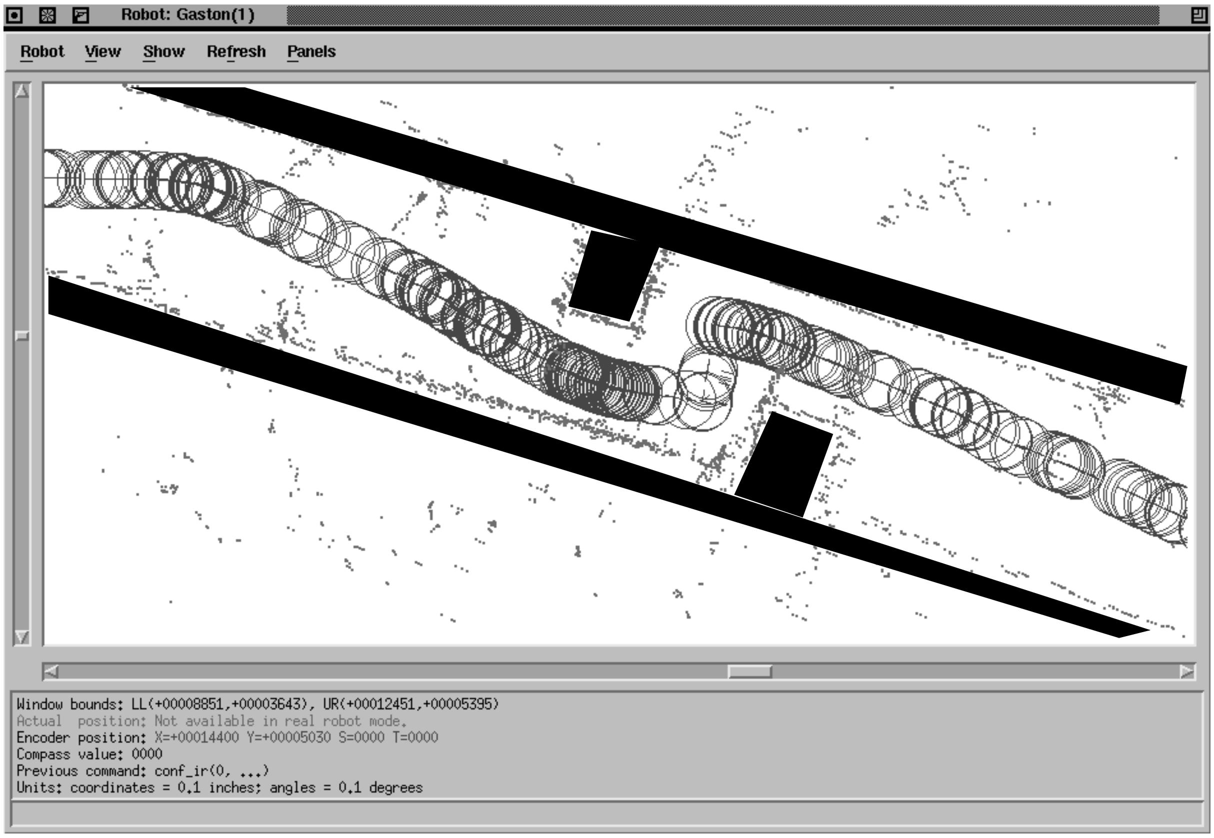


FIG. 6.11 – *Passage d'une chicane*

Lorsque le robot passe une chicane dans un couloir (figure 6.11), ses réactions par rapport au premier obstacle sont quasiment les mêmes que dans l'expérience précédente. Une fois le premier obstacle évité, le robot cherche à se décaler vers le milieu du couloir. Le deuxième obstacle très proche du premier est détecté par la zone frontale, ce qui provoque un décalage de la base par rapport à la tourelle. Cette suite d'actions motrices permet au robot de naviguer en crabe pour éviter le deuxième obstacle. Lorsqu'il a trouvé un passage pour éviter le deuxième obstacle, il réaligne sa base avec sa tourelle, puis se décale vers le milieu du couloir, puisque les actions motrices générées sont plus fortes d'un côté que de l'autre. Il va naviguer d'un côté à l'autre du couloir, pour progressivement se retrouver au milieu du couloir.

Traversée d'un couloir

Nous avons fait évoluer dans un couloir d'une longueur approximative de trente mètres. Au départ, le robot est placé au début du couloir, à égale distance des deux murs et dans une direction parallèle aux deux murs. Le robot mobile doit donc traverser le couloir jusqu'à l'autre extrémité et revenir au départ. Cette expérience a été réalisée environ cinquante fois, sans intervention humaine. Le robot mobile doit éviter les obstacles statiques,

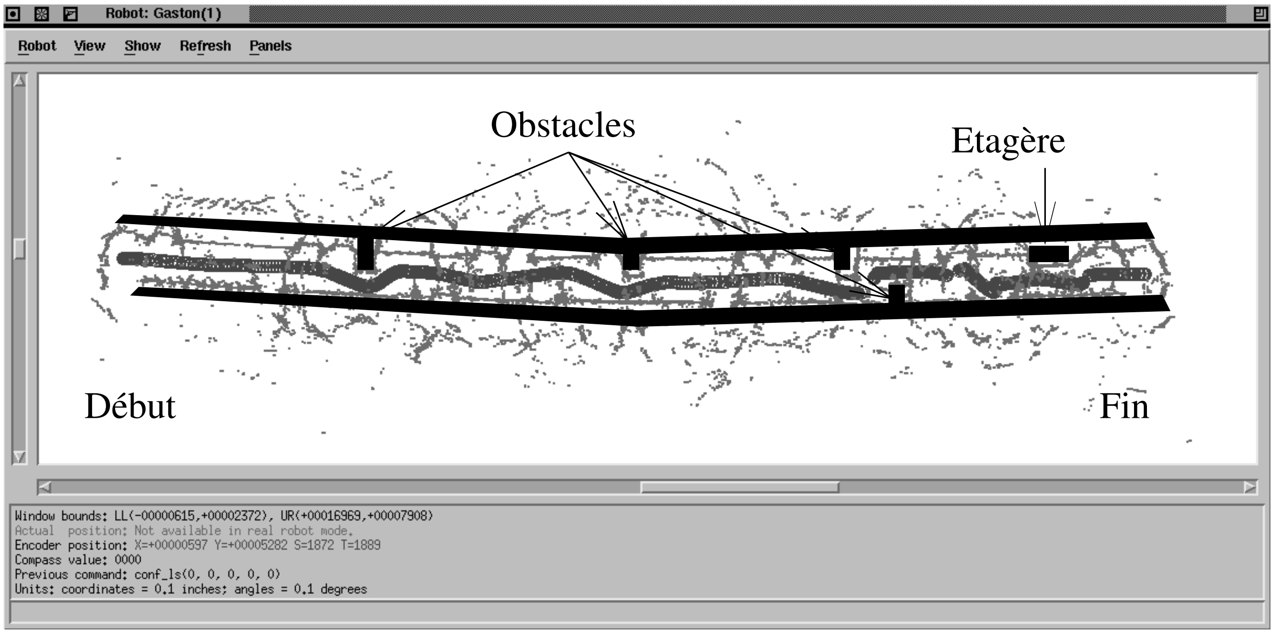


FIG. 6.12 – *Traversée d'un couloir*

et les personnes dans le couloir. Les obstacles statiques ont été volontairement déplacés au fur et à mesure des expériences. Un exemple de parcours du robot est présenté en figure 6.12.

Le robot mobile ne navigue pas vraiment droit, il "rebondit" d'un mur à l'autre. Mais, il ne touche jamais les murs et contourne les obstacles statiques. De plus, il évite les personnes dans le couloir.

Expérimentations diverses

Nous présentons ici un ensemble d'expérimentations diverses qui à part le rétrécissement de couloir dépassent le cadre initial des situations pour lesquelles notre contrôleur flou à deux niveaux a été conçu.

Lorsque le robot doit passer une porte pour entrer dans une pièce (figure 6.13), il se décale pour se mettre dans une position où il ne touchera pas les montants de la porte. Ce décalage est réalisé grâce aux zones gauche et droite. En effet, si le robot est plus proche d'un côté que de l'autre de la porte, l'action motrice de la zone située de ce côté l'en éloignera. Lorsque le robot est en train de franchir la porte, les actions motrices des zones de chaque côté lui permettront de ne pas toucher les montants de la porte. Une fois la porte franchie, le robot réagira en fonction de l'encombrement de la pièce dans laquelle il se trouve.

Lorsque le robot doit passer dans un endroit où le couloir se rétrécit (figure 6.14), les actions motrices des zones avant gauche et avant droite lui permettent de se placer dans une position où il pourra facilement pénétrer dans la zone rétrécie. Lorsque le robot se trouve dans la zone rétrécie, les actions motrices des zones de côté lui permettent de ne pas toucher les bords de la zone rétrécie. Dans la zone rétrécie, la trajectoire du robot oscille un peu. La somme des actions générées par les zones de côtés change souvent de

FIG. 6.13 – *Passage d'une porte*

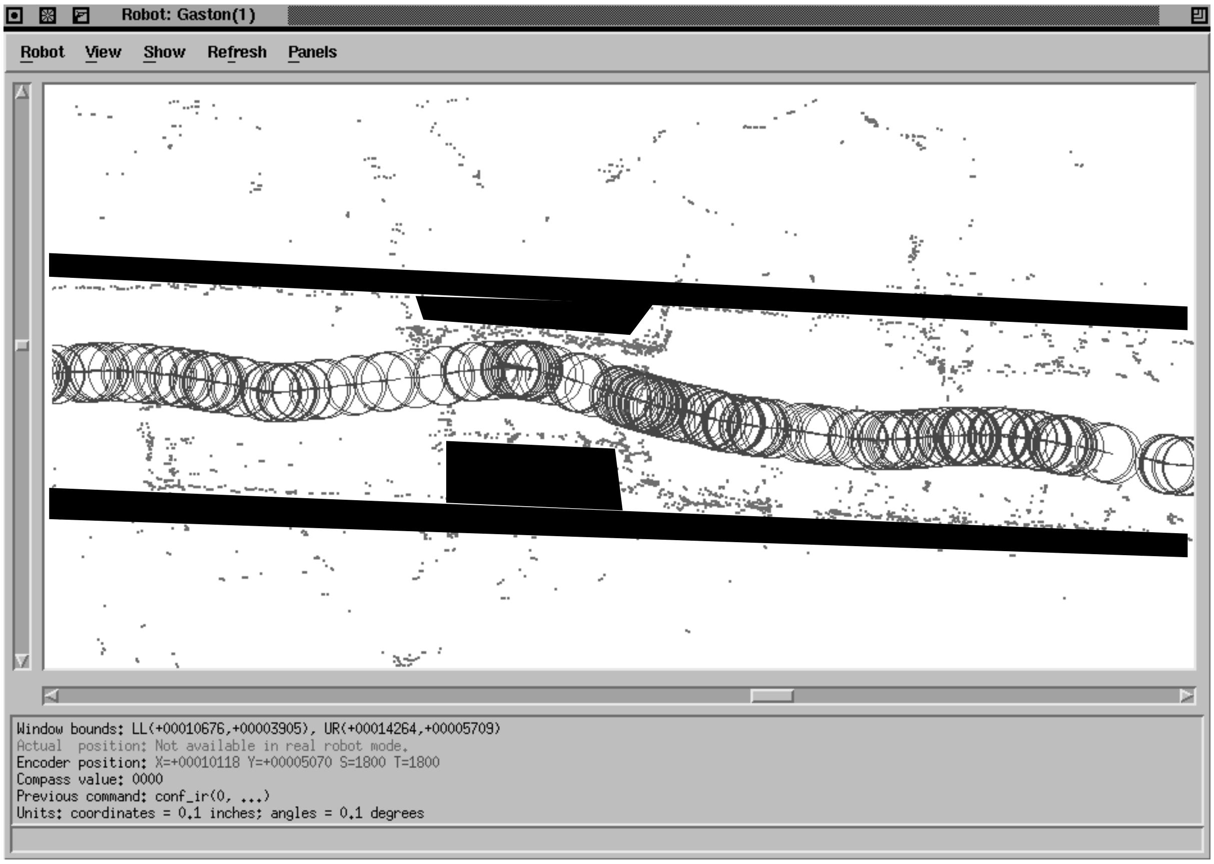


FIG. 6.14 – *Rétrécissement du couloir*

signe, à cause du peu de place dont dispose le robot. Une fois la zone rétrécie franchie, le robot va naviguer d'un côté à l'autre du couloir, pour progressivement se retrouver au milieu du couloir.

Les deux expériences présentées montrent que notre approche permet au robot de franchir avec succès des situations correspondant à des minimums locaux pour une approche à base de potentiels.

Evitement d'obstacles dans une pièce

Dans un deuxième temps, bien que ce ne soit pas la fonction initiale de notre module de navigation, nous avons testé notre approche dans une pièce. Il s'agit ici pour le robot de traverser la pièce en évitant les obstacles qui s'y trouvent. Le robot est placé dans une direction qui lui permet de traverser la pièce.

Lorsque le robot doit éviter un obstacle dans une pièce (figure 6.15), il commence tout d'abord à naviguer droit devant lui avec une vitesse de translation maximum. En effet, la zone frontale ne détecte aucun obstacle, et aucune action motrice n'est générée par les zones de côtés afin d'amener le robot au milieu de la pièce, car la largueur de la pièce est beaucoup plus importante que celle d'un couloir. Au fur et à mesure que le robot se rapproche de l'obstacle, les zones avant (gauche ou droite selon les cas) commencent à percevoir l'obstacle. Le robot commence donc à ralentir et à tourner pour éviter l'obstacle. Le côté droit étant plus libre que le côté gauche, c'est de ce côté que le robot choisit de passer. Une fois que le robot se trouve entre l'obstacle et le mur, les réactions des zones de côtés permettent au robot de traverser cette zone étroite sans toucher ni le mur, ni l'obstacle. Une fois l'obstacle évité, le robot s'éloigne progressivement du mur.

Nous avons répété l'expérience précédente en ajoutant un deuxième obstacle derrière le premier (figure 6.16). Le robot se trouve toujours à peu près à la même position de départ, avec la même orientation. On remarque que le robot a des réactions assez différentes par rapport au premier obstacle. En effet, il choisit ici de l'éviter en passant par la gauche. De plus, il a dû longer l'obstacle et s'en est donc très fortement approché, ce qui a provoqué un décalage de la base (saut dans la trajectoire) à cause d'une activation de la zone frontale. Le robot a ensuite longé les deux obstacles, puis une fois qu'il les a passés, il s'est progressivement éloigné du mur.

Cette expérience montre que lorsque le robot a le choix du côté de contournement de l'obstacle, car la résultante des actions motrices des zones avant et des zones de côtés est quasiment nulle, le seul facteur qui détermine le choix du côté d'évitement est l'orientation avec laquelle le robot arrive face à l'obstacle.

Lorsque le robot doit contourner deux obstacles dans une pièce (figure 6.17), il commence tout d'abord à naviguer droit devant lui. Lorsque les zones avant (gauche ou droite) détectent l'obstacle, le robot commence à se décaler pour longer l'obstacle. Lorsqu'il a longé l'obstacle, et qu'il se retrouve en face du mur, la zone frontale déclenche un décalage de la base pour permettre au robot de tourner et de se mettre face au passage. Une fois que le robot a tourné, les actions des zones de côté lui permettent de naviguer entre le mur et l'obstacle. Une fois l'obstacle évité, le robot commence à se décaler pour s'éloigner

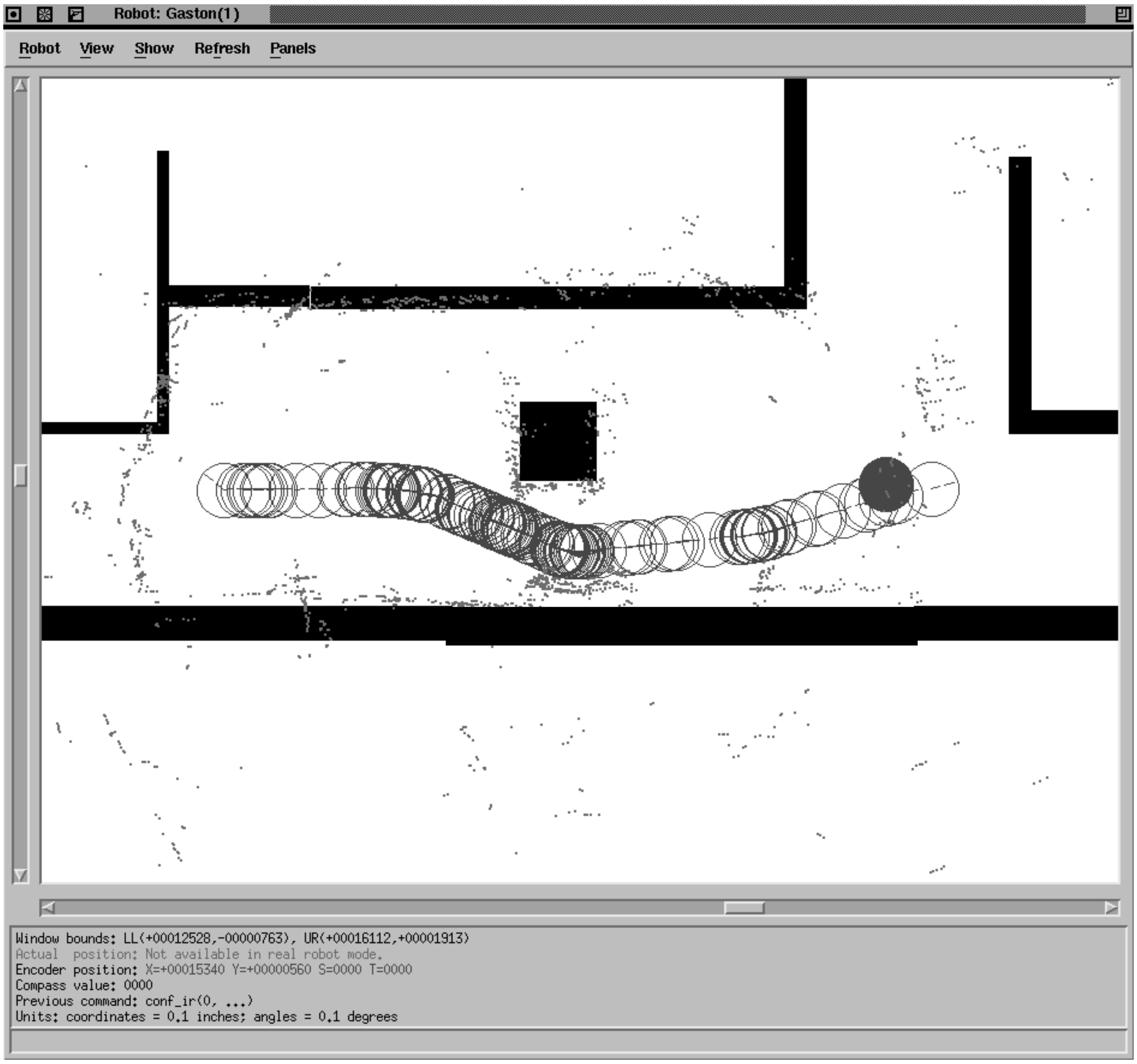


FIG. 6.15 – *Evitement d'un obstacle dans une pièce*

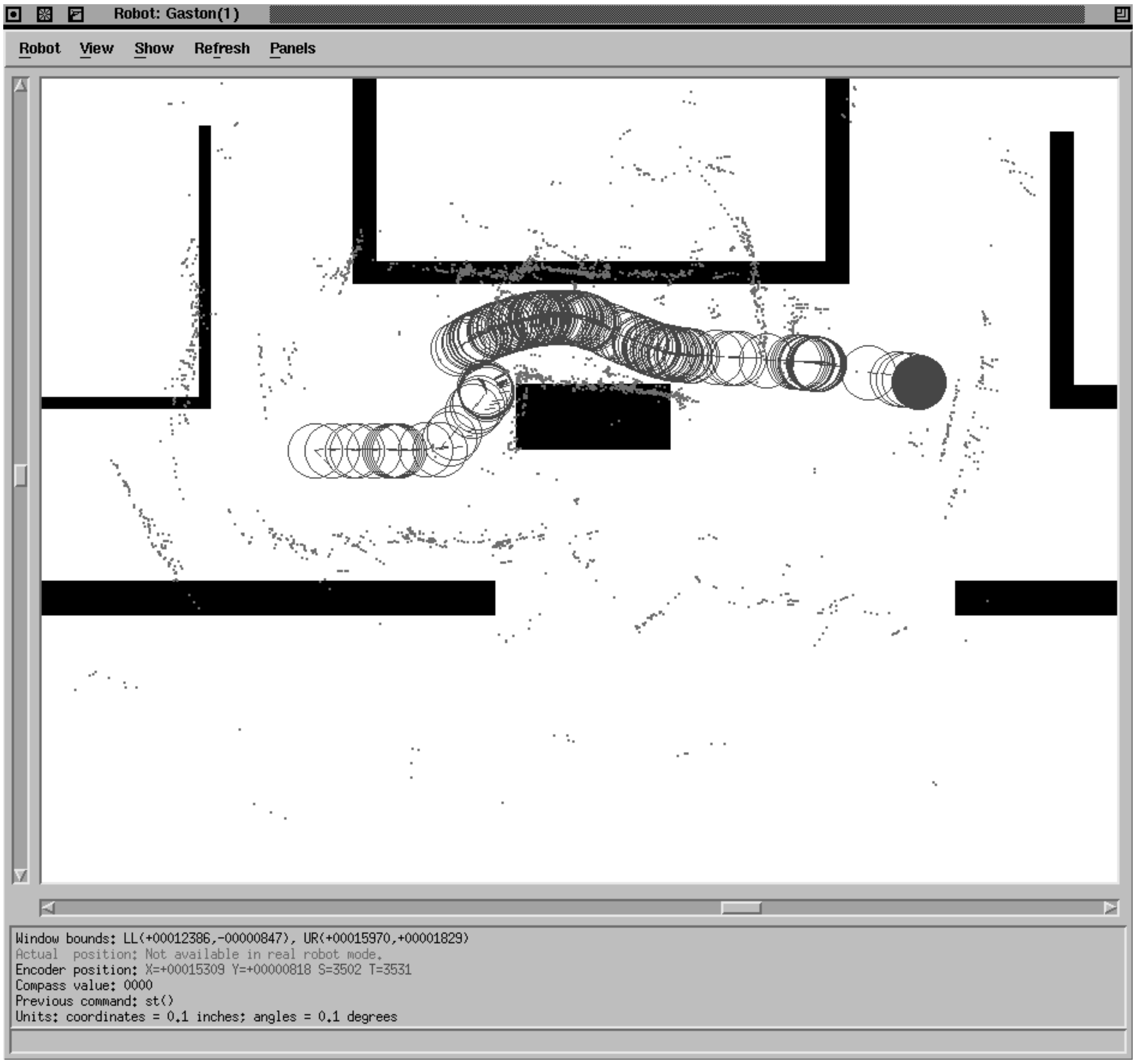


FIG. 6.16 – *Evitement de deux obstacles dans une pièce*

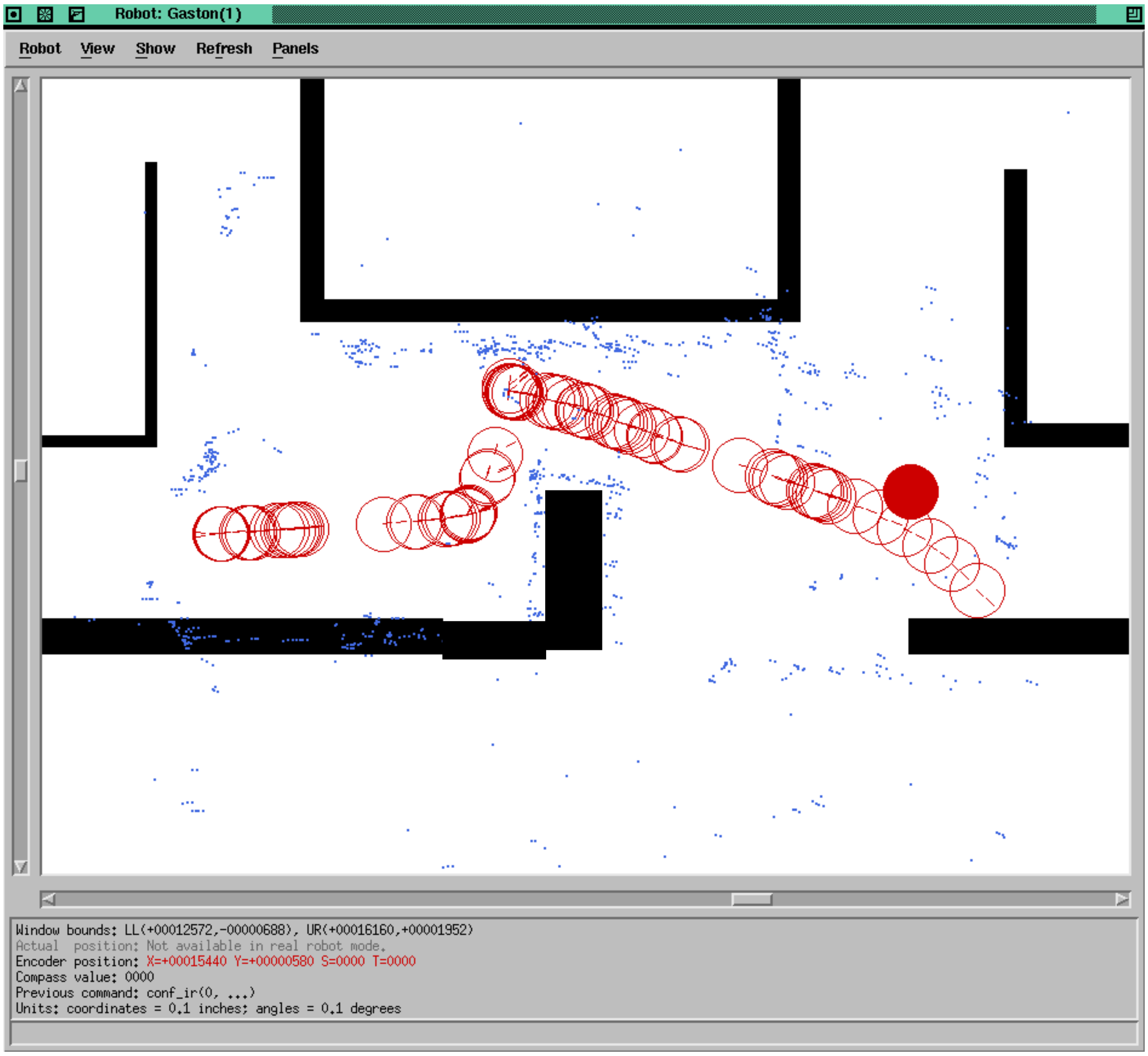


FIG. 6.17 – *Contournement de deux obstacles dans une pièce*

du mur.

Situation d'échec

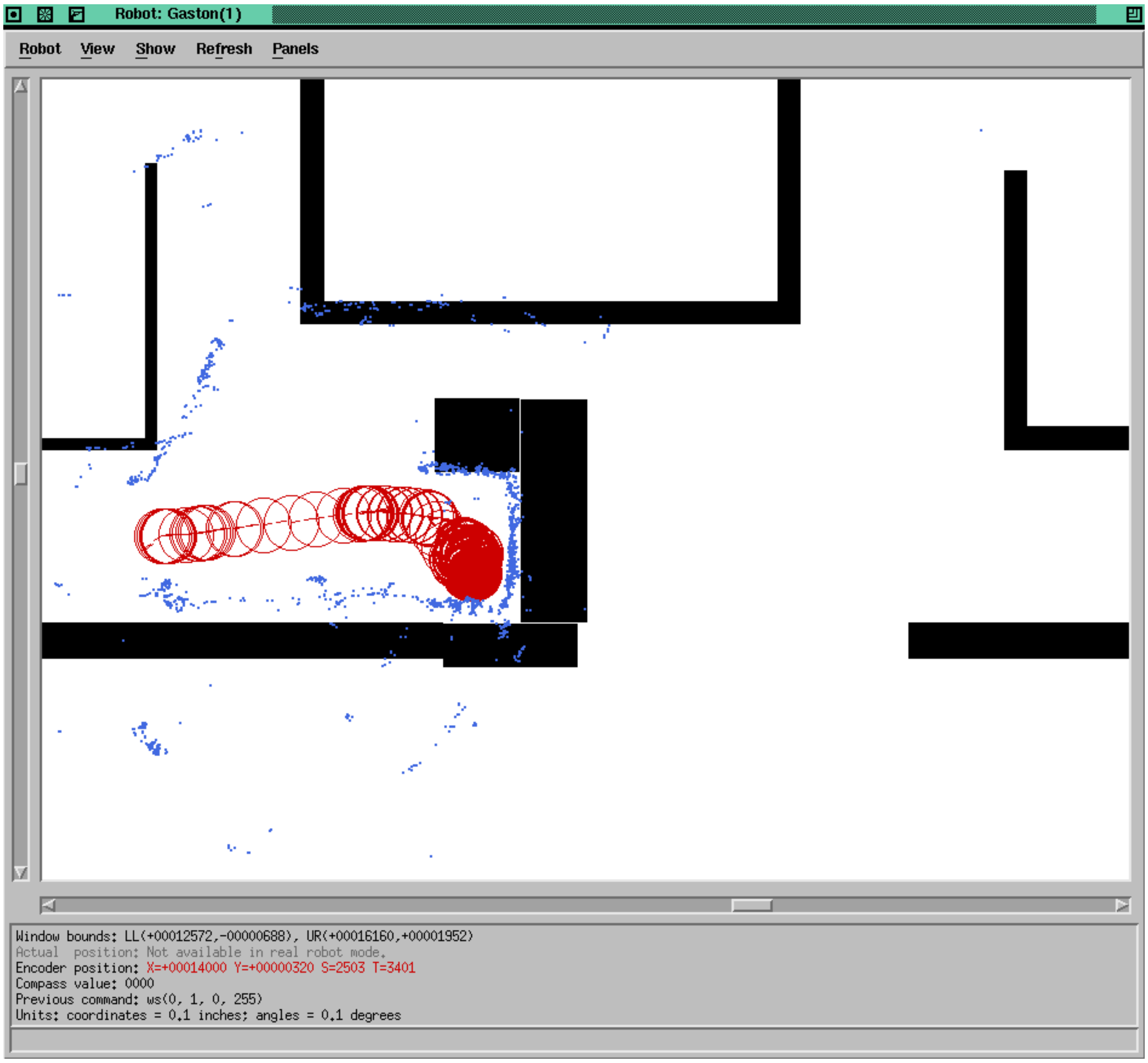


FIG. 6.18 – *Situation d'échec*

Lorsque le robot doit traverser une pièce où il y a un obstacle en forme de U (figure 6.18), il commence par naviguer tout droit. Il cherche ensuite à se décaler pour longer l'obstacle. En longeant l'obstacle, il se retrouve face à l'autre paroi de l'obstacle. Il va donc essayer de contourner l'obstacle en le longeant de l'autre côté. Il se retrouvera à nouveau face à la dernière paroi de l'obstacle. Il va essayer de contourner l'obstacle en le longeant de l'autre côté, ce qui va le ramener dans la situation précédente. Il continuera ainsi indéfiniment.

Cette situation permet de montrer les limites actuelles de notre approche. Le robot a essayé de contourner l'obstacle par la gauche et par la droite sans succès. Il n'a donc plus de solutions, et se retrouve bloqué.

6.3.8 Conclusion

Les expérimentations réalisées montrent que notre approche permet à notre robot de se déplacer dans des couloirs et des pièces tout en évitant des obstacles. L'évitement d'obstacles est particulièrement efficace, puisque le robot n'entre que très rarement en collisions avec les obstacles. L'évitement d'obstacles est réalisée de deux façons :

- Lorsque le robot n'est pas complètement face à l'obstacle, il se sert des zones avants gauche et droite pour se décaler afin d'éviter l'obstacle.
- Lorsque le robot est face à l'obstacle, il utilise les réactions de la zone frontale pour décaler sa base et naviguer en crabe. Cette réaction correspond souvent à des évitements d'urgence, lorsque le robot est à la fois face à un obstacle et en même temps très proche de cet obstacle. Dans ce cas, la position de l'obstacle est postée dans la mémoire partagée du niveau physique.

Ces techniques d'évitement d'obstacles lui permettent de réagir avec succès à de nombreuses autres situations telles que par exemple le rétrécissement de couloir ou le passage de couloir, ce que ne sont pas capables de faire des techniques basées sur les champs de potentiels. D'autre part, la façon dont les obstacles sont évités permet de ne pas à avoir à faire de rotations trop importantes. En effet, les zones de côté et à l'avant ne font faire au robot que des rotations assez faibles. Les rotations plus importantes, quand un obstacle se trouve devant le robot, ne sont faites que par la base. Ces comportements permettent à chaque capteur d'observer à chaque instant à peu près la même portion de l'environnement ce qui est très important pour la réalisation de notre module de localisation, puisque l'interprétation des données des capteurs est basée sur la variations des mesures des capteurs dans le temps.

Par contre, il reste bloqué, lorsqu'il doit éviter un obstacle en U. D'autre part, on constate que le robot est tout à fait capable de traverser un couloir, ou une pièce par contre il est incapable de rejoindre un but précis. Ceci est dû au fait que la notion de but est pour lui inconnue. La section suivante présente les ajouts que nous avons faits pour permettre à notre robot de rejoindre un but précis.

6.4 La prise en compte du but

6.4.1 Introduction

Notre robot mobile est actuellement capable de se déplacer dans un couloir et d'y éviter les obstacles. Comme la notion de but est inconnue, notre robot est incapable d'atteindre un but prédéfini. Nous allons donc ajouter un comportement qui prend en compte un but, et modifier le mécanisme d'intégration en conséquence. Dans les conditions dans lesquelles nous utilisons notre contrôleur flou ce comportement n'est pas indispensable, puisque notre robot navigue dans un environnement intérieur structuré composé quasiment exclusivement de couloirs, et dans ce cas nous avons vu dans la section précédente que le robot peut se contenter de traverser le couloir et d'éviter les obstacles pour arriver à son but. Cette section a pour but d'étendre notre approche afin de pouvoir l'utiliser autre part que dans des couloirs. Elle a aussi pour but de montrer le bien fondé de notre approche, à travers l'ajout d'un comportement de prise en compte du but qui à chaque instant est en compétition avec les comportements d'évitement d'obstacles pour contrôler le robot. L'arbitrage est réalisé grâce au mécanisme d'inhibition et nous voyons dans cette section la combinaison des deux mécanismes d'intégration : fusion et inhibition.

Nous présentons tout d'abord le comportement de prise en compte du but ainsi que les règles qui le définissent. Dans le deuxième paragraphe, nous expliquons comment il s'intègre au deuxième niveau de notre contrôleur flou. Puis nous expérimentons notre contrôleur flou dans des situations d'évitement d'obstacles dans une pièce.

6.4.2 Un nouveau comportement pour prendre en compte un but prédéfini

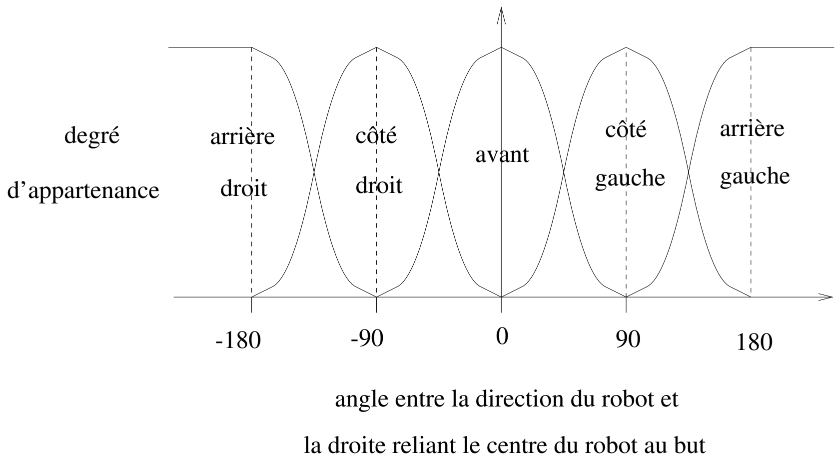


FIG. 6.19 – *Données linguistiques utilisées pour calculer l'angle entre la direction du robot et la droite reliant le centre du robot au but*

Le but de ce comportement est de placer le robot face à son but. Pour concevoir ce nouveau comportement, nous utilisons les informations odométriques pour calculer l'angle entre la direction du robot et la droite reliant le centre du robot au but. Comme l'odométrie n'est pas complètement fiable (section 4.2.5), nous fuzzifions l'angle entre la direction du robot et la droite reliant le centre du robot au but en utilisant des données linguistiques (figure 6.19) et utilisons les données linguistiques de la figure 6.5 pour la vitesse de rotation.

Pour ce comportement, nous désirons les réactions suivantes :

- Un but situé à l'arrière gauche du robot cause une vitesse de rotation rapide vers la gauche.
- Un but situé à gauche du robot cause une vitesse de rotation lente vers la gauche.
- Un but situé à l'arrière droite du robot cause une vitesse de rotation rapide vers la droite.
- Un but situé à droite du robot cause une vitesse de rotation lente vers la droite.

– Un but situé devant le robot ne cause pas de vitesse de rotation.

Donc, les règles sont :

```
si Angle est Arrière gauche alors tourner rapidement à gauche.
si Angle est Gauche alors tourner lentement à gauche.
si Angle est Arrière droite alors tourner rapidement à droite.
si Angle est Droite alors tourner lentement à droite.
Si Angle est Devant alors ne pas tourner.
```

6.4.3 Intégration du nouveau comportement

Etant donné que l'évitement d'obstacles et l'atteinte d'un but sont deux comportements antinomiques, nous ne pouvons pas les fusionner. Donc, nous utilisons une combinaison de deux types de mécanisme d'intégration :

– Les cinq premiers comportements locaux sont fusionnés pour éviter les obstacles dans l'environnement.
– Un mécanisme d'inhibition permet de définir quel comportement (entre les cinq comportements physiques et le comportement pour atteindre le but) contrôlera le robot mobile.

Nous pensons aussi que les cinq comportements physiques locaux sont plus importants que le comportement d'atteinte du but, nous leur donnons donc une priorité plus importante. Ce qui signifie que quand un des cinq comportements physiques locaux renvoie une vitesse de rotation non nulle, le comportement d'atteinte du but est automatiquement inhibé.

La vitesse de translation est toujours calculée à partir de la vitesse de rotation de la même façon que dans la section précédente. Il y a toutefois une petite différence : quand le comportement de prise en compte du but contrôle le robot, et que le robot se rapproche de son but, le robot ralentit.

6.4.4 Résultats

Nous avons implémenté et testé notre contrôleur flou sur notre robot mobile pour atteindre un but prédéfini dans une pièce. Au départ, le robot est positionné dans une position et une orientation connues mais différentes à chaque fois. Il doit atteindre un but prédéfini dans cette pièce. Dans chacune des figures présentées, le robot se déplace de la gauche vers la droite.

Evitement d'obstacles dans une pièce

Dans cette expérience (figure 6.20), le robot se trouve au départ face à son but. Le robot commence par naviguer droit devant lui pour rejoindre son but. Lorsqu'il se rapproche de l'obstacle, il se décale pour l'éviter. Il évitera l'obstacle de la même manière qui l'a fait dans l'expérience de la figure 6.17. Une fois l'obstacle évité, on voit nettement que le robot cherche à se réaligner face à son but. Au départ, il redétecte à nouveau l'obstacle

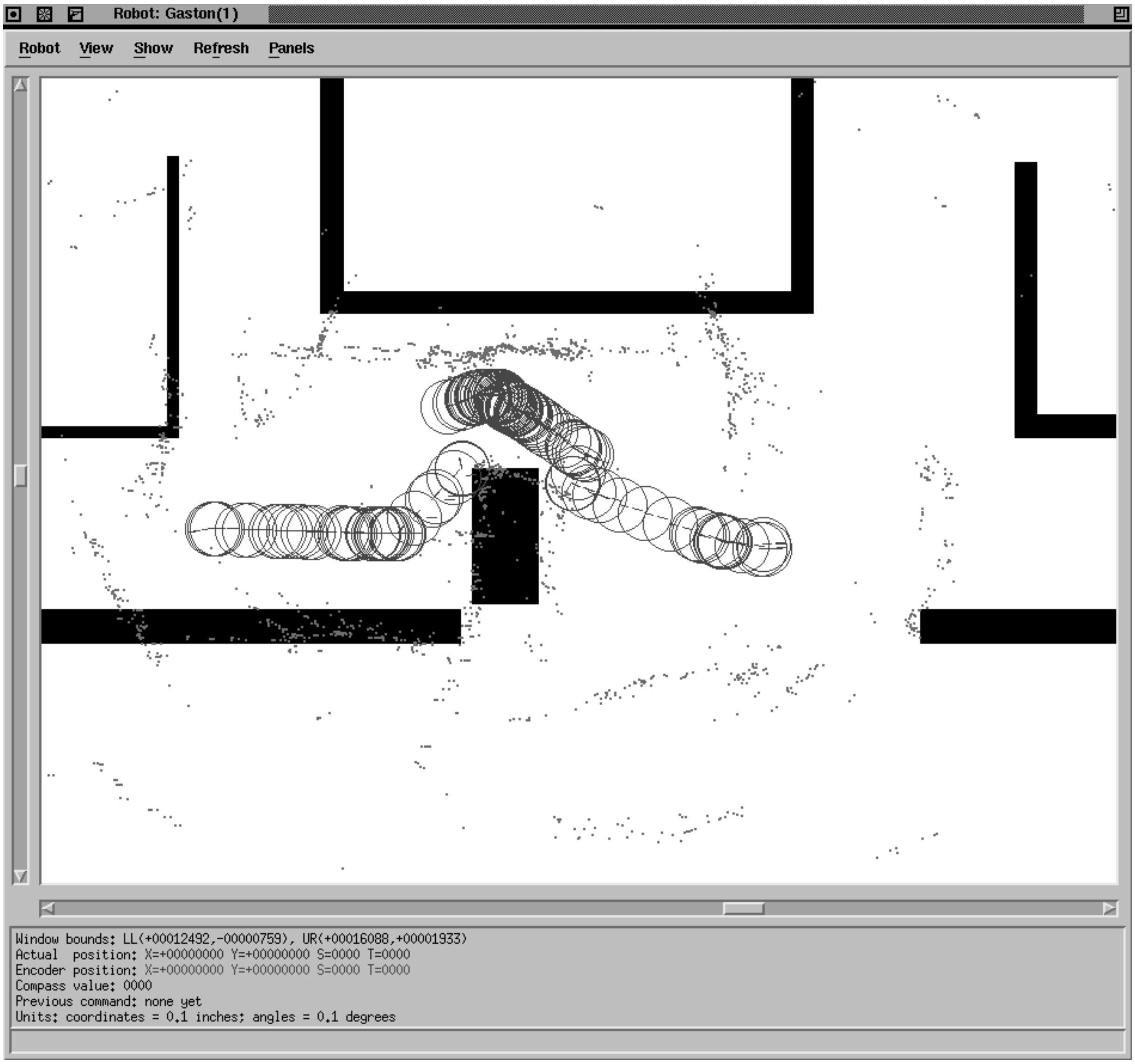


FIG. 6.20 – *Contournement de deux obstacles dans une pièce*

et cherche à l'éviter. Au moment où l'obstacle est derrière lui, le robot se dirige vers son but.

La partie finale (lorsque le robot a contourné l'obstacle) de la trajectoire prise par le robot est très différente selon que le robot cherche à rejoindre son but, ou qu'il cherche à traverser la pièce (figure 6.17), en s'éloignant simplement du mur et en naviguant tout droit ensuite.

FIG. 6.21 – *Contournement d'un groupe d'obstacles dans une pièce*

Dans cette expérience (figure 6.21), le robot doit rejoindre un but qui se trouve derrière les obstacles. Le robot doit dans ce cas contourner tout le groupe d'obstacles pour derrière et en bas du groupe d'obstacles. Le robot commence par naviguer tout droit pour rejoindre son but. Il se décale ensuite pour éviter le premier obstacle, et le longer. Il passe ensuite entre le mur et le deuxième obstacle. Il longe ensuite le troisième obstacle. Arrivé à la fin du troisième obstacle, il cherche à se remettre face à son but. Quand il est face à son but, il se retrouve à nouveau face au troisième obstacle. La réaction de la zone frontale décale la base. Le robot est finalement juste en face de son but, et il lui suffit d'aller tout droit pour le rejoindre.

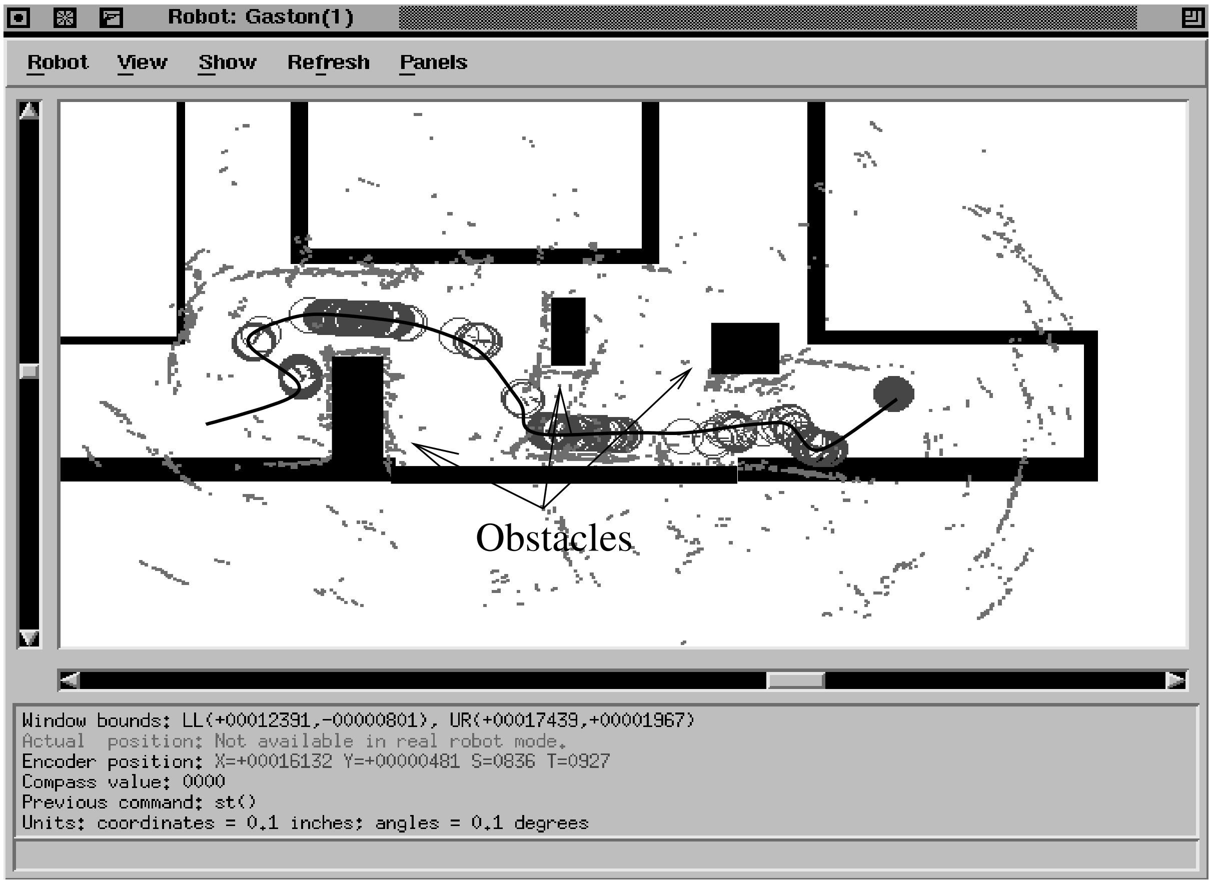


FIG. 6.22 – *Navigation dans une pièce*

Nous avons finalement fait naviguer notre robot mobile dans une pièce où il y a de nombreux obstacles disposés de différentes manières (figure 6.22).

Au départ, lorsqu'il n'y a pas d'obstacles dans l'environnement local du robot mobile, il se place face à son but, et va tout droit jusqu'à ce qu'il détecte un obstacle. A ce moment, il évite l'obstacle et se replace face à son but (figure 6.22). Le robot mobile atteint son but en évitant la position des obstacles et les gens dans le hall.

Le robot commence par avancer tout droit pour rejoindre son but. Il longe ensuite les deux premiers obstacles, puis passe entre eux et le mur. Une fois l'obstacle évité, le robot cherche à rejoindre à nouveau son but. Il se retrouve face à un nouvel obstacle, qu'il évite en décalant sa base. Puis, il navigue pour longer le dernier obstacle, tout en cherchant à s'aligner vers son but.

6.4.5 Conclusion

La prise en compte du but permet à notre robot d'atteindre un but précis et plus non seulement de se placer face à son but et de se déplacer réactivement vers celui-ci.

Elle montre aussi comment nous pouvons intégrer divers types de comportements dans notre contrôleur flou et les mettre en compétition pour contrôler le robot. Les résultats des expérimentations sont encourageants, puisque le robot évite toujours très bien les obstacles, et quand il se trouve dans une zone libre d'obstacles, il se dirige face à son but.

De la même façon que précédemment, il reste toujours bloqué lorsqu'il doit éviter un obstacle en U qui se trouve entre lui et le but à atteindre.

6.5 Conclusion

Dans ce chapitre, nous avons présenté le fonctionnement de notre module de navigation. Son rôle est de permettre au robot d'atteindre un but, tout en évitant les obstacles et sans faire de rotation trop importante de la tourelle. De plus, il poste les obstacles qu'il découvre au fur et à mesure de ses déplacements dans la mémoire partagée du niveau physique. Pour cela, le robot se place premièrement face à son but et deuxièmement se dirige vers son but. La première phase consiste simplement à faire une rotation de la base et de la tourelle de façon à ce que le robot se retrouve face à son but. Dans la deuxième phase, le module de navigation détermine à chaque instant l'action motrice qui le rapproche de son but et évite les obstacles qui se trouvent sur son chemin.

Pour déplacer le robot, nous proposons une nouvelle approche pour concevoir un contrôleur flou en deux niveaux. Cette approche permet la conception d'un contrôleur flou de manière simple. Au premier niveau, des zones locales et des règles locales sont définies pour concevoir des comportements locaux. Au deuxième niveau, un mécanisme d'intégration de ces comportements locaux fournit une action motrice globale au robot mobile. De plus, les comportements locaux peuvent être conçus et testés indépendamment et les mécanismes d'intégrations proposés donnent une cohérence globale au robot.

Notre approche a été intensivement testée et notre robot mobile touche rarement les obstacles ou les personnes. L'évitement d'obstacles se faisant de deux façons différentes, il permet au robot de se décaler lorsqu'il détecte l'obstacle assez tôt, ou carrément de naviguer en crabe pour longer un obstacle très proche.

Avec nos comportements locaux d'évitement des obstacles frontaux, le robot mobile ne fait jamais de rotations excessives pour éviter les obstacles frontaux, donc il ne fait jamais de demi-tour pour éviter un obstacle, et de plus chaque capteur observe toujours la même partie de l'environnement.

De plus, il est capable de passer dans des passages étroits ou de franchir une porte ce qui n'est pas possible avec une approche à base de champs de potentiels, à cause des problèmes de minimums locaux.

Notre robot mobile n'est actuellement pas capable d'atteindre son but, lorsqu'un obstacle en forme de U se trouve sur son chemin. Cette situation de blocage vient du fait que le robot ne peut ni contourner l'obstacle par la gauche ni par la droite. Il faudrait dans ce cas proposer au robot une alternative qui lui permettrait de sortir de cette situation de blocage. Les capteurs infrarouges/ultrasons de l'arrière du robot ne sont actuellement pas utilisés, ils pourraient être très utiles pour une marche arrière, ce qui constituerait une alternative à cette situation de blocage.

Grâce à notre approche, notre robot peut naviguer dans un couloir tout en faisant face à de multiples situations. La prise en compte du but lui permet d'atteindre un endroit précis. Cette prise en compte du but utilise l'odométrie, et ne permet pas toujours d'atteindre le but, à cause de l'accumulation d'erreurs odométriques (voir section 4.2.5) qui rend l'estimée de la position du robot de plus en plus mauvaise au fur et à mesure qu'il se déplace.

Dans notre module de navigation actuel, lorsque le robot doit traverser un couloir, ce qui est quasiment toujours le cas, nous utilisons notre module de navigation sans la prise en compte du but, ce qui évite les problèmes dûs à l'accumulation d'erreurs de l'odométrie. Nous avons vu que notre robot est tout à fait capable de se déplacer dans un couloir sans toucher les obstacles, simplement en le plaçant dans une direction parallèle aux deux murs du couloir, et au milieu du couloir. La position du but à atteindre n'est prise en compte que quand le robot évolue dans une pièce. Le module de navigation détermine automatiquement en fonction de sa position courante, du but à atteindre et de la carte de l'environnement, s'il se trouve dans un couloir ou dans une pièce, et décide donc de prendre en compte ou non la position précise du but à atteindre.

Deuxième partie

Notre module de localisation et de contrôle d'exécution d'actions

7

Introduction

Pour un robot mobile évoluant dans un environnement intérieur structuré, la reconnaissance automatique de balises naturelles (telles que les intersections de couloir, les portes ouvertes...) est importante pour plusieurs raisons :

- Elles contiennent des caractéristiques qu'un robot mobile est capable de reconnaître avec ses capteurs.
- Elles peuvent être utilisées pour la localisation [Jenkin et Jasiobedzki, 1993].
- Elles constituent des points de repères qui permettent à un robot mobile ayant une action à exécuter de vérifier son bon déroulement.
- Elles indiquent la possibilité de nouvelles directions de déplacements. Elles jouent donc un rôle fondamental dans la construction de cartes cognitives [Kortenkamp et Weymouth, 1994].

Pour reconnaître des balises, un robot mobile doit dans un premier temps les apprendre pour être capable de les reconnaître. Dans cette partie, nous décrivons notre approche à base de modèles de Markov cachés pour réaliser l'apprentissage et la reconnaissance de balises naturelles. Puis nous appliquons notre méthode pour concevoir un module de localisation et le contrôleur d'exécution du niveau physique Cette partie est composée de cinq chapitres :

Dans le second chapitre, nous présentons les deux types d'approches généralement utilisées pour réaliser l'apprentissage et la reconnaissance de balises :

- Définir des règles pour expliquer les observations[19], ou construire une représentation symbolique des balises.
- Utiliser les observations du robot mobile passant devant une balise, pour déduire des propriétés statistiques intéressantes afin de construire un modèle de ces observations (phase d'apprentissage). Ce modèle sera ensuite utilisé pour reconnaître des balises (phase de reconnaissance).

Parallèlement à cette classification, nous analysons dans la deuxième partie du chapitre les critères qui permettent de réaliser l'apprentissage et la reconnaissance de balises :

- Le nombre d'observations prises en compte.
- Le nombre de capteurs utilisés.

19. Une observation est définie comme la mesure d'un ou de plusieurs capteurs à un instant donné.

– L'interprétation des données (c-à-d, la balise reconnue).

Cette analyse nous permet de définir quelles sont les conditions à remplir pour apprendre et reconnaître des balises.

Pour terminer le deuxième chapitre, nous analysons les applications de l'apprentissage et de la reconnaissance de balises à la localisation, au contrôle d'exécution d'actions et à la construction de cartes cognitives.

Dans le troisième chapitre, nous présentons la classe particulière de modèles statistiques que nous avons utilisés pour réaliser l'apprentissage et la reconnaissance de balises : les modèles de Markov Cachés ou MMC (en anglais Hidden Markov Models ou HMM). Nous en donnons tout d'abord une définition, puis analysons différentes classes de modèles et les problèmes auxquels ils s'adaptent le mieux. Dans la seconde partie de ce chapitre, nous définissons les modèles d'ordre un. Nous décrivons tout d'abord l'algorithme de reconnaissance, et terminons par l'algorithme d'apprentissage des modèles ainsi que les deux critères généralement utilisés pour réaliser cet apprentissage. Notre approche a été réalisée en utilisant un outil, développé dans l'équipe par Jean-François Mari et Dominique Fohr, basé sur des HMMs d'ordre deux, nous présentons l'extension des modèles pour l'ordre deux.

Dans le quatrième chapitre, nous expliquons comment nous avons appris à reconnaître des balises naturelles à l'aide de modèles de Markov cachés, et donnons les résultats de nos expériences. Pour cela, nous décrivons les différentes étapes qui nous ont permis de réaliser cette approche. Plus précisément, nous choisissons un certain nombre de balises à apprendre et à reconnaître ainsi que le modèle retenu pour les représenter. Ensuite, nous expliquons comment nous avons défini un corpus d'apprentissage pour chaque modèle de balise. Puis, nous détaillons la procédure d'apprentissage des modèles. Pour terminer la première partie de ce chapitre, nous présentons une méthode pour évaluer le taux de reconnaissance et discutons des résultats obtenus. Cette discussion nous permet de mettre en évidence deux problèmes. Dans la dernière partie du cinquième chapitre, nous proposons deux améliorations pour augmenter le taux de reconnaissance.

Pour terminer, pour valider notre approche, nous l'appliquons à la localisation et au contrôle d'exécution d'actions. Ces applications font l'objet du cinquième chapitre de cette partie. Nous modifions tout d'abord l'algorithme de reconnaissance afin qu'il permette la reconnaissance des balises au fur et à mesure qu'elles sont vues. Ensuite, nous expliquons comment nous interprétons chaque balise reconnue afin de savoir à quelle balise de l'environnement elle correspond. A partir de là, nous expliquons comment lorsque le robot a reconnu une balise de son environnement, il se localise par rapport à cette balise. Ceci définit notre module de localisation. Pour terminer, nous revenons sur le contrôleur d'exécution du niveau physique. Nous détaillons son fonctionnement, c'est-à-dire comment il détermine la meilleure action de déplacement lorsque le robot doit se rendre à un but donné dans un couloir.

8

L'apprentissage et la reconnaissance de balises

8.1 Introduction

Dans ce chapitre, nous présentons les diverses approches utilisées pour l'apprentissage et la reconnaissance de balises pour robot mobile. L'apprentissage et la reconnaissance de balises peut être vue de deux manières différentes :

- Expliquer ou représenter les observations. Nous pouvons en effet définir des règles pour expliquer les observations et utiliser ces règles pour reconnaître des balises ou construire une représentation symbolique de balises, et utiliser cette représentation pour reconnaître les balises ainsi représentées.
- Faire des statistiques sur les observations. Pour cela, nous utilisons les observations du robot mobile passant devant une balise, pour déduire des propriétés statistiques intéressantes afin de construire un modèle de ces observations (phase d'apprentissage). Ce modèle est ensuite utilisé pour reconnaître des balises (phase de reconnaissances).

Ces deux approches peuvent être opposées l'une à l'autre. Dans la première approche, nous recherchons à expliquer ou à représenter les observations, alors que dans la seconde approche, nous désirons construire un modèle représentant les propriétés statistiques de ces observations. La présentation de ces différents types d'approche fait l'objet de la section 8.2.

Dans la deuxième partie de ce chapitre, nous analysons en nous appuyant sur les approches présentées trois critères qui permettent de distinguer la façon d'apprendre et de reconnaître des balises :

- Le nombre d'observations prises en compte qui détermine si l'apprentissage et la reconnaissance d'une balise se font de manière locale ou globale.
- Le nombre de capteurs utilisés qui détermine si l'apprentissage et la reconnaissance d'une balise se font de manière partielle ou totale.
- L'interprétation des données (c-à-d, la balise reconnue) qui détermine à chaque fois qu'une balise est reconnue le type de cette balise.

Cette analyse nous permet de définir quelles sont les conditions à remplir pour définir une bonne approche pour apprendre et reconnaître des balises.

Pour terminer, nous analysons dans la section 8.4 comment les approches présentées ont été appliquées à la localisation, au contrôle d'exécution d'actions ou à la construction de cartes cognitives.

8.2 L'apprentissage et la reconnaissance de balises

8.2.1 Introduction

Nous présentons dans les trois paragraphes suivants les diverses approches utilisées pour apprendre et reconnaître des balises : une technique qui permet d'expliquer les observations en définissant des règles, une technique de représentation des balises fait l'objet du deuxième paragraphe, puis par opposition aux deux premiers paragraphes, le dernier paragraphe montre comment nous pouvons déduire des propriétés des observations simplement en faisant des statistiques sur les observations.

Pour chaque paragraphe nous définissons tout d'abord la façon d'apprendre les balises puis de les reconnaître. De plus chaque approche est tirée d'un exemple :

- Pour expliquer les observations, nous analysons la méthode de [Kortenkamp et Weymouth, 1994] qui définit des règles sur la variation des capteurs pour reconnaître les balises.
- Pour représenter les observations, nous expliquons comment [Yamauchi, 1996b] utilise des grilles d'évidence pour construire une représentation des observations, et définit un algorithme pour reconnaître les balises apprises.
- Pour déduire des propriétés des observations, [Rompais et Pradet, 1995] entraînent un réseau de neurones à partir d'un corpus d'observations de balises, et s'en sert pour reconnaître les balises apprises.

8.2.2 Expliquer les observations

Introduction

Nous présentons dans ce paragraphe comment, en expliquant les observations faites par un robot mobile, nous pouvons définir des règles pour reconnaître des balises. Cet exemple est tiré des travaux de Kortenkamp qui utilise un robot mobile équipé d'une ceinture de capteurs ultrasons, et de deux caméras vidéo. Les règles sont basées sur les variations des capteurs ultrasons afin de reconnaître un type de balise, et les caméras servent à différencier deux balises de même type.

L'apprentissage de balises

Kortenkamp [Kortenkamp et Weymouth, 1994] définit des règles sur la variation des capteurs ultrasons pour reconnaître différents types de balises et ajoute une information visuelle pour reconnaître deux balises de même type. Il considère en fait deux types de

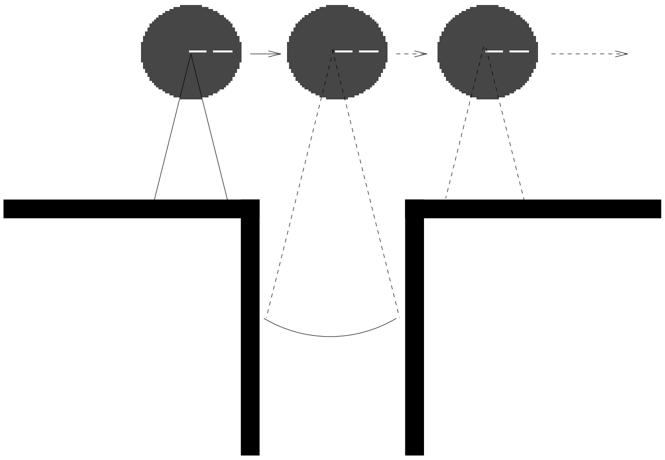


FIG. 8.1 – *Passage devant une balise*

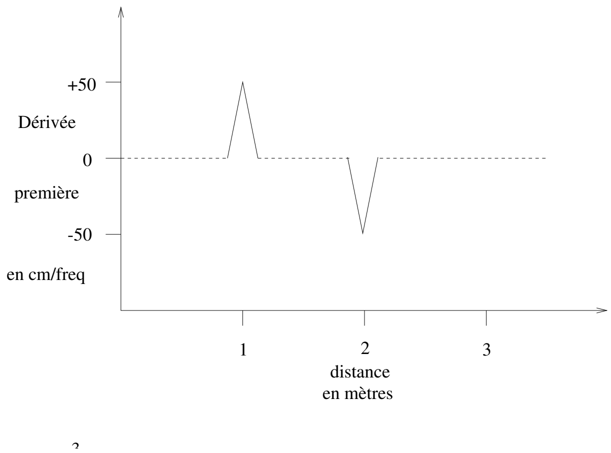


FIG. 8.2 – *Valeur dérivée correspondante au passage devant une balise*

balises (des intersections de couloir, des portes ouvertes) dans un environnement intérieur structuré, qu'il détecte de la même manière (figure 8.1) :

- Le robot s'aligne le long d'un mur en utilisant ses capteurs ultrasons.
- Le robot navigue le long du mur, tout en restant à une distance constante du mur.
- Le robot analyse constamment les valeurs du capteur perpendiculaire au mur situé sur la droite du robot et du capteur perpendiculaire au mur situé sur la gauche du robot pour détecter les ouvertures.
- Quand une ouverture est détectée, le robot continue à longer le mur pour trouver une fermeture correspondant à cette ouverture.
- Lorsque la fermeture est détectée, le robot détermine si l'ouverture est suffisamment importante pour qu'il passe. Si oui, il indique qu'il a reconnu une balise.

La détection des ouvertures et des fermetures est faite en mesurant sur le même capteur la différence entre deux instants (figure 8.2). Une ouverture est caractérisée par une brusque augmentation de la valeur du capteur perpendiculaire au mur. Une fermeture est caractérisée par une brusque diminution de la valeur du capteur perpendiculaire au mur.

Le robot stocke des informations visuelles associées à chaque balise pour pouvoir la reconnaître. En effet, les règles précédentes permettent de détecter toutes les balises, mais ne permettent pas de faire la différence entre elles. Pour faire cette différence, des informations visuelles sont associées à chaque balise. Ces informations visuelles représentent la scène correspondant à chaque balise. De cette scène, on extrait et on stocke des caractéristiques visuelles. On tient aussi compte de leurs positions dans la scène. Les informations visuelles extraites sont des contours verticaux, car ils se sont avérés très utiles en tant que caractéristiques visuelles dans un environnement intérieur, et ceci dans de nombreuses applications de robotique mobile [Kriegman et Binford, 1989] et [Crowley et Sarachik, 1991]. De plus, ils se sont avérés particulièrement efficaces dans l'environnement utilisé à cause du fort contraste entre les portes noires et les murs blancs. Un détecteur de contours de Sobel modifié extrait les contours verticaux marqués. Une deuxième image translatée de dix huit centimètres est analysée de la même manière. Les deux listes de contours sont fusionnées en utilisant la direction de transition du contour (est ce que le contour fait passer d'une zone sombre à une zone claire, ou d'une zone claire à une zone sombre ?), la longueur et la position. Le décalage dans les contours mis en correspondance est utilisé pour déterminer une distance grossière entre le robot et le contour. Les informations visuelles sont stockées dans des grilles de cinq lignes sur cinq colonnes. La taille de la grille choisie a été calculée en fonction de la longueur focale de la caméra. Chaque cellule permet de savoir si à la position correspondante dans l'image traitée, il y a un point noir ou blanc. On associe de plus à chaque cellule, les informations de longueur, distance et direction. Pour représenter une scène complète, le robot utilise huit grilles (figure 8.3).

La reconnaissance de balises

Pour reconnaître une balise déjà visitée, Kortenkamp utilise les deux sources d'information dont il dispose, et les combine en utilisant un simple réseau probabiliste [Charniak, 1991], dans lequel une balise est déterminée par une scène et une caractérisation par les capteurs. La table 8.1 montre les résultats de la reconnaissance en combinant les capteurs

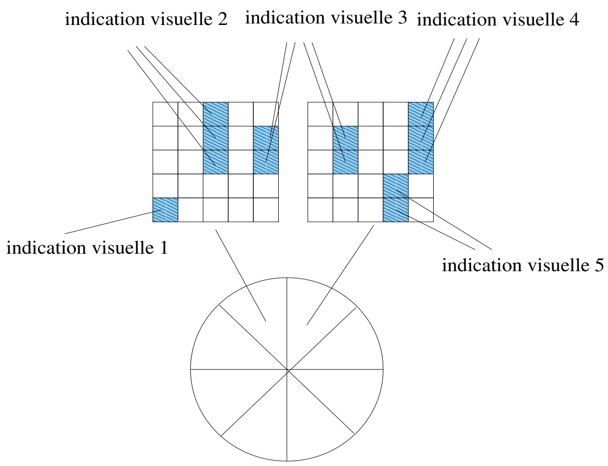


FIG. 8.3 – *Représentation des informations visuelles contenues dans une scène*

	A	B	C	D	E	F	G
A	0.95	0.01	0.02	0.01	0.01	0	0
B	0	0.65	0.26	0.07	0.01	0	0
C	0	0.17	0.52	0.29	0.01	0	0
D	0.01	0.07	0.33	0.58	0.01	0	0
E	0.04	0.12	0.12	0.12	0.61	0	0
F	0	0	0	0	0	0.90	0.10
G	0	0	0	0	0	0.09	0.91

TAB. 8.1 – *Maximum de vraisemblance en utilisant les capteurs ultrasons et la vision*

avec les informations visuelles.

La mise en correspondance entre la scène actuelle et une scène déjà vue a été réalisée de quatre manières différentes :

- Une mise en correspondance "segment à segment" en utilisant la distance et la direction, et où chaque segment de la scène courante doit correspondre à un segment de la scène vue précédemment. Un segment est constitué par un ensemble de cellules (au moins deux), où chaque cellule a au moins une cellule adjacente.
- Une mise en correspondance "cellule à cellule" en utilisant pour chaque cellule occupée la distance et la direction, et où chaque cellule de la scène courante doit correspondre à une cellule de la scène vue précédemment.
- Une mise en correspondance "cellule à cellule" en utilisant pour chaque cellule occupée seulement la direction.
- Une mise en correspondance "cellule à cellule" en comparant seulement les cellules occupées.

La scène actuelle correspond toujours à la scène stockée qui a obtenu le plus grand nombre de mis en correspondance.

Conclusion

L'approche de [Kortenkamp et Weymouth, 1994] paraît intéressante puisqu'elle permet de définir de manière intuitive la suite d'événements temporels qui caractérise une balise. Mais le robot navigue dans un environnement complètement statique où il peut rester à une distance constante de chaque mur, grâce à un simple algorithme de suivi de murs. Dans un cadre plus réaliste avec des obstacles à éviter, le robot ne peut plus rester constamment à la même distance des deux murs et avec une orientation parallèle à ces deux murs, donc le seuil (fixé expérimentalement) utilisé pour détecter les ouvertures devient inutilisable. De plus, le nombre de balises détectables est très limité. En fait, [Kortenkamp et Weymouth, 1994] ne détecte que les ouvertures à l'aide des capteurs. Les règles sont incapables de faire la différence entre une porte ouverte et une intersection. D'autre part, elles ne permettent pas de reconnaître précisement une balise mais un type de balise, la balise est reconnue grâce à la vision.

8.2.3 Représenter les observations

Introduction

Nous présentons dans ce paragraphe comment en représentant les observations faites par un robot mobile, nous pouvons construire une représentation des balises, et utiliser cette représentation pour reconnaître les balises. Cet exemple est tiré des travaux de [Yamauchi, 1996b] qui utilise un robot mobile Nomad200 et se sert des valeurs des capteurs ultrasons pour représenter une balise.

L'apprentissage de balises

Yamauchi [Yamauchi, 1996b] utilise des grilles d'évidence [Elfes, 1989] d'une taille d'environ 1,2 x 1,2 mètres pour représenter les balises. Les grilles d'évidence sont une méthode de représentation spatiale. Une région locale de l'espace est subdivisée en cellules dans une grille cartésienne, et chaque cellule contient une valeur représentant l'estimation de la probabilité d'occupation de cette cellule par un obstacle. Les données des capteurs ultrasons sont utilisées pour mettre à jour la grille d'évidence en se servant d'une règle de mise à jour Bayésienne basée sur un modèle particulier des capteurs. Par exemple, si le signal d'une largueur de 22,5 degré émis par le capteur ultrason détecte un obstacle à une certaine distance, alors toutes les cellules sur l'arc auront leur probabilité d'occupation augmentée, et toutes les cellules à l'intérieur de l'arc auront leur probabilité d'occupation réduite. Chaque balise est donc représentée par une grille d'évidence de 1,2 x 1,2 mètres.

La reconnaissance de balises

Yamauchi [Yamauchi, 1997] définit un algorithme qui permet de mesurer le niveau de mise en correspondance entre deux grilles d'évidences (une grille précédemment apprise, et une grille qui vient d'être construite par exemple). Dans chacune des grilles, pour chaque cellule, on mesure la similitude de probabilité d'occupation. Cette mesure de similarité est donnée par la formule suivante :

$$M_{x,y} = \sum_{\forall x,y} \begin{cases} 1 & \text{si } \frac{p_{x,y}-p_0}{p'_{x,y}-p_0} > 0 \\ 0 & \text{sinon} \end{cases}$$

où M est la mesure de similarité. p_0 est la probabilité d'occupation initiale d'une cellule, $p_{x,y}$ est la probabilité d'occupation de la cellule (x, y) dans la grille apprise, et $p'_{x,y}$ est la probabilité d'occupation de la cellule (x, y) dans la grille qui vient d'être construite.
Comme toutes les cellules sont initialisées avec la valeur p_0 avant que les ultrasons soient utilisés pour mettre à jour les grilles d'occupation. L'algorithme mesure la similarité entre les deux grilles, en comparant les cellules ayant les mêmes coordonnées. Si deux cellules ont une probabilité d'occupation supérieure à la probabilité d'occupation initiale, alors la mesure de similarité est augmentée. Si deux cellules ont une probabilité d'occupation inférieure à la probabilité d'occupation initiale, alors la mesure de similarité est augmentée. Sinon, les deux cellules n'ont pas d'influence sur la mesure de similarité. La grille reconnue sera la grille qui aura la plus haute mesure de similarité des grilles apprises.

Conclusion

Le gros avantage de cette méthode est que l'apprentissage des balises est autonome, puisque la représentation de chaque balise à apprendre est réalisée en mettant à jour le modèle de chaque balise, grâce à l'algorithme de construction d'une grille d'évidence.

Lors de la reconnaissance de balises, on ne sait pas quand se termine la construction de la balise à reconnaître, et quand commence la reconnaissance de la balise construite. De plus, il n'y a aucune interprétation des données. Contrairement à [Kortenkamp et Weymouth, 1994], l'approche de [Yamauchi, 1996a] reconnaît une balise précise mais pas un type de balise précise.

8.2.4 Faire des statistiques sur les observations

Introduction

Rompais [Rompais et Pradet, 1995] apprend à un réseau de neurones à reconnaître des balises dans un environnement intérieur structuré à partir des données d'un télémètre laser, et utilise ce réseau pour reconnaître les balises apprises précédemment.

L'apprentissage des balises

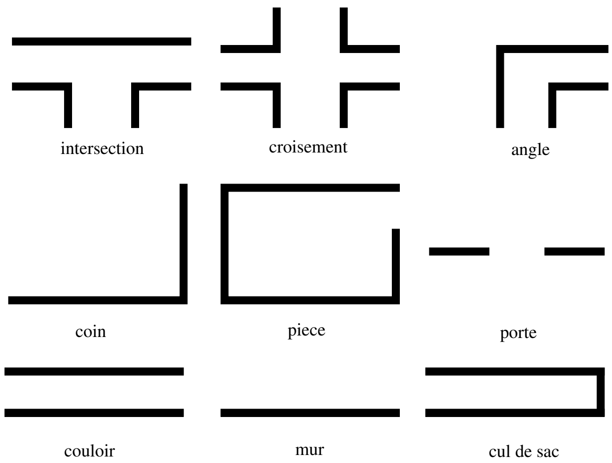


FIG. 8.4 – *Les balises à reconnaître*

Le réseau de neurones est entraîné pour apprendre à distinguer neuf balises (figure 8.4). Pour chaque balise, le robot est placé dans huit positions différentes. Pour chaque position, l'orientation du robot est choisie au hasard. Ensuite, seize rotations de 22,5 degré sont effectuées. Le corpus d'apprentissage est donc constitué de 1152 exemples. Le réseau de neurones comprend seize entrées correspondant aux seize rotations et neuf sorties correspondant aux neuf balises à reconnaître.

	données brutes	transformation linéaire	centrage/réduction
couloir	48	48	30
mur	48	48	48
cul de sac	48	48	48
coin	48	48	48
croisement	34	44	48
intersection	47	46	48
pièce	48	48	48
angle	48	47	46
porte	48	48	47
% reconnu	96,5	98,4	96,1

TAB. 8.2 – *Reconnaissance avec 48 données d'apprentissage*

	données brutes	transformation linéaire	centrage/réduction
couloir	8	10	9
mur	12	11	12
cul de sac	12	12	12
coin	9	10	10
croisement	6	8	9
intersection	6	6	6
pièce	12	12	12
angle	9	8	8
porte	12	11	12
% reconnu	79,6	81,5	83,3

TAB. 8.3 – *Reconnaissance avec douze données de test*

La reconnaissance des balises

Le réseau entraîné est utilisé pour reconnaître les neuf balises, à partir de deux types de données (table 8.2 : quarante huit données extraites du corpus d'apprentissage et table 8.2 : douze données acquises à part). Les résultats sont donnés avec des prétraitements différents sur les données d'entrée :

– Les données brutes.
– Une transformation linéaire des données pour qu'elles se trouvent dans l'intervalle [-1, +1].
– Les données centrées et réduites. Ceci est obtenu en soustrayant à chaque donnée d'entrée la moyenne arithmétique des 16 données composantes l'entrée, et en divisant le résultat par la déviation standard de la donnée.

Globalement, les résultats des reconnaissances sont bons (environ 80 %, pour les données de tests).

Indécisions	2
Erreurs mineures	3
Erreurs majeures	12

TAB. 8.4 – *Reconnaissance dans les couloirs du laboratoire*

Dans une seconde expérience, le robot navigue dans les couloirs du laboratoire, et on lui demande à chaque instant de dire quelle balise il voit. Deux types d'erreurs sont considérées :

– Des erreurs mineures. Une erreur mineure correspond à une confusion entre deux situations sans conséquence grave pour les déplacements du robot (par exemple, une confusion entre un angle et un coin).
– Des erreurs majeures. Une erreur majeure est l'indication d'une intersection dans un couloir, ou inversement. Ces erreurs peuvent avoir de graves conséquences sur les déplacements du robot.

- On considère les indécisions, en plus de ces deux types d'erreurs. C'est-à-dire les cas, où les valeurs de sorties de deux neurones (ou plus) sont très proches et donc où le choix de la balise reconnue est indécis.

Le résultat des balises reconnues (au nombre de cent vingt) lors du parcours du robot est donné par la table 8.4. Les résultats sont très bons puisqu'on ne compte que six erreurs graves et deux indécisions sur cent vingt réponses, ce qui fait un taux de reconnaissance de 93,3 %.

Conclusion

Dans cette approche, les balises sont reconnues d'une manière très locale puisque les entrées du réseau sont les mesures du télémètre laser pour chaque rotation. Pour cette raison, on se demande comment sont pris en compte les obstacles, et quel est leur influence sur la reconnaissance. D'autre part, le choix d'un seul réseau pour apprendre et reconnaître l'ensemble des balises est contestable. En effet, il aurait certainement été plus judicieux de choisir neuf réseaux pour apprendre et reconnaître chacune des balises, et les mettre en compétition lors de la reconnaissance. Mais dans ce cas, il faut certainement un corpus plus important que celui utilisé.

8.2.5 Conclusion

Nous avons présenté dans cette section les deux grandes techniques existantes pour apprendre et reconnaître des balises. Elles s'opposent l'une à l'autre. Dans la première, nous recherchons à expliquer ou à représenter les observations, alors que dans la seconde, nous désirons construire un modèle représentant les propriétés statistiques de ces observations.

Au delà de cette opposition, nous pouvons tirer quelques conclusions :

- La technique de [Kortenkamp et Weymouth, 1994] nous paraît intéressante puisqu'elle permet de définir intuitivement des règles pour apprendre et reconnaître les balises, mais elles ne caractérisent qu'un nombre très faible de balises.
- En représentant les observations, l'apprentissage est autonome, et de plus l'apprentissage permet de représenter n'importe quelle balise, puisque la représentation est associée à une balise et pas un type de balise. Par contre, cette technique est incapable de déterminer une fois qu'une balise est reconnue à quelle type de balises symboliques elle correspond.
- La recherche de propriétés statistiques permet à partir d'un corpus d'apprentissage de déduire les caractéristiques d'une balise. Cette technique est plus robuste au bruit puisqu'elle extrait automatiquement les caractéristiques d'une balise en incluant le bruit, mais est aussi plus longue à mettre en œuvre. Il faut aussi noter que cette technique est moins utilisée que la précédente. A notre connaissance, il n'existe pas d'ailleurs d'autres méthodes que celles que nous avons présentées, telles que par exemple une méthode basée sur un apprentissage non supervisé des balises ou une prise en compte du corpus d'une manière plus globale (c-à-d, pas avec une seule observation).

On pourrait s'intéresser à une approche à mi-chemin entre ces deux extrêmes. En effet, une approche telle que celle de [Kortenkamp et Weymouth, 1994] est facile à mettre en œuvre et l'apprentissage et la reconnaissance d'un type de balise sont basés sur la suite des événements qui la caractérisent. Mais la détection des événements par des règles rigides ne permet de l'apprendre et de la reconnaître que dans un contexte limité. Plutôt que de fixer des règles rigides permettant de reconnaître chaque balise, il peut être intéressant d'essayer d'apprendre la suite d'événements qui la caractérise. C'est le but de notre approche. En effet, une des principales propriétés des modèles de Markov cachés est leur capacité à construire des modèles stochastiques sur des suites temporelles d'événements sans rapport les uns avec les autres. Nous les utilisons donc pour apprendre et reconnaître la suite d'événements qui caractérisent une balise.

D'autre part, il nous faut définir un certain nombre de critères pour notre approche, tels que le nombre d'observations pris en compte, le nombre de capteurs utilisés ou l'interprétation des données. Cette analyse fait l'objet de la section suivante.

8.3 Critères retenus pour l'apprentissage et la reconnaissance de balises

8.3.1 Introduction

Il existe, parallèlement aux deux types d'approches utilisées pour l'apprentissage et la reconnaissance de balises, des critères qui permettent de définir la façon de réaliser l'apprentissage et la reconnaissance de balises. Nous citons ici trois critères qui nous permettront de mettre en évidence comment réaliser l'apprentissage et la reconnaissance de balises :

- L'apprentissage et la reconnaissance de balises peuvent être vus soit en considérant les balises de manière locale (c-à-d, une balise est apprise et reconnue en utilisant une observation), soit en considérant les balises de manière globale (c-à-d, une balise est apprise et reconnue en utilisant une suite d'observations).
- On note aussi que certaines approches présentées n'utilisent qu'un capteur alors que d'autres utilisent un ensemble (voire la totalité) des capteurs.
- Nous avons vu aussi que, selon la méthode adoptée, la reconnaissance est associée à une interprétation symbolique des données qui permet de connaître le type de la balise reconnue (c-à-d, une porte ouverte, une intersection...).

Dans les trois paragraphes suivants, nous analysons comment ces critères ont été pris en compte dans les exemples de la section 8.2.

8.3.2 Le nombre d'observations utilisées

Le nombre d'observations utilisées permet de définir la façon dont nous apprenons et reconnaissons une balise. Si nous utilisons un nombre d'observations assez faible, la balise est apprise et reconnue de manière locale, en considérant que la partie observée est suffisante. Au contraire, dans le cas où le nombre d'observations est assez élevé, la balise est apprise et reconnue de manière globale (c-à-d, en considérant toutes les observations lorsque le robot passe devant la balise).

Dans leur approche, [Kortenkamp et Weymouth, 1994] utilisent une séquence d'observations. Il considère les balises de manière globale, puisque la reconnaissance démarre au moment où le robot passe devant le début de la balise (brusque augmentation sur le capteur perpendiculaire au mur due au début de l'ouverture), et se termine au moment où le robot arrive à la fin de la balise (brusque diminution sur le capteur perpendiculaire au mur due à la fin de l'ouverture). Le nombre d'observations utilisées n'est pas toujours le même et va dépendre du nombre d'observations que le robot fait entre le moment où il commence à reconnaître l'ouverture et où il finit de la reconnaître. De la même manière, [Yamauchi, 1996a] utilise un nombre important d'observations pour construire une représentation géométrique d'une balise. Par contre, nous ne savons pas si le nombre d'observations utilisé est toujours le même, c'est-à-dire si Yamauchi utilise un nombre fixe d'observations pour représenter une balise ou s'il est variable et dépend par exemple d'un critère qui définit quand la construction de la représentation géométrique du modèle est terminé. Il en est de même pour la reconnaissance. A l'opposé, [Rompais et Pradet, 1995] utilisent une observation pour l'apprentissage et la reconnaissance de balises. Cette méthode ne permet qu'un apprentissage et une reconnaissance très local des balises. Il paraît difficile d'imaginer comment une balise est reconnue lorsque lors de l'observation un obstacle est présent.

Nous pensons qu'il vaut mieux utiliser une séquence d'observations pour apprendre et reconnaître une balise naturelle, et que le nombre d'observations constituant cette séquence doit être variable. Ainsi nous prenons en compte la balise dans sa globalité, ce qui la rend plus robuste au bruit. Pour la même raison, une balise pourrait être reconnue même si un obstacle est présent lors de la reconnaissance.

8.3.3 Le nombre de capteurs utilisés

Le nombre de capteurs utilisés pour constituer l'observation définit les parties de la balise que l'on observe pour l'apprendre et la reconnaître. Une observation est définie comme la valeur d'un ou de plusieurs capteurs à un instant donné. Le nombre de capteurs constitue en fait l'observation. En utilisant un nombre de capteurs assez faible, la balise est apprise et reconnue de manière partielle, en considérant que cette vue partielle, qui ne correspond qu'à une partie de l'observation lorsque le robot passe devant une balise, est suffisante. Au contraire, dans le cas où le nombre de capteurs est assez élevé, la balise est apprise et reconnue de manière globale (c-à-d, en considérant tous les capteurs lorsque le robot passe devant une balise).

Kortenkamp considère les balises de manière très partielle, puisqu'il ne s'intéresse qu'au capteur perpendiculaire à chaque mur du couloir. En effet, il ne considère que les balises qui provoquent des fortes variations sur ce capteur. Contrairement à cela, [Yamauchi, 1996a] et [Rompais et Pradet, 1995] utilisent tous les capteurs pour apprendre et reconnaître des balises. Ils cherchent à apprendre et reconnaître des balises assez variées et dont le nombre est assez important.

A notre avis, plus le nombre de capteurs utilisés est important, plus le type des balises est variée et le nombre de balises élevé.

8.3.4 L'interprétation des données

L'interprétation des observations permet une fois la balise reconnue de donner une interprétation symbolique de cette reconnaissance, c'est-à-dire de déterminer le type de balise que nous venons de reconnaître.

Yamauchi se contente de construire une représentation des observations, mais ne donne à aucun moment une interprétation symbolique des balises reconnues. Par contre, Kortenkamp et Rompais définissent un certain nombre de balises et c'est à partir de ces balises qu'ils définissent comment apprendre et reconnaître des balises. A chaque reconnaissance, ils associent donc une interprétation symbolique des données.

Comme nous l'avons déjà dit, notre but est d'utiliser les balises naturelles pour déterminer l'état de l'environnement. Nous avons donc besoin de connaître, à chaque fois qu'une balise est reconnue, l'interprétation symbolique de cette balise.

8.3.5 Conclusion

Par rapport aux trois critères retenus pour l'apprentissage et la reconnaissance de balises, nous pensons qu'une bonne approche doit prendre en compte les mesures de tous les capteurs pour toutes les observations réalisées lorsque le robot passe devant une balise. De plus, pour appliquer notre approche à la localisation et au contrôle d'exécution d'un déplacement, nous avons besoin d'avoir une interprétation symbolique des observations.

8.4 Applications

8.4.1 Introduction

Les trois utilisations de l'apprentissage et de la reconnaissance de balises sont :

- La localisation qui permet au robot de savoir où il se trouve dans son environnement.
- Le contrôle d'exécution d'actions, qui à partir des balises reconnues détermine si l'action s'exécute comme prévu lors de la planification, et choisit la meilleure action à réaliser pour atteindre le but.
- La construction de cartes topologiques, à partir des balises reconnues et des chemins qui existent entre elles.

Dans cette section, nous expliquons comment l'apprentissage et la reconnaissance de balises sont utilisés pour localiser un robot mobile, puis nous nous intéressons au contrôle d'exécution d'actions et en particulier aux processus de décision Markoviens partiellement observables (POMDP) qui permettent à chaque instant en fonction du contexte et du but à atteindre de déterminer la meilleure action à exécuter. Pour terminer, nous décrivons le système ELDEN qui permet de construire une carte topologique de l'environnement.

8.4.2 La localisation

Quand un robot mobile se déplace, la connaissance de sa position et de son orientation relativement à son environnement est très utile. Ces informations sont cruciales pour savoir si un but prédéfini a été atteint, ou pour connaître la position d'un robot mobile sur un chemin prédéfini.

Comme nous l'avons présenté dans la section 4.2.5, l'odométrie est une méthode simple qui intègre les translations et les rotations des roues pour déterminer la position cartésienne du robot. Cependant, à cause des glissements des roues du robot sur le sol, les translations et les rotations mesurées par les encodeurs des roues ne reflètent pas exactement les actions du robot, donc l'estimée de la position du robot se dégrade progressivement.

De nombreuses études ont été faites pour éviter cette dérive. Ces techniques sont généralement appelées techniques de recalage ou de localisation [20]. Ceci peut être fait de deux manières différentes [Feng *et al.*, 1994] :

– Dans la première technique (figure 8.5) appelée "localisation par rapport à une carte", le robot utilise ses capteurs pour créer une carte de son environnement local. Cette carte locale est comparée avec une carte globale stockée auparavant dans la mémoire. Si une mise en correspondance est trouvée, alors le robot peut calculer sa position et son orientation dans l'environnement.

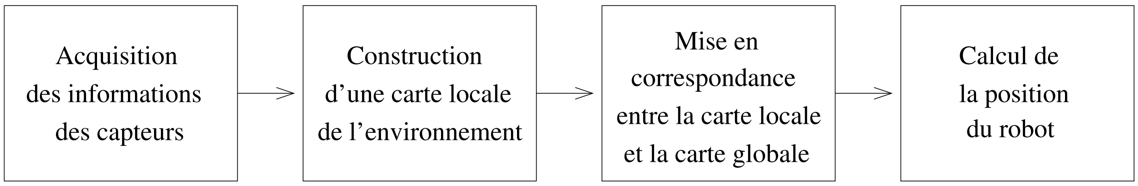


FIG. 8.5 – *Localisation par rapport à une carte*

Le gros avantage de cette technique est qu'à chaque instant le robot mobile connaît précisément sa position et son orientation. Mais cette méthode n'est utilisable que dans un environnement contenant suffisamment de caractéristiques facilement identifiables pour pouvoir être utilisées pour la mise en correspondance. De plus, la plupart des travaux actuels [21] utilisant cette technique sont limités aux environnements intérieurs relativement simples et très statiques.

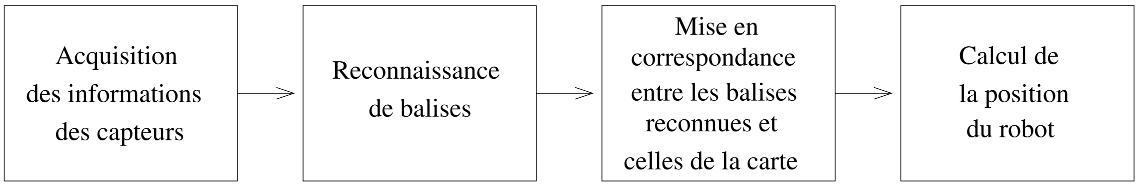


FIG. 8.6 – *Localisation par rapport à des balises naturelles*

20. Par la suite, nous utiliserons indifféremment le terme recalage ou localisation. La différence tient à la prise en compte des informations odométriques ou non. En effet, si on considère que ces informations permettent d'avoir une information grossière sur la position et l'orientation du robot, le processus de localisation se bornera à calculer la position et l'orientation précise du robot, pour éviter la dérive odométrique au cours du temps. On parlera dans ce cas de recalage ou plus précisement de recalage odométrique. Dans le cas contraire, on parlera de localisation.

21. Le lecteur intéressé pourra se reporter à [Leonard et Durrant-Whyte, 1992]

- La deuxième méthode (figure 8.6) est appelée “localisation par rapport à des balises naturelles”. Des balises naturelles telles que les intersections ou les portes ouvertes ont une position fixe et connue dans l'environnement, par rapport à laquelle le robot peut se localiser. Le robot utilise donc ses capteurs pour reconnaître des balises dans son environnement local, et calcule ensuite sa position par rapport aux balises pour se localiser.

  Contrairement à la précédente, cette technique réalise seulement un recalage périodique, mais comme elle est basée sur la reconnaissance de balises, elle est plus robuste aux changements de l'environnement, et peut donc être utilisée dans des environnements plus dynamiques ou dans des environnements extérieurs. Dans cette section, nous expliquons comment les diverses techniques d'apprentissage et de reconnaissance de balises naturelles que nous avons présentées dans la section 8.2 sont utilisées pour localiser un robot mobile.

Etant donné que [Kortenkamp et Weymouth, 1994] travaillent avec un environnement constitué de couloirs, d'intersections de couloirs et de portes ouvertes, la localisation se borne dans leur cas à savoir quelle est la position du robot dans le couloir, c'est à dire à quelle distance est le robot des deux extrémités du couloir. Comme de plus, ils utilisent un “suivi de murs” pour se déplacer, la position et l'orientation par rapport aux deux murs constituant le couloir sont connues à chaque instant. Donc ils recalent simplement le robot par rapport aux extrémités du couloir à chaque fois que le robot a reconnu une balise, car la position des balises dans le couloir est connue.

En utilisant des grilles d'évidence, [Yamauchi, 1996a] peut calculer, lorsqu'il passe par une balise apprise, la dérive odométrique en position et en orientation. Pour cela, Yamauchi décale et/ou tourne la grille d'évidence apprise pour savoir quel décalage et quelle rotation donnent la meilleure mise en correspondance entre la grille apprise et la grille reconnue (figure 8.7). Ceci lui permet de calculer la dérive odométrique et ainsi de recaler son robot en position et orientation.

Dans leur article, [Rompais et Pradet, 1995] ne traitent pas le problème de localisation.

8.4.3 Le contrôle d'exécution d'actions

Lorsqu'un robot mobile se déplace dans un environnement dynamique, il planifie tout d'abord son déplacement puis l'exécute. Les balises qu'il reconnaît au cours de ses déplacements lui permettent de suivre le bon déroulement de son plan. Pendant l'exécution de son plan, il est confronté à diverses incertitudes :

- Une incertitude de contrôle du robot : l'exécution précise d'une action ne peut être garantie, du fait de la structure du sol mais aussi de la présence d'obstacles, qu'ils soient mobiles ou non.
- Une incertitude sur les mesures des capteurs, du fait des réflexions.
- Une incertitude sur la position du robot (liée aux deux précédentes).

Ces incertitudes peuvent être gérées en utilisant un Processus de Décision Markovien Partiellement Observable [Bellman, 1957], qui permet à un robot mobile de choisir la meilleure action en fonction de la connaissance partielle qu'il a de son environnement et

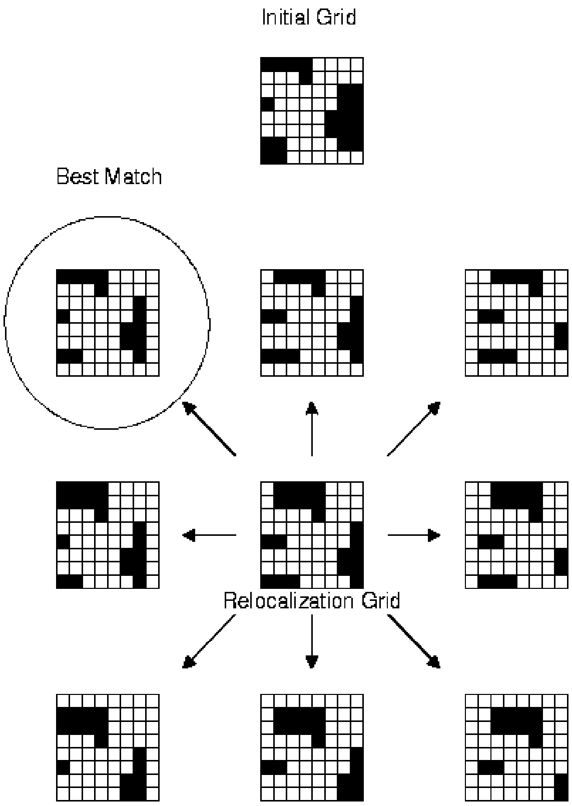


FIG. 8.7 – *Mise en correspondance de grilles d'évidence pour la relocalisation*

de sa situation. L'utilisation d'un POMDP se fait en deux étapes :

- Le calcul d'un plan à suivre en tenant compte des incertitudes et du but à atteindre.
- L'exécution de ce plan qui à chaque instant permet au robot de choisir la meilleure action en fonction de la connaissance qu'il a de son environnement et du but à atteindre.

Dans le paragraphe suivant, nous définissons les différents constituants d'un POMDP dans le cadre du contrôle d'exécution d'actions en robotique. Puis, nous décrivons les divers types d'algorithmes de calcul du plan en citant un exemple. Nous présentons ensuite l'algorithme d'exécution du plan. Cette étude est reprise de [Laroche, 1996].

Pour terminer, nous illustrons ce paragraphe par deux exemples d'application des POMDPs à la planification et à l'exécution d'actions en robotique mobile.

Définition

De manière formelle, un Processus de Décision Markovien Partiellement Observable est défini par un sextuplet $< \mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R},\mathcal{O},\mathcal{O}' >$:

- $\mathcal{S}$: un ensemble fini d'états de l'environnement *identifiables* par le robot. Un état est généralement une portion de l'environnement (habituellement un carré d'un mètre sur un mètre) associée à une orientation du robot. A l'intérieur d'un même état, quatre orientations (ou plus) sont possibles (nord, est, sud, ouest), une portion de l'environnement correspondra donc à quatre états différents.
- $\mathcal{A}$: un ensemble fini d'actions. Les actions peuvent être modélisées à plus ou moins haut niveau. Cela peut aller des actions haut-niveau telles que "Eviter obstacle" ou "Traverser couloir" jusqu'à des actions plus simples et plus faciles à mettre en œuvre. Généralement, trois actions de ce type sont utilisées : "Avancer", "Pivoter vers la gauche selon un angle de 90°" et "Pivoter vers la droite selon un angle de 90°".
- $\mathcal{T}$: une fonction de transition entre états selon l'action effectuée.

  $$\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$$

  $\mathcal{T}(s,a,s')$: probabilité de passer de l'état s à l'état s' en effectuant l'action a. La fonction de transition établit les probabilités de passage entre états en fonction de l'action effectuée. Cette formalisation appelle plusieurs remarques quant au cadre d'utilisation de cette méthode :

  - les états sont censés être identifiables de façon sûre par le robot, ce qui présuppose que chaque état peut être différencié d'une façon ou d'une autre des autres états.
  - la fonction de transition, si elle permet de gérer les incertitudes de contrôle du robot, ne prend pas en compte la dynamique de l'environnement (par exemple, si le passage entre un état et un autre est obstrué par un obstacle imprévu, la fonction de transition ne sera plus représentative de la réalité).

  D'autre part, dans la plupart des travaux sur les POMDPs en robotique, la construction de cette fonction de transition est considéré comme donnée, ou est construite de manière empirique.

- $\mathcal{R}$: fonction de gain
$$\mathcal{R} : \mathcal{S} \rightarrow \Re$$
$\mathcal{R}(s)$: gain obtenu du fait de se trouver dans l'état s. L'objet de la fonction de gain est de désigner le but à atteindre en affectant une récompense au robot en fonction de l'état dans lequel il se trouve. *A priori*, plus l'état dans lequel se trouve le robot est proche du but, plus la récompense doit être forte. Cela nous donne une fonction de gain qui peut être calculée en fonction de la distance entre l'état considéré et le but. Cette fonction de gain devra donc être redéfinie à chaque fois que le but change.
- $\mathcal{O}$: ensemble fini d'observations de l'environnement *identifiables* par le robot. Ces observations peuvent être de simple mesures des capteurs ou des balises naturelles symboliques.
- $\mathcal{O}'$: une fonction d'observation qui peut être définie ainsi :
$$\mathcal{O}' : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{O})$$
$\mathcal{O}'(s,a,o)$: probabilité d'observer o de l'état s après avoir exécuté l'action a.
Cette fonction d'observation permet de prendre en compte l'environnement dans lequel évolue le robot et lui est utile pour se localiser en fonction des observations qu'il fait. C'est en fait ici que la reconnaissance de balise joue un rôle important dans ce genre d'approche, pour deux raisons : premièrement elle permet d'utiliser des observations symboliques et deuxièmement elle permet de construire la fonction d'observation. Nous montrerons lors de l'application de notre approche comment nous avons construit cette fonction de transition.
De même que pour la fonction de transition, dans la plupart des travaux sur les POMDP la construction de cette fonction de transition est considérée comme donnée, ou est construite de manière empirique.

L'algorithme de recherche du plan

Une fois l'environnement modélisé et le but fixé, celui-ci pourra être atteint grâce à l'exécution d'un plan π, qui associe à chaque état une action optimale ($\pi : \mathcal{S} \rightarrow \mathcal{A}$). Le plan retenu est celui qui maximise la valeur de chaque état, valeur définie comme suit :

$$V_\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_t$$

où $\mathcal{R}_t$ est le gain reçu à l'étape t de l'exécution du plan π après avoir commencé à l'état s et γ est un coefficient pondérateur ($0 \leq \gamma < 1$) permettant de faire converger la somme et d'accorder plus ou moins d'importance aux gains futurs.

Deux types d'algorithmes sont généralement utilisés pour calculer le plan. Le premier type ne prend pas en compte les observations lors du calcul, et ne les utilise que lors de l'exécution, c'est par exemple le cas de l'algorithme Policy Iteration [Howard, 1960]. La figure 8.8 donne un exemple de plan généré par l'algorithme Policy Iteration. Le deuxième type prend en compte les observations lors du calcul et lors de l'exécution, c'est par exemple le cas de l'algorithme Witness [Littman *et al.*, 1995].

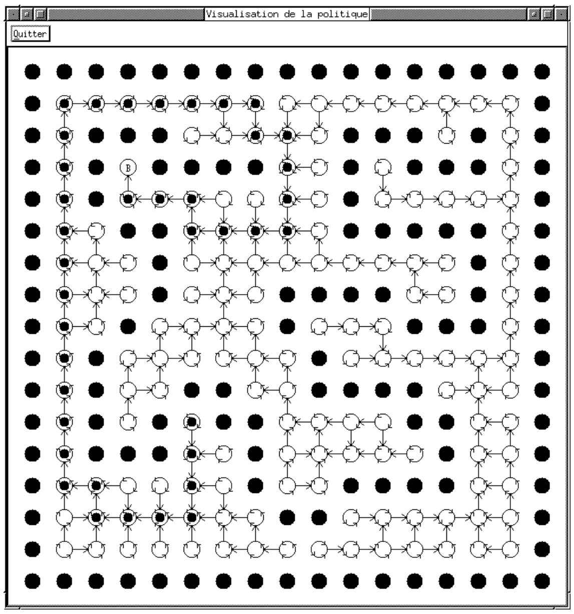


FIG. 8.8 – *Plan généré avec l'algorithme Policy Iteration*

L'exécution du plan

Le but de l'exécution du plan est de trouver à chaque instant la meilleure action à exécuter en fonction du but à atteindre et de l'état dans lequel se trouve le robot. Pour cela, à chaque fois que le robot se déplace et reconnaît une balise, il faut définir les états où le robot peut se trouver et la probabilité que le robot a d'être dans ses états.

Afin de stocker ces probabilités, on introduit un "belief state", liste à $|\mathcal{S}|$ éléments où $b(s)$ note la probabilité que le robot soit dans l'état s :

$$b : \mathcal{S} \rightarrow \Pi(\mathcal{S})$$

Ce belief state permettra de connaître les états possibles à l'instant t ($b(s) \neq 0$) et les états dans lesquels il est impossible que le robot se trouve ($b(s) = 0$). Celui-ci sera modifié après chaque action en fonction du belief state précédent, de la dernière action et de la dernière observation faite. Ceci est fait en utilisant un estimateur d'état (SE) qui calcule les composantes s' du nouveau belief state de la façon suivante :

$$\begin{aligned} SE_{s'}(b,a,o) &= P(s'/a,o,b) \\ &= \frac{P(o/s',a,b)P(s'/a,b)}{P(o/a,b)} \\ &= \frac{O(s',a,o)\sum_{s\in\mathcal{S}} b(s)\mathcal{T}(s,a,s')}{P(o/a,b)} \end{aligned}$$

où $P(o/a,b)$ est un facteur normalisateur défini ainsi :

$$P(o/a,b) = \sum_{s'\in\mathcal{S}} O(s',a,o) \sum_{s\in\mathcal{S}} b(s)\mathcal{T}(s,a,s')$$

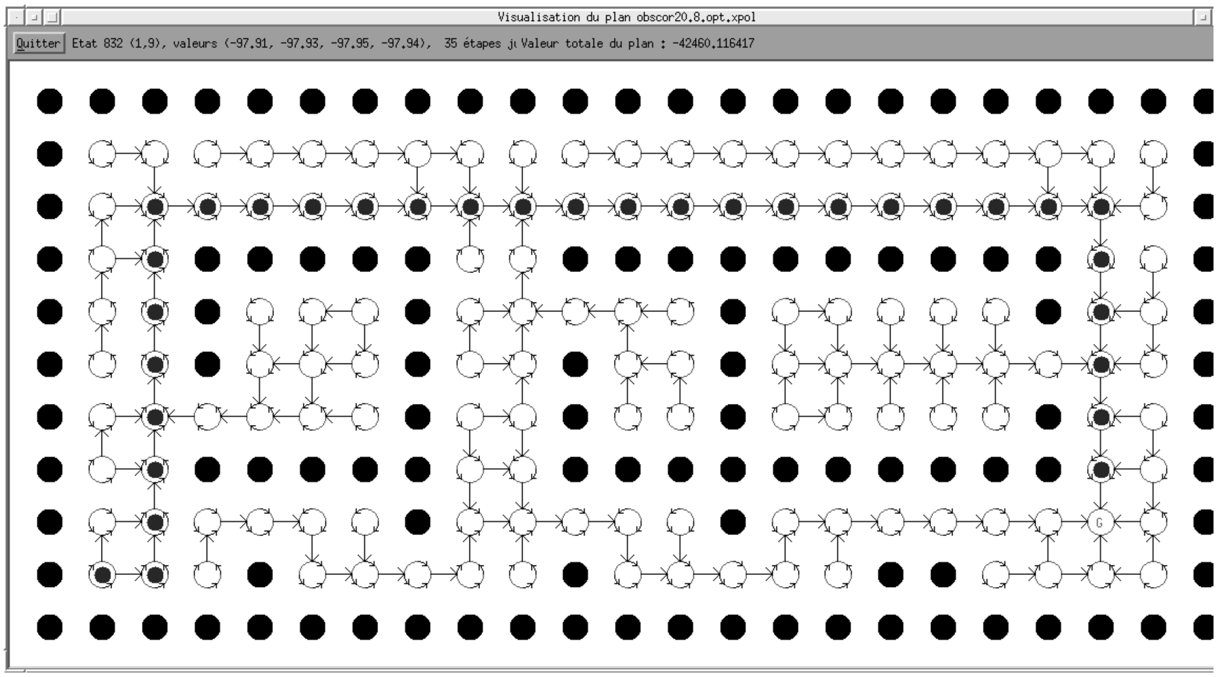


FIG. 8.9 – *L'exécution du plan*

Chaque fois que l'ensemble des "beliefs states" est réestimé, nous utilisons une stratégie pour définir l'action à exécuter. La stratégie la plus simple revient à exécuter l'action correspondante à l'état dont le belief state est le plus probable.

Exemples

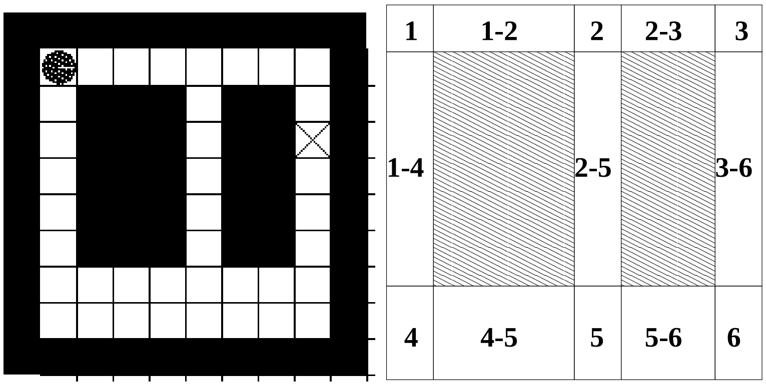


FIG. 8.10 – *Un environnement 8*8 ramené à 13 états*

Les POMDPs permettent d'appréhender efficacement les incertitudes de l'environnement, mais la complexité des algorithmes de planification ne permet pas de les utiliser dans une application réelle de robotique mobile. A partir de ce constat, une approche originale est développée dans [Nourbakhsh *et al.*, 1993] et [Simmons et Koenig, 1995] : l'environnement est représenté par un POMDP, mais le plan est calculé à l'aide d'un graphe. Ce graphe est obtenu à partir d'une représentation abstraite de l'environnement, chaque nœud englobant plusieurs états du POMDP, comme le montre la figure 8.10.

Cette représentation, si elle a l'avantage de réduire considérablement l'espace d'états, permet également l'utilisation d'un algorithme A^* pour calculer un plan reliant l'état initial à l'état but. Ce plan, calculé en ne tenant pas compte de l'incertitude de transition entre états, affecte une action à chaque état du chemin trouvé. Pendant l'exécution du plan, un belief state permet de gérer l'incertitude de position.

Les deux approches différent en ce qui concerne l'exécution du plan. La solution de Nourbakhsh exécute l'action affectée à l'état le plus probable. S'il s'avère que cet état n'est pas sur le chemin calculé, un nouveau plan est construit. Pour Koenig, c'est l'action qui a la plus forte probabilité d'être optimale vis-à-vis des états probables qui est exécutée. Les états adjacents à ceux situés sur le chemin optimal se voient affecter une action qui les ramène sur ce chemin, ce qui permet de ne pas calculer un nouveau plan quand une mauvaise exécution d'action a fait sortir le robot du chemin optimal.

Ces deux approches sont intéressantes puisqu'elles permettent de contrôler l'exécution d'un déplacement, et de définir à chaque instant la meilleure action à réaliser par le robot. Le point le plus délicat de ces deux approches (comme la plupart des applications des POMDPs) est la construction des fonctions de transitions et d'observations. Nous nous intéressons ici particulièrement à la fonction d'observations, puisqu'elle constitue une utilisation possible de l'apprentissage et de la reconnaissance de balises.

Dans les deux approches présentées, la construction de la fonction d'observation est réalisée de la même manière : on détermine dans chaque état et pour chaque transition issue de cet état dans quel état le robot va se rendre, et à partir de là, la balise qu'il va voir. Pour construire la fonction d'observation, on ajoute à chaque balise reconnue une probabilité de confiance. Par exemple, admettons que le robot se déplace entre deux états. En analysant le déplacement entre ces deux états, nous savons quelle balise le robot doit voir pendant son déplacement. Admettons maintenant que quand le robot doit voir cette balise (càd, quand il se déplace entre ces deux états), il la reconnaisse dans 70% des cas, il la confonde avec une autre balise dans 20% des cas et ne la reconnaisse pas dans 10% des cas. Ces probabilités déterminent en fonction de la balise reconnue la probabilité d'observation associée entre les deux états correspondants.

Pour construire sa fonction d'observation, [Nourbakhsh *et al.*, 1993] utilisent des règles similaires à [Kortenkamp et Weymouth, 1994] pour reconnaître les balises de l'environnement. Il ajoute une confiance à chaque balise reconnue, mais cette confiance est calculée empiriquement, et ne repose sur aucun fondement théorique. [Simmons et Koenig, 1995] représente des balises naturelles telles que des portes ouvertes, des intersections ou des murs par des grilles d'évidence, et utilise un algorithme de reconnaissance de balises du même type que [Yamauchi, 1996a]. Il peut à partir de cette reconnaissance, déduire une confiance accordée à chaque type de balise reconnue, pour construire sa fonction d'observation. Comme [Nourbakhsh *et al.*, 1993] ne reposent sur aucune méthode statistique et semble être calculée à partir de constatations complètement empiriques.

Conclusion

L'utilisation des POMDPs en robotique mobile est assez récente, et est un domaine de recherche actuellement très actif. Les POMDPs sont idéaux pour réaliser un contrôleur d'exécution d'actions, puisqu'à chaque instant ils permettent de connaître les différentes positions possibles d'un robot mobile dans son environnement, ainsi que leur vraisemblance. D'autre part, ils permettent de déterminer pour chaque position identifiable de l'environnement (càd, chaque état du modèle) la meilleure action à exécuter pour se rapprocher du but. L'apprentissage et la reconnaissance de balises interviennent dans la construction de la fonction d'observation qui est utilisée lors de l'exécution d'un plan pour mettre à jour les beliefs states. Malheureusement, dans la totalité des travaux actuels, cette fonction est construite empiriquement comme nous l'avons vu dans les deux exemples cités.

8.4.4 La construction de cartes topologiques

Comme nous l'avons dit précédemment, l'apprentissage et la reconnaissance de balises est indispensable pour la construction de cartes topologiques. Nous présentons dans cette section le système ELDEN (Exploration and Learning in Dynamic ENvironnements) (figure 8.11) de [Yamauchi, 1995]. Le système RPLAN de Kortenkamp n'est pas présenté. Le lecteur intéressé peut se reporter à sa thèse [Kortenkamp, 1993].

Le système ELDEN

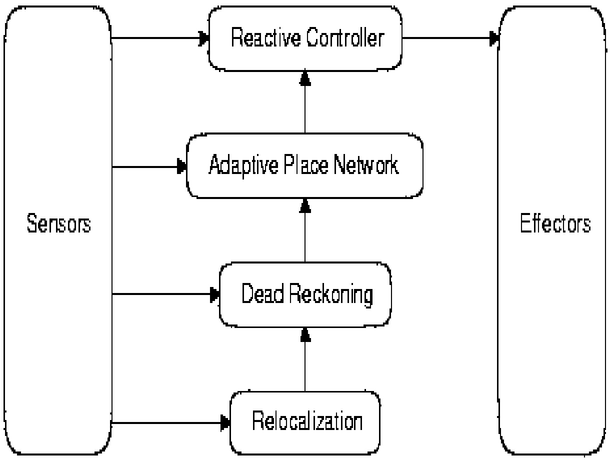


FIG. 8.11 – *Le système ELDEN*

ELDEN (Exploration and Learning in Dynamic ENvironnements) (figure 8.11) [Yamauchi, 1995] est un système autonome d'apprentissage spatial conçu pour permettre à des robots mobiles d'explorer un environnement dynamique et d'y naviguer. Il opère selon deux modes :

– Un mode d'exploration, dans lequel le but du système est d'apprendre la structure spatiale de l'environnement.

– Un mode de navigation, dans lequel le but du système est de déplacer le robot à une position particulière.

ELDEN a trois composantes principales, qui sont conçues pour faire face à un type particulier de changement de l'environnement :

– Un système de contrôle de bas niveau composé de comportements réactifs. Il s'occupe des changements passagers de l'environnement, tels que les personnes marchant devant le robot, et garantit que le robot sera capable d'opérer même si sa représentation spatiale ne reflète pas exactement l'état courant de l'environnement.
– Un réseau adaptatif de balises qui apprend la topologie et la géométrie de l'environnement. Il s'occupe des changements topologiques et s'adapte constamment pour refléter chaque nouvelle balise de l'environnement qui est rencontrée. Il fournit une représentation spatiale et un système d'apprentissage pour ELDEN qui contient à la fois des informations métriques et topologiques sur la structure de l'environnement. Le réseau adaptatif de balises est composé de balises et des liens qui les relient. Un lien est constitué de quatre attributs (i, j, θ, c). i et j représentent le numéro des deux balises reliées par le lien. θ représente l'angle approximatif pour aller de la balise i à la balise j. c représente le niveau de confiance associé à ce lien.
  L'apprentissage de l'environnement est toujours actif que le système soit en mode d'exploration ou de navigation.
  Au début d'une mission, lorsque le robot ne connaît pas du tout son environnement le réseau de balises adaptatif est vide. A chaque instant, la distance entre la position courante du robot et la balise connue la plus proche est calculée. Si cette distance est inférieure à un seuil fixé (75 centimètres), alors le robot est considéré comme étant devant cette balise, sinon une nouvelle balise est créée. Un lien est créé entre deux balises lorsque le robot passe de l'une à l'autre. θ est initialisé avec l'orientation courante du robot, et c à 0,5. Au fur et à mesure que le robot passe par ce lien, θ est recalculé en fonction de l'orientation courante du robot et de la valeur précédente de θ. Le niveau de confiance associé à ce lien est augmenté.
– Un système de relocalisation qui utilise des grilles d'évidence pour recalibrer le système odométrique du robot. Il s'occupe d'une forme spécifique de changement perceptuel, le glissement des roues sur le sol qui provoque des erreurs d'estimation de position et d'orientation du robot.

8.5 Conclusion

Dans ce chapitre, nous avons vu que la reconnaissance de balises naturelles sert dans de nombreuses tâches en robotique mobile telles que la localisation, le contrôle d'exécution d'actions ou la construction de cartes cognitives. De plus, nous avons présenté, en nous appuyant sur des exemples, les deux types de méthodes qui permettent d'apprendre et de reconnaître des balises. La première consiste à expliquer les observations ou à construire une représentation de ces observations, alors que la seconde consiste à construire un modèle représentant les propriétés statistiques de ces observations.

Au delà de cette classification, on remarque que les approches présentées prennent en compte trois autres critères pour l'apprentissage et la reconnaissance de balises :

- L'apprentissage et la reconnaissance de balises peut être vu soit en considérant les balises de manière locale (c-à-d, une balise est apprise et reconnue en utilisant une observation), soit en considérant les balises de manière globale (c-à-d, une balise est apprise et reconnue en utilisant une suite d'observations).
- On note aussi que certaines approches présentées n'utilisent qu'un capteur alors que d'autres utilisent un ensemble (voire la totalité) des capteurs.
- Nous avons vu aussi que, selon la méthode adoptée, la reconnaissance pouvait être liée à une interprétation des données qui permet de savoir quel type de balise vient d'être reconnue.

Ces trois critères cités ci-dessus permettent de distinguer les approches que nous avons présentées pour apprendre et reconnaître des balises naturelles. D'après l'analyse de ces trois critères, nous retenons qu'une bonne approche pour apprendre et reconnaître des balises doit :

- Utiliser les mesures de plusieurs capteurs pour constituer l'observation.
- Utiliser une séquence d'observations composée d'un nombre variable d'observations dépendant des critères utilisés pour apprendre et reconnaître la balise.
- Donner une interprétation symbolique de chaque balise reconnue.

Pour [Kortenkamp et Weymouth, 1994] une balise est caractérisée par une suite d'événements temporels reconnues grâce à une suite de règles. Cette approche nous semble très intéressante pour diverses raisons :

- La reconnaissance de balises est réalisée en plusieurs étapes en considérant l'ensemble des observations du capteur perpendiculaire au mur. D'autre part, la définition des règles est simple à réaliser, puisqu'elle est très intuitive.
- Une fois la reconnaissance terminée, on sait quel type de balise a été reconnue.

Par contre, un certain nombre de points posent problèmes :

- La détection de chaque événement est réalisée par une comparaison entre la mesure du capteur perpendiculaire au mur et un seuil prédéfini. Ce seuil déterminé expérimentalement ne permet une détection des événements temporels que dans des situations prédéfinies et très restreintes. Il paraît évident que dans un environnement plus dynamique que celui qu'utilisent [Kortenkamp et Weymouth, 1994] le robot ne peut pas se contenter de longer les murs du couloir, et à ce moment-là la détermination du seuil devient difficile.
- Le nombre de type de balises détectables reste très restreint, car le nombre d'événements détectables est faible. De plus, la seule prise en compte des événements caractérisant une balise n'est pas toujours suffisante.
- D'autre part, seules les mesures du capteur perpendiculaire aux murs sont prises en compte. Il y a donc une perte considérable d'informations. Ces informations pourraient justement servir à détecter d'autres événements sur d'autres capteurs, et permettre de reconnaître un nombre plus important de balises.
- Etant donné, que lorsqu'une balise est reconnue, aucune information sur la position et l'orientation du robot par rapport à cette balise n'est donnée, la localisation du robot ne peut être que grossière.

Cette approche pourrait être considérablement améliorée :

– en laissant le robot apprendre les événements temporels qui caractérisent une balise, et en utilisant d'autres, tels que la longueur d'une balise par exemple.
– en utilisant les informations de plusieurs capteurs pour avoir plus d'informations sur l'environnement du robot.

Dans les chapitre suivants, nous présentons notre approche basée sur les modèles de Markov cachés qui permettent de résoudre les différents problèmes de l'approche de [Kortenkamp et Weymouth, 1994]. Ces modèles ont justement la capacité d'apprendre, à partir d'une séquence d'observations provenant d'un ou de plusieurs capteurs, à caractériser un phénomène à apprendre sous la forme d'une suite d'événements temporels. Notre approche se situe à mi-chemin entre les deux types d'approches présentées puisqu'elle apprend et reconnaît une balise sous forme d'une suite d'événements temporels, qui ne sont pas détectés par des règles rigides mais qui sont appris automatiquement.

9

Les modèles de Markov cachés

9.1 Introduction

Dans ce chapitre, nous présentons une classe particulière de modèles statistiques que nous avons utilisés pour réaliser l'apprentissage et la reconnaissance de balises naturelles : les modèles de Markov Cachés ou MMC (en anglais Hidden Markov Models ou HMM). Ils ont été introduits dans les années soixante dix par des chercheurs américains d'IBM[Jelinek, 1976]. Une des principales propriétés des modèles de Markov cachés est leur capacité à construire des modèles stochastiques sur des suites temporelles d'événements sans rapport les uns avec les autres. La modélisation d'un signal temporel par des modèles de Markov cachés est fondée sur deux principes : (i) les MMC supposent que le signal temporel peut être découpé en segments par une chaîne de Markov et (ii) le signal est la réalisation d'un processus stationnaire représenté par une densité de probabilité sur l'espace des observations à l'intérieur d'un segment. Nous avons vu dans le chapitre précédent que le signal temporel représentant une balise peut être découpé en trois parties grâce à de simples règles sur la variation des capteurs. conditions qui permettent l'utilisation de MMC sont vérifiées, Ils se prêtent donc tout à fait bien à l'apprentissage et à la reconnaissance des événements temporels indépendants caractérisant une balise naturelle. Sur un plan pratique, leur utilisation est justifiée : plutôt que de fixer des règles rigides permettant de reconnaître la suite d'événements temporels caractérisant une balise, il peut être intéressant d'essayer d'apprendre de manière automatique la suite d'événements temporels caractérisant une balise. C'est exactement ce que vont faire les MMCs.

Dans la section suivante, nous présentons les modèles de Markov cachés[22]. Nous commençons par quelques rappels sur les chaînes de Markov, puis définissons ce qu'est un modèle de Markov caché ainsi que les différents types de modèle et les applications dans lesquels ils sont utilisés. Pour terminer nous analysons différentes classes de modèles ainsi que les problèmes auxquels ils peuvent s'appliquer. Cette section est illustrée par la définition d'un modèle représentant un couloir et une intersection de l'environnement.

Dans la section 9.3, nous nous intéressons à l'apprentissage et à la reconnaissance des modèles d'ordre un. Nous expliquons tout d'abord ce qu'est un alignement temporel et

22. Pour une étude plus approfondie de la théorie des modèles de Markov, et des algorithmes présentés dans ce chapitre, le lecteur pourra se reporter à [Rabiner, 1989].

comment se calcule sa vraisemblance. Ensuite, nous présentons l'algorithme de Viterbi utilisé pour la reconnaissance qui permet à partir d'une séquence d'observations de déterminer quel modèle a la plus grande probabilité de générer cette séquence d'observations. Nous donnons ensuite les deux critères les plus souvent utilisés pour l'apprentissage des modèles, puis nous terminons par l'algorithme de Baum-Welch utilisé pour l'apprentissage qui à partir d'un corpus permet de construire un modèle apte à reconnaître les séquences d'observations de ce corpus. Cette section est illustrée par l'apprentissage d'un modèle de couloir et d'un modèle d'intersection à partir de deux corpus simulés (l'un représentant un ensemble de séquences d'observations lorsque le robot traverse un couloir et l'autre lorsqu'il passe devant une intersection).

Dans la section 9.4, nous nous intéressons à l'extension des algorithmes d'apprentissage et de reconnaissance à l'ordre deux. En effet, notre approche a été réalisée à l'aide d'un outil développé dans l'équipe par Jean-François Mari et Dominique Fohr. Cet outil basé sur des MMCs d'ordre deux permet l'apprentissage d'un modèle à partir d'un corpus d'apprentissage, et la reconnaissance du modèle correspondant à une séquence d'observations.

9.2 Les Modèles de Markov Cachés

9.2.1 Introduction

Notre but dans cette section est de définir les modèles de Markov cachés. Nous faisons quelques rappels sur les chaînes de Markov. Pour cela, nous donnons la définition des processus et des suites stochastiques. Ensuite, nous présentons les composantes d'un MMC, ainsi que les différents types de modèles : modèle d'ordre un ou modèle d'ordre deux, modèle utilisant des observations discrètes ou continues. Enfin, nous analysons quelques topologies de modèles et les applications auxquelles elles se destinent. Cette section est illustrée, dans sa dernière partie, par la définition d'un MMC d'ordre un avec des observations discrètes qui permet de modéliser un couloir et une intersection.

9.2.2 Les chaînes de Markov

Processus stochastique

On appelle processus stochastique toute famille de variables aléatoires $X(t)$ où t est un paramètre réel. $t \in T$ où T représente l'espace des paramètres. En général, T est dénombrable et représente le temps, le processus est alors dit *discret*.

q_t indicée par t, on appelle processus stochastique l'ensemble :

Suite stochastique

Soit un système pouvant se trouver dans un ensemble dénombrable d'états. On pose : $\mathbf{S} = \{s_1, s_2, \ldots, s_N\}$ Entre les instants t et $t+1$, le système passe aléatoirement de l'état s_i à l'état s_j. On appelle Ω l'ensemble des transitions notées (i,j) entre deux états. Pour représenter ce processus aléatoire, on définit la variable q_t de la façon suivante : $q_t = s_i$ signifie que le système est dans l'état s_i au temps t. Par définition l'ensemble Q $(q_1,q_2,\ldots q_t,\ldots)$ est une suite stochastique à ensemble discret d'états.

Dans la suite de ce mémoire on ne s'intéresse qu'aux suites stochastiques dont l'espace d'état S et l'espace des paramètres T sont finis et discrets.

Chaîne de Markov

Une chaîne de Markov est une suite stochastique à ensemble discret d'états telle que :

- Suite du premier ordre

$$\begin{aligned} &Prob(q_t = s_j/q_{t-1} = s_i,q_{t-2} = \ldots) \\ &= Prob(q_t = s_j/q_{t-1} = s_i) \end{aligned} \tag{9.1}$$

- Suite du deuxième ordre

$$\begin{aligned} &Prob(q_t = s_k/q_{t-1} = s_j,q_{t-2} = s_i,q_{t-3} = \ldots) \\ &= Prob(q_t = s_k/q_{t-1} = s_j,q_{t-2} = s_i) \end{aligned} \tag{9.2}$$

En d'autres termes, l'évolution du système entre deux instants t et $t+1$ ne dépend que de l'état de ce système au temps $t-1$ (ordre un) ou aux instants précédents $t-1$ et $t-2$ (ordre deux).

Une chaîne est dite homogène (ou stationnaire) si les probabilités de transition ne dépendent pas du temps. On peut alors poser : $a_{ij} = Prob(q_t = s_j/q_{t-1} = s_i)$ et $a_{ijk} = Prob(q_t = s_k/q_{t-1} = s_j,q_{t-2} = s_i)$.

Si l'ensemble des états **S** est fini et possède N états alors on dit que la chaîne est finie. Dans ce cas, on a les contraintes suivantes :

- Chaînes du premier ordre

$$\sum_{j=1}^{N} a_{ij} = 1 \quad , 1 \leq i \leq N$$

- Chaînes du second ordre

$$\sum_{k=1}^{N} a_{ijk} = 1 \quad , 1 \leq i,j \leq N$$

Pour définir complètement, une chaîne de Markov, on ajoute un vecteur de probabilités initiales $\pi = \{\pi_i\}$ où $\pi_i = Prob(q_1 = s_i)$ *avec* $1 \leq i \leq N$ qui détermine la probabilité de l'état s_i à t $= 1$.

9.2.3 Les modèles de Markov

- Un ensemble S de N états : $\{s_1,s_2,\ldots,s_N\}$.

- Une matrice de probabilités de transitions A.
  - Modèle d'ordre un, A $= \{a_{ij}\}$ à deux dimensions sur S x S.

$$a_{ij} = Prob(q_t = s_j, q_{t-1} = s_i)$$

$$= Prob(q_t = s_j / q_{t-1} = s_i, q_{t-2} = ...)$$

    avec la contrainte :

$$\sum_{j=1}^{N} a_{ij} = 1 \quad avec\ 1 \leq i \leq N$$

  - Dans le cas d'un modèle d'ordre deux, A $= \{a_{ijk}\}$ à trois dimensions sur S x S x S.

$$a_{ijk} = Prob(q_t = s_k / q_{t-1} = s_j, q_{t-2} = s_i)$$

$$= Prob(q_t = s_k / q_{t-1} = s_j, q_{t-2} = i, q_{t-3} = \ldots)$$

    avec la contrainte :

$$\sum_{k=1}^{N} a_{ijk} = 1 \quad avec\ 1 \leq i,j \leq N$$

  où q_t est l'état actuel au temps t.
- Un vecteur de probabilités initiales $\pi = \{\pi_i\}$ où $\pi_i = Prob(q_1 = s_i)\ avec\ 1 \leq i \leq N$

$Q_t = (q_1, q_2, \ldots, q_t)$ est une chaîne de Markov finie et homogène d'ordre un ou deux, définie par $\{N,S,A,\pi\}$, qui constitue la *partie cachée* du modèle. La *partie visible* du modèle est un processus stochastique $O_t = (o_1, o_2, \ldots, o_t)$, où o_t est l'observation actuelle au temps t.

Les observations associées à chaque état sont soit discrètes, soit continues . Dans le cas discret, nous avons :

- Un ensemble de M observations appelé V=$\{v_1, v_2, ..., v_M\}$ qui définit l'ensemble des valeurs possibles de O_t.
- Une matrice de probabilités d'observations B $= b_i(o_t)$ à deux dimensions sur S x V.

$$b_i(o_t) = Prob(o_t = v_k / q_t = s_i) \quad avec\ 1 \leq i \leq N\ ,\ 1 \leq k \leq M \tag{9.3}$$

  où o_t est l'observation au temps t. avec la contrainte :

$$\sum_{k=1}^{M} b_i(k) = 1 \quad avec\ 1 \leq i,j \leq N$$

Et dans le cas continu :

- O_t est un vecteur de dimension d qui prend ses valeurs dans R^d. La probabilité d'observations du vecteur au temps t associé à l'état s_i est une somme pondérée des probabilités d'observations des composantes du vecteur. Chaque composante

22. Par la suite, nous ne nous intéressons qu'aux observations discrètes. Le cas des observations est traité en détail dans [Rabiner, 1989].

du vecteur d'entrée O_t est modélisée par une loi normale $\mathcal{N}(\mu_{id},\sigma_{id})$. La densité de chaque O_t est définie par la donnée des densités $b_i() sur S$ x R^d représentant chacun des états s_i du processus de Markov caché. On modélise $b_i()$ par une somme pondérée de d lois normales $\mathcal{N}(\mu_{id},\sigma_{id})$ de moyenne μ_{id} et de matrice de covariances σ_{id}.

$$b_i(O_t) = \sum_{j=1}^{d} c_{ij}\mathcal{N}(O_t;\mu_{ij},\sigma_{ij}), \quad \text{avec} \quad \sum_{d=1}^{D} c_{id} = 1$$

L'ensemble de ces paramètres $\lambda = \{N,S,A,\pi,V,B\}$ constitue le modèle de Markov caché.

Lors de la définition d'un modèle de Markov caché, le nombre N d'états du modèle, l'ensemble des observations possibles, ainsi que le vecteur de probabilités initiales π sont fixés au départ, et leurs valeurs ne sont plus jamais modifiées. Par contre, la matrice de probabilités de transitions A et la matrice de probabilités d'observations B sont données au départ, mais elles sont modifiées lors de l'apprentissage afin que le modèle corresponde au mieux au type de séquence d'observations qu'il doit reconnaître.

9.2.4 Quelques topologies de modèles

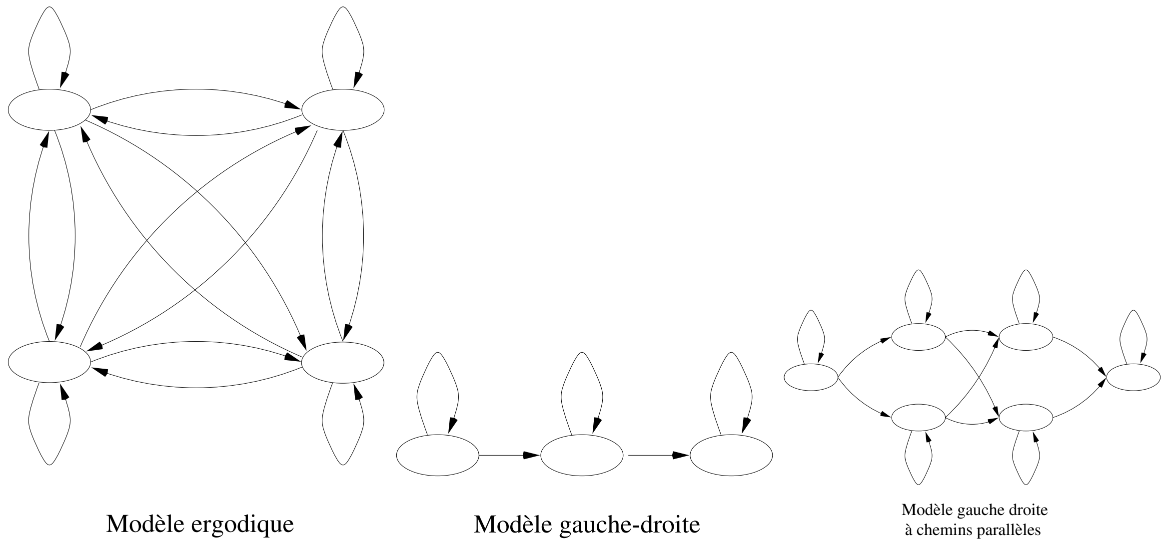


FIG. 9.1 – *Exemple de type de modèle de Markov*

Une possibilité de classer les modèles peut être faite par la structure de la matrice de transition, c'est à dire par la topologie du modèle. Nous pouvons en effet lors de la construction de la matrice de transition supprimer certaines transitions afin que la topologie du modèle s'adaptent à des classes de problèmes donnés en répondant à des contraintes particulières. Nous présentons ici les topologies de modèles les plus connus :

- Le modèle ergodique. Ce modèle a la propriété que chaque état peut être atteint à partir de n'importe quel état en un nombre apériodique d'étapes.

– Le modèle Gauche Droite. Ce modèle a la propriété qu'au fur et mesure que le temps augmente, l'état dans lequel se trouve le processus à modéliser augmente aussi (ou reste le même). Ce type de modèle a la propriété de modéliser des signaux dont les propriétés changent au cours du temps. On démarre de l'état le plus à gauche pour progressivement se déplacer d'un état vers la droite à chaque fois qu'on reconnaît la caractéristique du signal correspondante à cet état. Nous verrons plus tard que c'est le modèle que nous avons retenu pour l'apprentissage et la reconnaissance des balises caractérisant notre environnement.
– Le modèle gauche droite à chemins parallèles. Il possède une certaine flexibilité par rapport à un modèle gauche droite classique, puisqu'il permet de passer par deux chemins pour aller de l'état initial à l'état final, et il permet aussi de passer d'un chemin à l'autre, en cours de reconnaissance.

9.2.5 Exemple

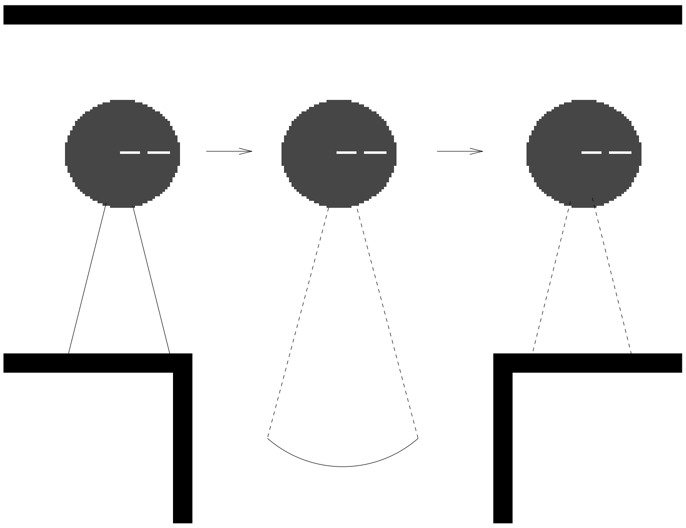


FIG. 9.2 – *Passage du robot devant une intersection*

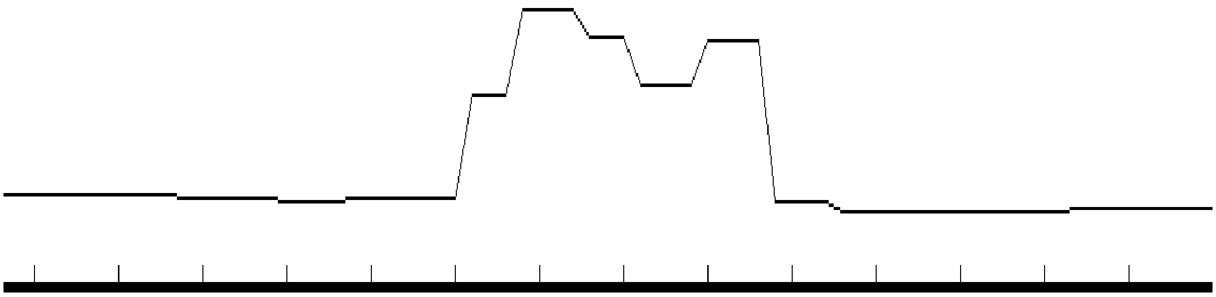


FIG. 9.3 – *Mesures correspondantes du capteur perpendiculaire au mur*

Lorsque notre robot se déplace dans les couloirs du laboratoire (figure 9.2) en utilisant notre module de navigation, on peut comme [Kortenkamp et Weymouth, 1994] caractériser chaque balise naturelle en observant les variations du capteur perpendiculaire au mur. En effet :

- Un couloir est caractérisé par peu ou pas de variation (cf. figure 9.3).
- Une intersection est caractérisée par une brusque et importante augmentation, puis peu ou pas de variation, et pour finir par une importante diminution (cf. figure 9.3).

Une caractérisation simple d'une intersection peut se faire sous la forme de trois événements temporels. La reconnaissance d'une balise est donc faite en trois étapes, et est représentée par un modèle à trois états. Le modèle Gauche Droite de la figure 9.1 est bien adapté au type d'apprentissage et de reconnaissance que nous voulons faire. En effet, en démarrant la reconnaissance dans l'état un, on restera dans cet état tant que le premier événement temporel ne se sera pas produit, et lorsqu'il se sera produit on passera dans l'état deux. En procédant de la même manière pour l'état deux et l'état trois, on sortira de l'état trois, lorsque les trois événements temporels se seront produits, donc que la balise associée au modèle est reconnue.

La matrice A de probabilités de transitions (avant apprentissage) et le vecteur de probabilités initiales π de chaque balise, étant donné le modèle gauche-droite de la figure 9.1, sont :

$$A = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \pi = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Nous admettons qu'avant apprentissage toutes les transitions sont équiprobables.

Pour simplifier le problème et utiliser des observations discrètes qui sont plus faciles à manipuler, nous discrétisons l'espace des mesures des capteurs en cinq classes et de plus considérons qu'une observation est composée uniquement des variations des mesures du capteur perpendiculaire au mur. Le but de cette illustration est de toute façon simplement d'illustrer l'utilisation des modèles de Markov cachés sur un exemple d'apprentissage et de reconnaissance de balises, et pas de montrer comment apprendre et reconnaître des balises dans un cadre réaliste.

Admettons que nous ayons trois types d'observations correspondant aux variations des mesures du capteur perpendiculaire au mur, à savoir :

- Une forte diminution de la mesure.
- Une forte augmentation de la mesure.
- Une variation nulle ou très faible de la mesure.

Pour tenir compte du bruit sur les mesures des capteurs, et des faibles variations dues au module de navigation, nous introduisons deux types d'observations supplémentaires :

- Une faible diminution de la mesure.
- Une faible augmentation de la mesure.

Nous représenterons par la suite ces cinq types d'observations par des entiers allant de -2 à +2 : V=$\{-2, -1{,}0{,}1{,}2\}$.

- -2 correspondant à une forte diminution de la mesure.
- -1 correspondant à une faible diminution de la mesure.

- 0 correspondant à une variation très faible ou nulle de la mesure.
- +1 correspondant à une faible augmentation de la mesure.
- +2 correspondant à une forte augmentation de la mesure.

La matrice de probabilités d'observations (avant apprentissage) B associée est :

$$B = \begin{pmatrix} 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 & 0.2 \end{pmatrix}$$

Ce modèle défini ci-dessus constitue le modèle initial de chaque balise à apprendre et à reconnaître.

9.2.6 Conclusion

Dans cette section, nous avons présenté un modèle de Markov ainsi que ses composantes et les différents types de modèles généralement utilisés. D'autre part, nous avons référencé quelques topologies de modèle, et les applications dans lesquels on les utilise généralement. Ceci nous a permis de définir un modèle d'ordre un avec cinq types d'observations pour représenter un couloir et une intersection. Dans la section suivante, nous présentons l'algorithme de Viterbi qui va permettre de mettre en compétition ces deux modèles afin de savoir étant donné une séquence d'observations de quel modèle elle se rapproche le plus. Puis, nous présentons l'algorithme de Baum-Welch qui à partir d'un corpus d'apprentissage simulé va permettre d'estimer les matrices de transitions et d'observations lui correspondant le mieux.

9.3 L'apprentissage et la reconnaissance pour le premier ordre

9.3.1 Introduction

Cette section est consacrée aux algorithmes d'apprentissage et de reconnaissance des modèles d'ordre un. Nous expliquons tout d'abord ce qu'est un alignement temporel et comment se calcule sa probabilité. Puis nous présentons tout d'abord l'algorithme de Viterbi qui à partir d'une séquence d'observations détermine la séquence d'états (et donc le modèle) le plus probable. Ce paragraphe est illustré par un exemple de reconnaissance d'une séquence d'observations correspondant à une intersection où les modèles de couloir et d'intersection définis précédemment sont mis en compétition. Les deux critères les plus couramment utilisés pour l'apprentissage des modèles sont présentés : le critère du maximum de vraisemblance et le critère d'information mutuelle. Pour terminer, nous décrivons l'algorithme de Baum-Welch utilisé pour l'apprentissage des modèles à partir d'un corpus d'apprentissage de chaque modèle, et illustrons cet algorithme par un exemple d'apprentissage du modèle du couloir et de l'intersection à partir de deux corpus simulés.

9.3.2 Probabilité d'un alignement temporel

Un alignement temporel entre une suite d'états et une suite d'observations est une correspondance (état, observation). Sa probabilité est définie par :

$$Prob(Q,O) = \prod_t a_{q_{t-1}q_t} b_{q_t}(o_t) \tag{9.4}$$

dans lequel le terme $a_{q_{t-1}q_t}$ représente la probabilité de la contribution de la chaîne de Markov entre les instants t-1 et t. Quant au terme $b_{q_t}(o_t)$ il représente la vraisemblance de l'observation o_t dans l'état q_t.

Un grand intérêt des modèles de Markov cachés est qu'ils permettent lors de la reconnaissance une recherche de tous les alignements afin de retrouver la segmentation possédant la probabilité définie par l'équation 9.4 maximale. Pour réaliser cette recherche, un algorithme couramment utilisé est généralement *l'algorithme de Viterbi* (section 9.3.3) qui est un algorithme de programmation dynamique [Forney, 1973]. Il existe d'autres algorithmes pour réaliser cette recherche tels que par exemple l'algorithme "forward-backward" que nous utilisons dans la section 11.2. La différence essentielle entre ces deux algorithmes repose sur le critère d'optimalité choisi. Dans l'algorithme de Viterbi, l'alignement le plus probable au temps t est défini par la séquence d'états qui méne à l'état le plus probable au temps t. Alors que dans l'algorithme "forward-backward", l'alignement le plus probable au temps t est défini par la séquence d'états les plus probables à chaque instant précédent t.

9.3.3 L'algorithme de Viterbi (reconnaissance)

Dans les modèles de Markov cachés, plusieurs séquences d'états peuvent générer la même séquence observée $O_1,\ldots,O_T$. Etant donné une séquence observée, nous cherchons à déterminer la séquence d'états $Q_1,\ldots,Q_T$ la plus vraisemblable qui aurait pu générer cette séquence d'observations. Cette séquence se calcule en maximisant la probabilité jointive de sortie des états $prob((Q_1,\ldots,Q_T = q_1,\ldots,q_T),\ (O_1,\ldots,O_T = o_1,\ldots,o_T)\ /\ \lambda\ )$ où $Q_1..Q_T = q_1..q_T$ signifie que la séquence de variables aléatoires $Q_1,\ldots,Q_T$ prend les valeurs $q_1,\ldots,q_t$ et la condition λ spécifie les matrices A et B. Etant donné une séquence d'états, la probabilité jointive, qui sera dénotée par $prob((q_1,\ldots,q_T),\ (o_1,\ldots,o_T)\ /\ \lambda)$ est donnée par :

$$prob((q_1,...,q_T),(o_1,...,o_T)/\lambda) = \pi_{q_1} b_{q_1}(o_1) \prod_{t=2}^{T} a_{q_{t-1}q_t} b_{q_t}(o_t) \tag{9.5}$$

La séquence d'état qui maximise la probabilité jointive précédente est calculée par l'algorithme de Viterbi qui est une recherche dynamique qui calcule la meilleure séquence partielle d'état jusqu'au temps t pour tous les états. Donc, nous résolvons récursivement pour t de 2 à T pour la probabilité $\delta_t(j)$ du meilleur alignement partiel se terminant à la transition $(s_i,\ s_j)$ au temps t en maximisant :

$$\delta_t(j) = max_{1\leq i\leq N}[\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t),\ \ 2 \leq t \leq T,\ \ 1 \leq j \leq N \tag{9.6}$$

avec l'initialisation suivante :

$$\delta_1(i) = \pi_i b_i(O_1),\ \ 1 \leq i \leq N$$

La séquence d'état la plus vraisemblable $q_1,...,q_T$ est obtenue en gardant la trace pour chaque pas de calcul de la transition précédente qui conduit au chemin partiel ayant la plus haute probabilité. En revenant de l'état final à l'état initial, nous obtenons la meilleure séquence d'états.

Exemple

Reprenons les deux modèles que nous avons définis dans la section 9.2 pour représenter une intersection et un couloir. Admettons maintenant qu'après apprentissage les valeurs des matrices de transitions et d'observations soient :

– Modèle d'intersection

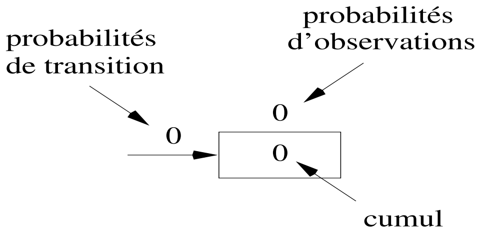
$$A = \begin{pmatrix} 0.26 & 0.74 & 0 \\ 0 & 0.26 & 0.74 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 0.14 & 0.08 & 0.08 & 0.7 \\ 0 & 0.24 & 0.36 & 0.4 & 0 \\ 0.82 & 0.07 & 0.06 & 0.05 & 0 \end{pmatrix}$$

– Modèle de couloir

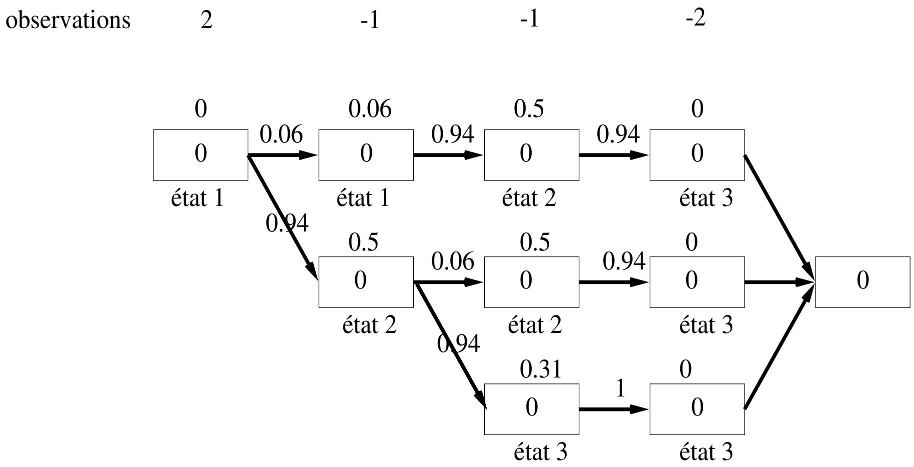
$$A = \begin{pmatrix} 0.06 & 0.94 & 0 \\ 0 & 0.06 & 0.94 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 0.06 & 0.5 & 0.44 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0.31 & 0.69 & 0 & 0 \end{pmatrix}$$

La construction de ces matrices est détaillée en exemple du paragraphe 9.3.5.

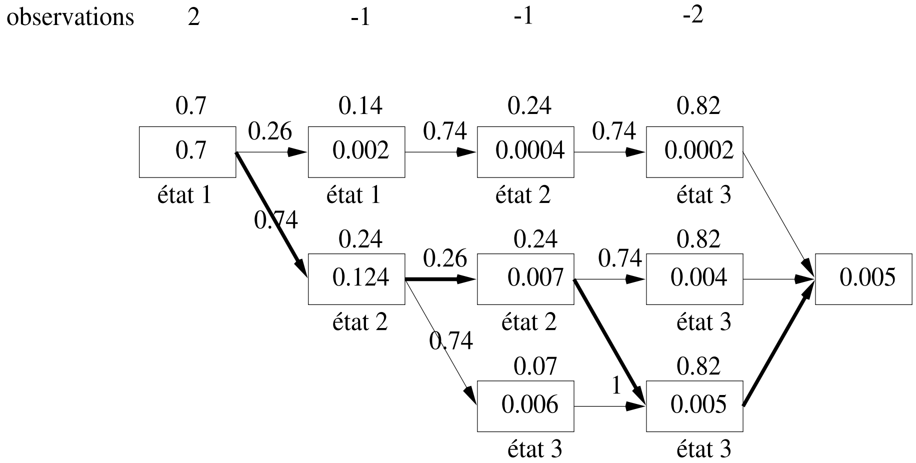
Mettons en compétition le modèle d'intersection et le modèle de couloir sur la séquence d'observations (2 -1 -1 -2) correspondante aux mesures du capteur perpendiculaire au mur :



– Modèle de couloir



– Modèle d'intersection



On constate que le taux de reconnaissance du modèle d'intersection est meilleur que celui du modèle de couloir, donc la séquence d'observations (2 -1 -1 -2) correspond à une intersection. Intuitivement, si on analyse cette séquence d'observations, on se rend compte qu'elle correspond tout à fait au modèle d'intersection que nous avons décidé d'apprendre. En effet, la séquence d'observations commence par une forte augmentation de la mesure correspondante au début de l'intersection, puis se poursuit par des faibles diminution de la mesure correspondante au passage devant l'intersection, et se termine par une forte diminution de la mesure correspondante à la fin de l'intersection.

Remarque

La probabilité des différents modèles d'avoir reconnu une séquence d'observations ne doit être vue que d'une façon relative. Elle permet uniquement de savoir quel est le modèle qui a le plus de chance de correspondre à la séquence d'observations. Dans l'absolu, cette probabilité n'a aucune signification, et ne permet absolument pas de savoir si l'observation a une grande ou une faible probabilité de correspondre au modèle le plus probable.

9.3.4 Les critères d'apprentissage

Il existe différents critères d'estimation d'apprentissage. Nous citons ici les deux critères les plus courants :

– Le plus connu et le plus généralement utilisé est le critère d'estimation selon le maximum de vraisemblance. C'est le critère choisi dans l'outil que nous utilisons. Pour ce critère, on cherche θ qui rende maximale la quantité :

$$\prod_{i=1}^{N} prob_\theta(O^N/\lambda),$$

où O^N sont les séquences d'observations des échantillons d'apprentissage. Ce produit représente la vraisemblance du corpus d'apprentissage. Ceci revient à rendre maximum :

$$\sum_{i=1}^{N} \log(prob_\theta(O^N/\lambda))$$

Il faut noter que ce critère ne permet pas de distinguer les modèles comme le ferait un réseau de neurones, mais il essaie seulement d'augmenter la probabilité qu'un modèle génère le corpus qu'il a appris indépendamment de ce qu'apprennent les autres modèles.

– Un autre critère très connu est le critère d'estimation du maximum d'information mutuelle. Dans cette estimation, on cherche θ qui rende maximum l'information mutuelle entre les observations O^N du corpus d'apprentissage et le modèle λ :

$$I(O^N,\lambda) = \log \frac{prob_(O^N,\lambda)}{prob_(O^N) prob_(\lambda)}$$

$$= \log \frac{prob_(O^N|\lambda) prob_(\lambda)}{prob_(O^N) prob_(\lambda)}$$

$$= \log prob_(O^N|\lambda) - \log prob_(O^N)$$

où la somme s'exécute sur tous les modèles en compétition.

La thèse de [Brown, 1987] pose les problèmes d'estimation par maximum de vraisemblance et maximum d'information mutuelle en décrivant bien les intérêts de chacune de ces deux méthodes ainsi que leurs relations.

9.3.5 L'algorithme de Baum-Welch (apprentissage)

L'apprentissage des modèles est réalisé en utilisant le critère d'estimation du maximum de vraisemblance qui détermine les meilleurs paramètres du modèle en fonction du corpus d'apprentissage.

Comme plusieurs séquences d'états peuvent générer une séquence de sortie donnée, la probabilité qu'un modèle génère une séquence $o_1,...,o_T$ est donnée par la somme des probabilités jointives (donnée dans l'équation 9.5) sur toutes les séquences d'états (i.e, la densité marginale des séquences de sorties). Pour éviter une explosion combinatoire, un calcul récursif similaire à l'algorithme de Viterbi (section 9.3.3) peut être utilisé pour évaluer la somme au-dessus. La probabilité "forward" $\alpha_t(i)$ est :

$$prob((O_1,...,O_t = o_1,...,o_t),q_t = s_i/\lambda) \tag{9.7}$$

Cette probabilité représente la probabilité de démarrer à l'état 1 et de terminer à l'état s_i au temps t et de générer la sortie $o_1,...,o_t$ en utilisant toutes les séquences d'états possibles. La propriété de Markov permet un calcul récursif de la probabilité "forward" comme :

$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i).a_{ij}.b_j(O_{t+1}), \;\; 1 \leq t \leq T-1, \;\; 1 \leq j \leq N \tag{9.8}$$

avec l'initialisation suivante :

$$\alpha_1(i) = \pi_i.b_i(O_i), \;\; 1 \leq i \leq N$$

Ce calcul est similaire à l'algorithme de Viterbi (section 9.3.3) si ce n'est que la somme est utilisée à la place du maximum. La valeur $\alpha_T(j)$ est la probabilité que le modèle M génère la séquence $o_1,...,o_t$.

Une autre quantité très utile est la fonction "backward" $\beta_t(i)$, définie comme la probabilité de la séquence d'observations partielles de t+1 à T commençant à l'état i, étant donné le modèle λ, qui peut être exprimée comme :

$$\beta_t(i) = Prob(O_{t+1},...O_T,q_t = s_i/\lambda), \tag{9.9}$$

Cette quantité peut elle aussi être calculée récursivement :

$$\beta_t(j) = \sum_{i=1}^{N} a_{ji}.b_i(O_{t+1})\beta_{t+1}(i), \;\; t = T-1,T-2,...,1, \;\; 1 \leq j \leq N \tag{9.10}$$

Les probabilités "forward" et "backward" sont utilisés pour estimer un modèle.

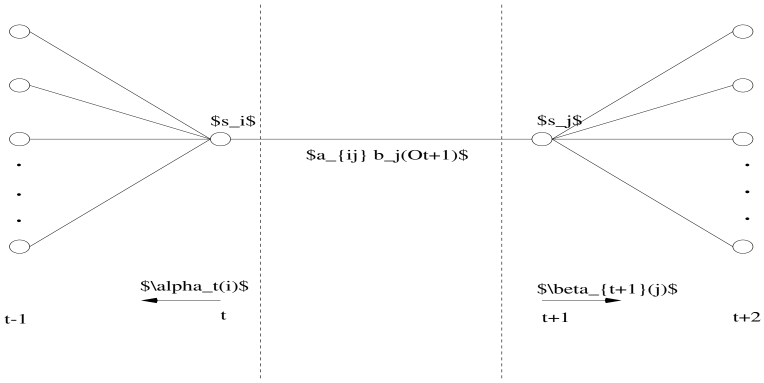


FIG. 9.4 – *Expression de $\xi_t(i,j)$ en fonction de α et β*

Estimation du modèle

Pour estimer les paramètres d'un modèle à partir d'une séquence d'observations $o_1,...,o_T$ et des paramètres courants du modèle λ, nous définissons la probabilité :

$$\xi_t(i,j) = Prob(q_t = s_i,q_{t+1} = s_j/(O_1,...,O_t = o_1,...,o_t),\lambda) \tag{9.11}$$

qui correspond à la probabilité d'être à l'état s_i au temps t, et à l'état s_j au temps t+1, étant donné le modèle λ et la séquence d'observations $o_1,...,o_t$. Il est clair qu'en utilisant les variables "forward" et "backward" nous écrivons $\xi_t(i,j)$ sous la forme (cf. figure 9.4) :

$$\xi_t(i,j) = \frac{\alpha_t(i).a_{ij}.b_j(O_{t+1}).\beta_{t+1}(j)}{Prob(O/\lambda)}$$

où le numérateur représente Prob($q_t = s_i,q_{t+1} = s_j$, O/λ) et la division par Prob(O/λ) donne la probabilité désiré.

Nous définissons $\gamma_t(i)$ comme la probabilité d'être dans l'état s_i au temps t, étant donné la séquence d'observations et le modèle. Donc :

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Si nous sommons $\gamma_t(i)$ sur l'indice de temps t, nous obtenons une quantité qui peut être interprétée comme le nombre de fois où l'état s_i est visité, ou de manière équivalente, le nombre de transitions faites à partir de l'état s_i. De la même manière, la somme sur t de $\xi_t(i,j)$ peut être interprétée comme le nombre de fois où la transition $(s_i,\, s_j)$ a été visitée. C'est à dire :

$$\sum_{t=1}^{T-1} \gamma_t(i) = \textit{Nombre de fois où l'état } s_i \textit{ est visité}$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \textit{Nombre de fois où la transition } (s_i,s_j) \textit{ a été visitée}$$

Nous pouvons donc donner une méthode pour réestimer les paramètres d'un modèle de Markov caché :

$$\begin{aligned} a_{ij} &= \frac{\textit{Nombre de fois où la transition } (s_i,s_j) \textit{ a été visitée}}{\textit{Nombre de fois où l'état } s_i \textit{ est visité}} \\ &= \frac{\sum_{t=1}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \xi_t(i,j)} \end{aligned}$$

et

$$\begin{aligned} b_j(k) &= \frac{\textit{Nombre de fois où l'état } s_j \textit{ est visité en observant le symbole } v_k}{\textit{Nombre de fois où l'état } s_j \textit{ est visité}} \\ &= \frac{\sum_{t_{avec O_t = v_k}=1}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \xi_t(i,j)} \end{aligned}$$

En normalisant a_{ij} et $b_j(k)$, on obtient une nouvelle estimée de A et B.

Remarque

Intuitivement, cet algorithme compte le nombre d'occurences de chaque transition entre les états dans le corpus d'apprentissage, ainsi que le nombre d'occurences de chaque observation pour un état donné dans le corpus d'apprentissage. Chaque compte est pondéré par la probabilité de l'alignement (état, observation). Dans le cas où les observations sont continues, l'algorithme estime $c_{ij},\mu_{ij},\sigma_{ij}$, $1 \leq i \leq N$, $1 \leq j \leq d$.

Réestimation successive du modèle

En pratique, dans l'outil que nous avons utilisé les modèles sont réestimés plusieurs fois en utilisant le corpus d'apprentissage, comme décrit dans l'algorithme 9.1. En effet, d'après le théorème de Baum[Baum, 1972], si A et B sont les matrices initiales et $\bar{A}$ et $\bar{B}$ sont les matrices réestimées, alors nous avons deux cas possibles :

- Les matrices A et B définissent un point critique (ce qui est rare) de la fonction de vraisemblance, $P(O/\lambda)$. Dans ce cas, nous avons $A = \bar{A}$ et $B = \bar{B}$.

entrée : Les matrices A et B initiales (c-à-d, avant apprentissage), O l'ensemble de séquences d'observations (c-à-d, le corpus d'apprentissage) et NB le nombre d'itérations.
sortie : Les matrices $\bar{\mathrm{A}}$ et $\bar{\mathrm{B}}$ réestimées (c-à-d, après apprentissage)

pour K de 1 a NB **faire**
1) Calculer $\xi_t(i,j)$ et $\gamma_t(i)$ pour $1 \leq i,j \leq N$, $1 \leq t \leq T-1$.
2) En deduire une nouvelle estimee de A et B.
fin-pour

Algorithme 9.1: Algorithme de réestimation des matrices de transition et d'observation

– Les matrices $\bar{\mathrm{A}}$ et $\bar{\mathrm{B}}$ sont plus probables que les matrices A et B. Nous avons donc trouvé une matrice de transitions $\bar{\mathrm{A}}$ et une matrice d'observations $\bar{\mathrm{B}}$ à partir desquelles la suite d'observations a plus de chance d'être produite. Cette réestimation permet donc d'augmenter la probabilité de reconnaître les séquences d'observations qui se trouvent dans le corpus d'apprentissage.

Cet algorithme appelle deux remarques :

– L'algorithme converge vers des valeurs de paramètres qui forment un point critique de $\mathrm{P}(\mathrm{O}/\lambda)$. Donc, nous obtenons un maximum local ou un point d'inflexion. D'où la nécessité d'un bon choix des matrices initiales pour éviter les points d'inflexion. Dans notre exemple, ainsi que dans la grande majorité des applications utilisant des MMCs, pour essayer d'éviter ce problème, nous choisissons des matrices initiales de transitions (resp. d'observations) où toutes les transitions (resp. observations) sont équiprobables.
– Il existe dans la littérature un algorithme qui fixe les conditions d'arrêts à vérifier par le point critique et permet ainsi d'arrêter automatiquement le cycle d'itération. Cet algorithme est l'algorithme EM (Expectation Modification) [Dempster *et al.*, 1977]. Cet algorithme n'étant pas implanté dans l'outil que nous utilisons, le test d'arrêt est simplement le nombre d'itérations qui est fixé empiriquement.
– La taille du corpus influe, elle aussi, sur les résultats.

Exemple

Nous allons montrer comment à partir d'un ensemble de séquences d'observations simulé le plus représentatif possible de la caractérisation d'une balise, nous estimons les modèles de chaque balise définis précédemment.

Apprentissage du modèle d'intersection

Nous définissons un corpus d'apprentissage de cinq séquences d'observations le plus représentatif possible du modèle d'intersection. Ce corpus est simulé, et ne vise qu'à illustrer l'apprentissage du modèle d'intersection. Chaque séquence commence en effet par une forte augmentation de la mesure du capteur perpendiculaire au mur, puis se poursuit par une partie stationnaire (i.e, une suite de faible ou très faible variation) de plus ou moins longue durée, et se termine par une forte diminution de la mesure du capteur perpendiculaire au mur.

$$\begin{pmatrix} 2 & 0 & -2 & & \\ 2 & 1 & -2 & & \\ 2 & 0 & -1 & -2 & \\ 2 & 1 & 1 & -2 & \\ 2 & -1 & -1 & 0 & -2 \end{pmatrix}$$

A partir de ces cinq séquences d'observations, pour estimer le modèle d'intersection, nous comptons le nombre d'occurences de chaque transition entre les états ainsi que le nombre d'occurrences de chaque observation pour un état donné dans le corpus d'apprentissage. Il n'est pas nécessaire de pondérer la probabilité de l'alignement (état, observation), puisque d'après le modèle initial, toutes les transitions et toutes les observations sont équiprobables.

Intuitivement, nous pouvons penser que le premier état contiendra la brusque augmentation du signal correspondante au début de l'intersection, le deuxième état contiendra la partie stationnaire du signal (où la dérivée est proche de zéro) et le troisième état contiendra la brusque diminution du signal correspondante à la fin de l'intersection.

Dans le paragraphe suivant, nous nous intéressons en particulier à la séquence (2 1 1 -2).

Apprentissage de (2 1 1 -2)

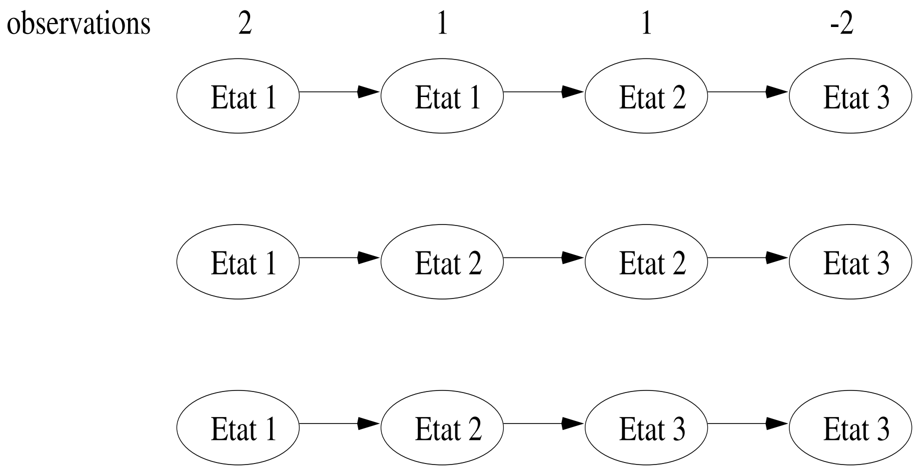


FIG. 9.5 – *Apprentissage de (2 1 1 -2)*

On cherche tout d'abord le nombre de chemins composée de quatre états, en utilisant le modèle d'intersection. Ils sont au nombre de trois. Etudions maintenant ce qui se passe lorsqu'on est à l'état un sur l'ensemble de ces trois chemins :

- Pour le premier chemin, on va une fois à l'état un et une fois à l'état deux.
- Pour le deuxième chemin, on va une fois à l'état deux.
- Pour le troisième chemin, on va une fois à l'état deux.

Donc, pour l'ensemble des chemins, en partant de l'état un, on va une fois à l'état un et trois fois à l'état deux. En faisant la même opération pour l'état deux et l'état trois, on obtient :

$$A = \begin{pmatrix} 1 & 3 & 0 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

En normalisant A, on obtient une nouvelle estimée correspondant à la séquence d'observations (2 1 1 -2) :

$$A = \begin{pmatrix} \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 1 \end{pmatrix}$$

Faisons de même, pour la matrice B :
- Pour le premier chemin, on se trouve deux fois à l'état un. On y observe une fois un 2, et une fois un -1.
- Pour le deuxième chemin, on se trouve une fois à l'état un. On y observe un 2.
- Pour le troisième chemin, on se trouve une fois à l'état un. On y observe un 2.

Donc, pour l'ensemble des chemins, on se trouve quatre fois à l'état un, on y observe trois fois un 2, et une fois un 1. En faisant la même opération pour l'état deux et l'état trois, on obtient :

$$B = \begin{pmatrix} 0 & 0 & 0 & 1 & 3 \\ 0 & 0 & 4 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \end{pmatrix}$$

En normalisant B, on obtient une nouvelle estimée correspondant à la séquence d'observations (2 1 1 -2) :

$$B = \begin{pmatrix} 0 & 0 & 0 & \frac{1}{4} & \frac{3}{4} \\ 0 & 0 & 1 & 0 & 0 \\ \frac{3}{4} & 0 & \frac{1}{4} & 0 & 0 \end{pmatrix}$$

Prise en compte de tout le corpus d'apprentissage

En effectuant la même opération pour chacune des séquences d'observations, puis en sommant les matrices A et B correspondantes, et en normalisant, on obtient le modèle d'intersection réestimé suivant :

$$A = \begin{pmatrix} 0.26 & 0.74 & 0 \\ 0 & 0.26 & 0.74 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 0.14 & 0.08 & 0.08 & 0.7 \\ 0 & 0.24 & 0.36 & 0.4 & 0 \\ 0.82 & 0.07 & 0.06 & 0.05 & 0 \end{pmatrix}$$

Si l'on regarde la matrice B, on se rend compte que lors de la reconnaissance :
- l'état un va augmenter les chances d'une séquence d'observations commençant par une forte augmentation.
- l'état deux va augmenter les chances d'une séquence d'observations ayant au milieu une partie variant faiblement ou très faiblement.

– l'état trois va augmenter les chances d'une séquence d'observations se terminant par une forte diminution.

On constate donc que notre modèle d'intersection sera tout à fait apte à reconnaître des séquences d'observations commençant par une forte augmentation, continuant par une partie ne variant quasiment pas, et se terminant par une forte diminution. Ce type de séquences d'observations correspond tout à fait à une intersection.

Apprentissage du modèle de couloir

De la même façon, nous définissons un corpus d'apprentissage de quatre séquences d'observations le plus représentatif possible du modèle de couloir. Chaque séquence est constituée d'une suite de faible ou très faible variation de la mesure du capteur perpendiculaire au mur.

$$\begin{pmatrix} 0 & 0 & 0 & \\ 0 & -1 & -1 & \\ 1 & 0 & 0 & \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

En effectuant la même opération que dans le paragraphe précédent pour chacune des séquences d'observations, on obtient le modèle de couloir réestimé suivant :

$$A = \begin{pmatrix} 0.06 & 0.94 & 0 \\ 0 & 0.06 & 0.94 \\ 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 0 & 0.06 & 0.5 & 0.44 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0.31 & 0.69 & 0 & 0 \end{pmatrix}$$

Remarque

Les matrices A et B estimées du modèle d'intersection et de couloir sont celles que nous avons utilisées pour la reconnaissance dans la section 9.3.3.

Comme nous l'avons dit précédemment, dans un cadre réaliste, chaque compte est pondérée par la probabilité de l'alignement (état, observation). Dans cet exemple simple, nous n'avons pas tenu compte, lors du comptage, de la probabilité de l'alignement (état, observation) pour la simple raison que nous avons choisie un modèle dans lequel au départ tous les alignements ont la même probabilité.

9.3.6 Conclusion

Dans cette section, nous avons défini les algorithmes d'apprentissage et de reconnaissance pour les modèles d'ordre un, et les principaux critères utilisés pour l'apprentissage.

L'exemple choisi illustre bien le fait que les modèles de Markov cachés sont tout à fait adaptés à l'apprentissage et à la reconnaissance de balises naturelles dans un environnement intérieur structuré. Dans la section suivante, nous étendons les algorithmes de Viterbi et de Baum-Welch pour le second ordre.

9.4 L'apprentissage et la reconnaissance pour le second ordre

9.4.1 Introduction

Notre but [Aycard *et al.*, 1997a] est d'utiliser les modèles de Markov cachés pour apprendre et reconnaître des balises naturelles. Pour cela, un outil développé dans l'équipe est mis à notre disposition. Cet outil utilise des modèles du second ordre. En effet, [Mari *et al.*, 1997] ont montré que dans de nombreux cas, les modèles de Markov cachés du second ordre donnent de meilleurs résultats (c-à-d, un meilleur taux de reconnaissance) que ceux du premier ordre. Nous utilisons donc des modèles de Markov cachés du second ordre.

Dans cette section, nous présentons les extensions à apporter aux modèles de Markov cachés pour le second ordre.C'est le type de modèle implanté dans l'outil utilisé. Cette extension est reprise des travaux de [Mari, 1996].

9.4.2 Probabilité d'un alignement temporel

A l'ordre 2, la probabilité d'un alignement temporel est définie par :

$$Prob(Q,O) = \prod_t a_{q_{t-2}q_{t-1}q_t} b_{q_t}(o_t) \tag{9.12}$$

dans lequel le terme $a_{q_{t-2}q_{t-1}q_t}$ représente la probabilité de la contribution de la chaîne de Markov entre les instants t-2 et t. Quant au terme $b_{q_t}(o_t)$ il représente la vraisemblance de l'observation o_t dans l'état q_t.

9.4.3 L'algorithme de Viterbi pour le second ordre

L'extension de l'algorithme de Viterbi au deuxième ordre ne présente pas de difficultés majeures dans sa définition. On remplace simplement la référence à un état de l'ensemble **S** par une référence à un élément du produit cartésien **S x S**. La probabilité du meilleur alignement entre une suite partielle d'observations $o_1,o_2,\ldots,o_t$ et une suite d'états $q_1,...q_{t-2},q_{t-1} = s_j,q_t = s_k$ s'achevant par la transition $s_j \longrightarrow s_k$ entre les instants t-1 et t s'écrit :

$$\delta_t(j,k) = Max_{q_1,...q_{t-2}} Prob(q_1,...q_{t-2},q_{t-1} = s_j,q_t = s_k,O_1,...,O_t/\lambda), \quad \begin{array}{lll} 1 & \leq j,k & \leq N \\ 2 & \leq t & \leq T \end{array}$$

Un calcul récursif peut se développer de la façon suivante :

$$\delta_t(j,k) = Max_{1\leq i\leq N}[\delta_{t-1}(i,j) \cdot a_{ijk}] \cdot b_k(O_t), \quad \begin{array}{lll} 1 & \leq j,k & \leq N \\ 3 & \leq t & \leq T \end{array} \tag{9.13}$$

A partir des initialisations suivantes :

$$\begin{array}{lll} \delta_1(i) & = & \Pi_i \cdot b_i(O_1), \qquad 1 \leq i \leq N \\ \delta_2(i,j) & = & \delta_1(i) \cdot a_{ij} \cdot b_j(O_2), \quad 1 \leq i,j \leq N \end{array}$$

9.4.4 L'algorithme de Baum-Welch pour le second ordre

Pour estimer les paramètres d'un MMC2, nous définissons les nouvelles fonctions "forward" et "backward" sur lesquelles est basé l'algorithme de Baum-Welch étendu au deuxième ordre.

La fonction $\alpha_t(j,k)$ définit la probabilité d'avoir simultanément la suite d'observations partielle
$O_1,...,O_t$ et la transition $s_j \longrightarrow s_k$ entre les instants $t-1$ et t.

$$\alpha_t(j,k) = Prob(O_1,O_2,...,O_t,q_{t-1} = s_j,q_t = s_k/\lambda), \quad \begin{array}{lll} 2 & \leq t & \leq T \\ 1 & \leq j,k & \leq N \end{array}$$

Comme pour le premier ordre, $\alpha_{t+1}(j,k)$ peut être calculée à l'aide de $\alpha_t(i,j)$ où $s_i \longrightarrow s_j$ et $s_j \longrightarrow s_k$ sont deux transitions entre les instants $t-1$ et t d'une-part et t et $t+1$ d'autre-part.

$$\alpha_{t+1}(j,k) = \sum_{i=1}^{N} \alpha_t(i,j).a_{ijk}.b_k(O_{t+1}), \quad 2 \quad \leq t \quad \leq T-1 \tag{9.14}$$

$$1 \quad \leq j,k \quad \leq N \tag{9.15}$$

On définit aussi la fonction $\beta_t(i,j)$, comme étant la probabilité de la suite d'observations depuis $t+1$ à T, connaissant le modèle λ et la transition $s_i \longrightarrow s_j$ entre les instants t-1 et t.

$$\beta_t(i,j) = Prob(O_{t+1},...O_T/q_{t-1} = s_i,q_t = s_j,\lambda), \quad \begin{array}{lll} 2 & \leq t & \leq T-1 \\ 1 & \leq i,j & \leq N \end{array}$$

Tout comme pour le premier ordre, cette quantité peut être définie récursivement :

$$\beta_t(i,j) = \sum_{k=1}^{N} \beta_{t+1}(j,k) a_{ijk} b_k(O_{t+1}), \quad 2 \quad \leq t \quad \leq T-1 \tag{9.16}$$

$$1 \quad \leq i,j \quad \leq N \tag{9.17}$$

9.5 Conclusion

Dans ce chapitre, nous avons défini les modèles de Markov cachés du premier ordre et du second ordre avec des observations continues ou discrètes, ainsi que les algorithmes d'apprentissage et de reconnaissance d'un modèle.

D'autre part, nous avons montré sur un exemple simple que les modèles de Markov cachés sont tout à fait adaptés à l'apprentissage et à la reconnaissance des événements temporels qui caractérisent une balise naturelle dans un environnement intérieur structuré. Ils possèdent toutes les conditions requises pour apprendre et reconnaître des balises naturelles :

- Ils permettent à partir d'une séquence d'observations d'apprendre et de reconnaître les événements temporels qui caractérisent un type de balise naturelle.
- Chaque observation est composée de la mesure d'un ou plusieurs capteurs.
- La reconnaissance permet de connaître le type de balise reconnue.
- Dans le chapitre suivant, nous verrons aussi qu'ils sont capables d'apprendre à caractériser des balises plus complexes qu'une intersection, en utilisant d'autres caractéristiques telles par exemple que la longueur des balises (c-à-d, le nombre d'observations utilisés pour l'apprendre et la reconnaître).

Leur seul inconvénient est qu'il ne délivre aucune information sur la position et l'orientation du robot par rapport à la balise reconnue. D'autre part, ces modèles nécessitent de gros corpus d'apprentissage qui de plus doivent être le plus représentatif possible du signal que nous voulons caractériser. Comme nous le verrons dans le chapitre suivant, la constitution de ces corpus est longue et inévitable. Dans le chapitre suivant, nous expliquons comment nous avons utilisé l'outil, à base de MMC2 avec des observations continues, mis à notre disposition pour apprendre et reconnaître des balises naturelles dans un environnement réel.

10

Notre approche pour apprendre et reconnaître des balises

10.1 Introduction

Nous avons illustré dans le chapitre précédent sur un exemple simple, que les modèles de Markov cachés étaient aptes à apprendre et reconnaître des balises naturelles.

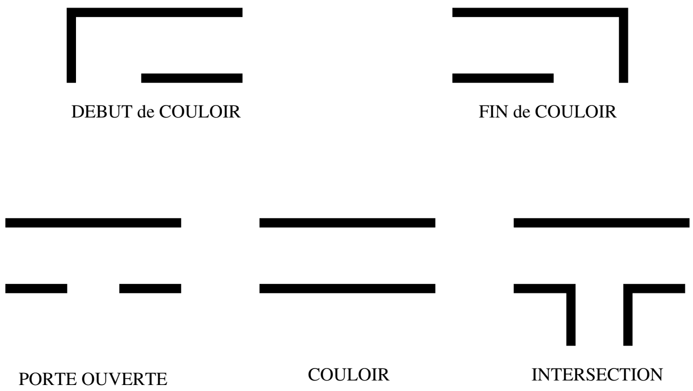


FIG. 10.1 – *Les balises naturelles constituant notre environnement*

Dans ce chapitre, nous montrons comment nous avons utilisé l'outil mis à notre disposition pour apprendre et reconnaître des balises [Aycard *et al.*, 1997a]. Nous choisissons d'apprendre cinq balises distinctes (figure 10.1) : un couloir[23], un début de couloir, une fin de couloir, une intersection et une porte ouverte. Nous avons choisi ces cinq balises, parce qu'elles sont représentatives de notre environnement et d'un environnement intérieur structuré en général.

Pour réaliser l'apprentissage et la reconnaissance de balises naturelles, nous avons un certain nombre de problèmes à résoudre :

- Définir un modèle de Markov caché représentant chaque balise. Nous présentons dans la section 10.2 le modèle de Markov caché que nous avons choisi.

23. Il faut faire la différence entre la balise couloir qui est constituée par deux murs l'un en face de l'autre, et un couloir qui est constitué d'une suite de balises.

- L'apprentissage étant réalisé à partir d'un ensemble de séquences d'observations, nous devons construire, à partir de données réelles, les cinq corpus d'apprentissage qui nous permettent d'apprendre les cinq modèles de balises naturelles. La constitution des différents corpus d'apprentissage fait l'objet de la section 10.3.
- Lors de l'apprentissage et de la reconnaissance, il nous faut déterminer les capteurs constituant l'entrée du modèle (ie, l'observation). D'autre part, lors de la reconnaissance, le robot n'a plus à reconnaître une seule balise, mais toutes les balises qui l'a vues lors de son déplacement. Dans la section 10.4, nous expliquons comment nous avons tenu compte de ce problème lors de l'apprentissage. La section 10.5 est consacrée à la reconnaissance, et à la mise en place d'une procédure pour évaluer le taux de reconnaissance.

Dans la section 10.6, nous présentons les résultats obtenus lors de la mise en œuvre de notre approche. Les expériences ont consisté à mesurer ses capacités à détecter et identifier les balises lors du cheminement du robot Gaston dans les couloirs du LORIA. Une analyse des premiers résultats obtenus est ensuite présentée et nous conduit à proposer deux améliorations que nous exposons dans la section 10.7. Une nouvelle expérimentation nous permet de confirmer le bien fondé de ces améliorations.

10.2 Le modèle de Markov caché représentant chaque balise

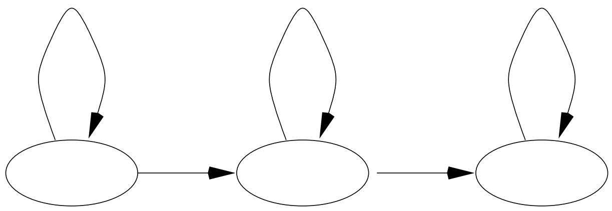


FIG. 10.2 – *Le modèle Gauche Droite à trois états retenu pour représenter chaque balise*

Pour les raisons évoquées dans le chapitre précédent, une balise est représentée par le modèle Gauche-Droite à trois états au centre de la figure 10.2.

La matrice A de probabilités de transitions (avant apprentissage) et le vecteur de probabilités initiales π de chaque balise, étant donné le modèle au centre de la figure 9.1, sont :

$$A = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \pi = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Chaque état s_i est associé à un mélange de distributions gaussiennes :

$$b_i(O_t) = \sum_{m=1}^{M} c_{im} \mathcal{N}(O_t; \mu_{im}, \sigma_{im}), \tag{10.1}$$

$$avec \quad \sum_{m=1}^{M} c_{im} = 1$$

où O_t est l'observation au temps t. La définition des capteurs constituant l'observation sera définie dans la section 10.4. En effet, elle sera définie en fonction des capteurs pertinents pour caractériser un événement temporel. Nous avons décidé d'utiliser des observations continues constituées des valeurs de plusieurs capteurs pour deux raisons : premièrement, nous avons vu que la définition d'une bonne approche pour réaliser l'apprentissage et la reconnaissance doit prendre en compte les valeurs de plusieurs capteurs pour constituer l'observation et deuxièmement, dans un cadre réel, les mesures des capteurs ne sont pas discrètes. Nous aurions évidemment pu discrétiser l'espace des mesures des capteurs, mais ceci aurait probablement généré une perte d'informations.

Le modèle défini ci-dessus constitue le modèle initial de chacune des cinq balises à apprendre et à reconnaître.

10.3 La constitution des corpus d'apprentissage

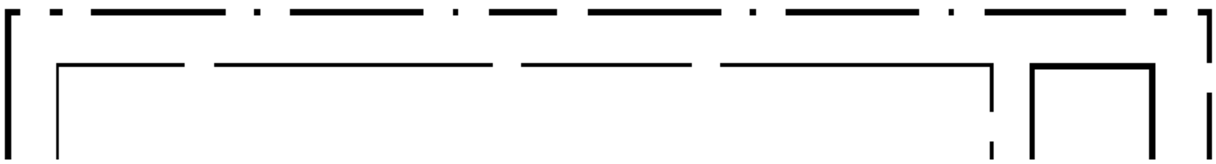


FIG. 10.3 – *Le couloir utilisé pour réaliser notre corpus d'acquisition*

Pour apprendre une balise, un MMC a besoin d'observations de cette balise, pour déduire des propriétés statistiques intéressantes de ces observations. Nous devons construire un corpus pour entraîner le modèle de chaque balise. Pour cela, notre robot mobile a effectué cinquante allers et retours dans un long couloir (approximativement trente mètres) en utilisant le module de navigation, et nous avons enregistré l'ensemble des mesures effectuées par ses capteurs ultrasons[24] au cours de chaque traversée. Ce couloir (figure 10.3) peut être complètement décrit à partir des cinq balises que nous avons définies. Il contient un début de couloir et une fin de couloir, une intersection et des portes ouvertes (au moins quatre à chaque passage, mais pas toujours les mêmes). Les acquisitions sont réalisées dans des conditions réelles avec des personnes se déplaçant dans le couloir, des portes complètement ou partiellement ouvertes et des obstacles statiques comme des étagères ou des cartons.

Afin d'entraîner chaque modèle de balise, le corpus ainsi constitué est segmenté et étiqueté manuellement. La segmentation et l'étiquetage consistent à identifier les balises que le robot a vues lors de la traversée du couloir. Comme [Kortenkamp et Weymouth, 1994], nous définissons des règles qui permettent de segmenter chaque traversée en balise. Pour cela, nous devons choisir les capteurs pertinents à observer lors du passage devant une balise. Cette tâche est complexe car les mesures des capteurs sont bruitées et quand il y a une balise sur le côté gauche, il y en a une autre sur le côté droit du robot. Pour ces raisons,

24. Dans la suite, nous parlons par abus de langage de capteur plutôt que de capteur ultrason.

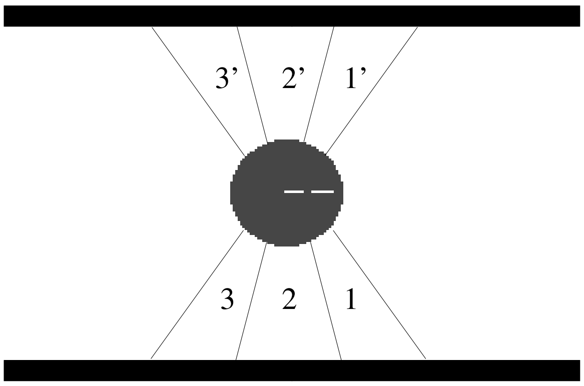


FIG. 10.4 – *Les six sonars utilisés pour la segmentation de chaque côté*

nous avons choisi de segmenter les traversées de manière différente pour chaque côté, en utilisant le sonar perpendiculaire à chaque mur du couloir et les deux sonars adjacents à celui-ci (figure 10.4). Ces trois sonars donnent normalement des mesures valides. Nous aurons donc pour chaque traversée de couloir deux segmentations : une pour le côté droit, et une pour le côté gauche. Une fois, ces règles définies nous les utilisons pour segmenter et étiqueter manuellement le corpus. Pour cela, nous avons developpé un éditeur de signal.

Nous montrons dans le paragraphe suivant, comment en utilisant ces trois capteurs nous définissons des règles pour segmenter chaque balise. Dans le dernier paragraphe, nous expliquons comment nous segmentons et étiquetons, à l'aide de notre éditeur de signal et en utilisant les règles, une traversée de couloir afin de savoir quelles balises le robot a vues.

10.3.1 La segmentation des balises

La segmentation d'une intersection

Nous segmentons le début d'une intersection lorsque la mesure du premier sonar augmente soudainement et la fin d'une intersection lorsque la mesure du dernier sonar diminue soudainement. La figure 10.5 montre un exemple de segmentation d'une acquisition correspondante à une intersection effectuée sur les trois capteurs droits. La première droite (la plus à gauche sur la figure) correspond au début de l'intersection (augmentation soudaine de la valeur du premier capteur), et la deuxième droite correspond à la fin de l'intersection (diminution soudaine de la valeur du troisième capteur).

La segmentation d'une porte ouverte

Nous segmentons une porte ouverte de la même façon qu'une intersection. La figure 10.6 montre un exemple de segmentation d'une acquisition correspondante à une porte ouverte effectuée sur les trois capteurs droits. Bien que la segmentation des portes ouvertes soient la même que celles des intersections, l'apprentissage devra utiliser d'autres

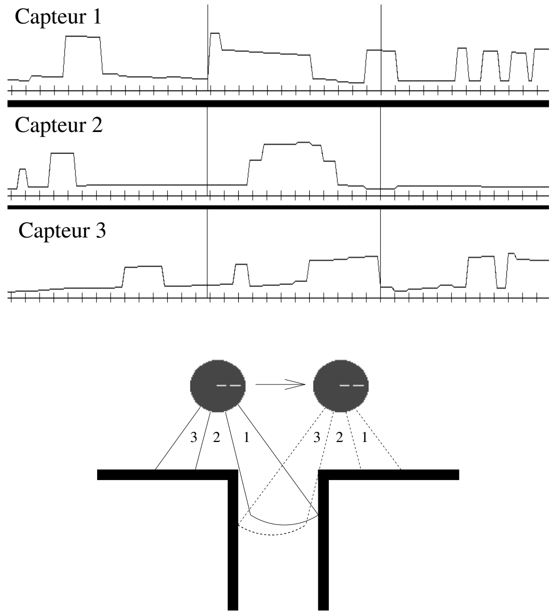


FIG. 10.5 – *Exemple de segmentation d'une intersection sur le côté droit*

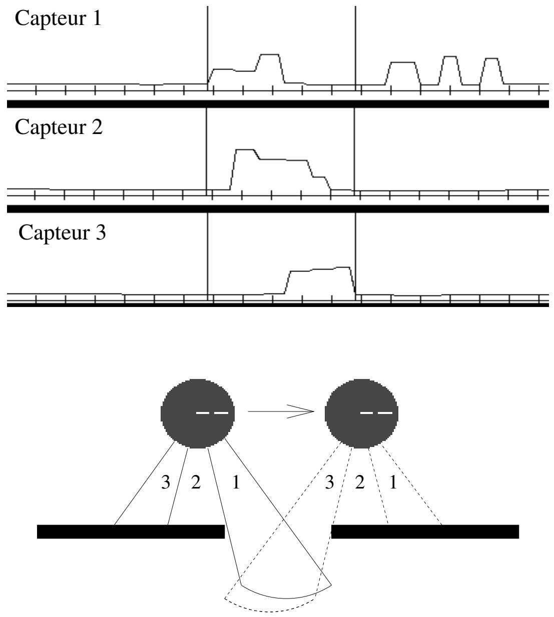


FIG. 10.6 – *Exemple de segmentation d'une porte ouverte sur le côté droit*

caractéristiques de chacune de ces deux types de balises telles que la forme du signal [25] ou la longueur de la balise, afin de les différencier.

La segmentation d'un début de couloir

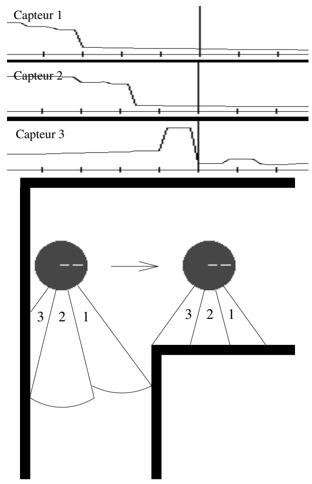


FIG. 10.7 – *Exemple de segmentation d'un début de couloir sur le côté droit*

Nous définissons le début d'un couloir lorsque le robot est au début du couloir, c'est à dire lorsque la mesure des trois capteurs est élevé, et la fin d'un début de couloir lorsque la mesure du troisième capteur diminue soudainement. La figure 10.7 montre un exemple de segmentation d'une acquisition correspondante à un début de couloir effectuée sur les trois capteurs droits.

La segmentation d'une fin de couloir

Nous définissons la fin d'un couloir lorsque la mesure du premier capteur augmente soudainement et la fin d'une fin de couloir lorsque le robot arrive au bout du couloir, c'est à dire lorsque la mesure des trois capteurs est élevé. La figure 10.8 montre un exemple de segmentation d'une fin de couloir effectuée sur les trois capteurs droits.

La segmentation d'un couloir

Une balise de type couloir se définit par pas ou peu de variations sur la mesure des capteurs. La figure 10.9 montre un exemple de segmentation d'une acquisition correspondante à un couloir effectué sur les trois capteurs droits. Nous voyons clairement dans cette figure que ces règles de segmentation ne sont suffisantes. En effet, la figure 10.9 représente les mesures des trois capteurs dans le temps qui devraient pas ou peu varier lorsque le robot se trouve dans un couloir. Pourtant nous voyons de fortes variations sur les deux capteurs adjacents au capteur perpendiculaire au mur qui ne correspondent qu'à du bruit.

25. Nous définissons la forme du signal d'une balise comme la forme de la représentation graphique des observations de la balise dans le temps, comme dans la figure 10.5.

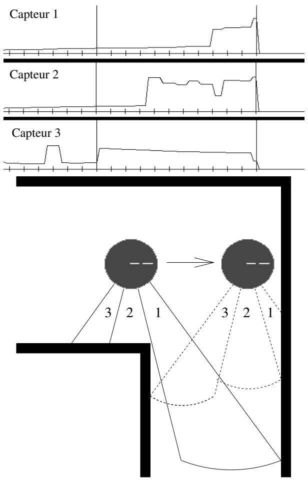


FIG. 10.8 – *Exemple de segmentation d'une fin de couloir sur le côté droit*

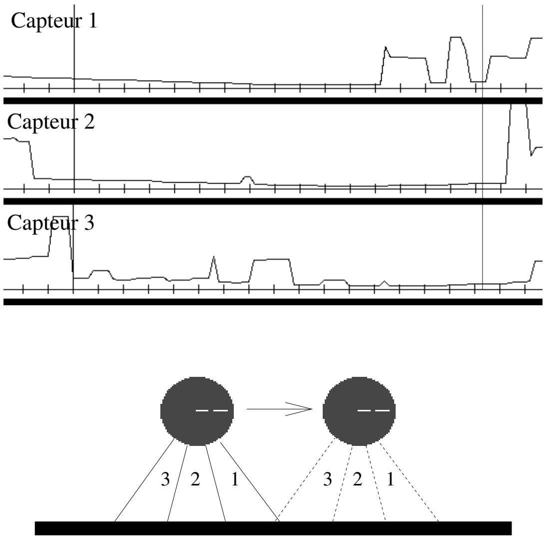


FIG. 10.9 – *La segmentation correspondante à un couloir*

10.3.2 La segmentation et l'étiquetage d'une traversée de couloir

Pour pouvoir segmenter une traversée de couloir, nous avons développé un éditeur de signal. Les règles données dans le paragraphe précédent permettent de définir comment segmenter une traversée de couloir, mais elles ne sont pas suffisantes. Il est impossible en visualisant les variations dans le temps des trois capteurs utilisés pour définir les règles, et les règles de segmentation de savoir à quelle balise correspondent ces variations, ou si elles ne correspondent pas tout simplement à du bruit. Notre outil permet de voir la valeur au cours du temps des six capteurs utilisés pour la segmentation ainsi que la représentation géométrique des valeurs de tous les capteurs au cours du temps. On peut donc savoir, pour chaque variation sur les mesures des capteurs, la représentation géométrique correspondante à ces variations. On peut ainsi facilement déterminer à quelle type de balise correspondent les variations. La figure 10.10 donne un exemple de segmentation

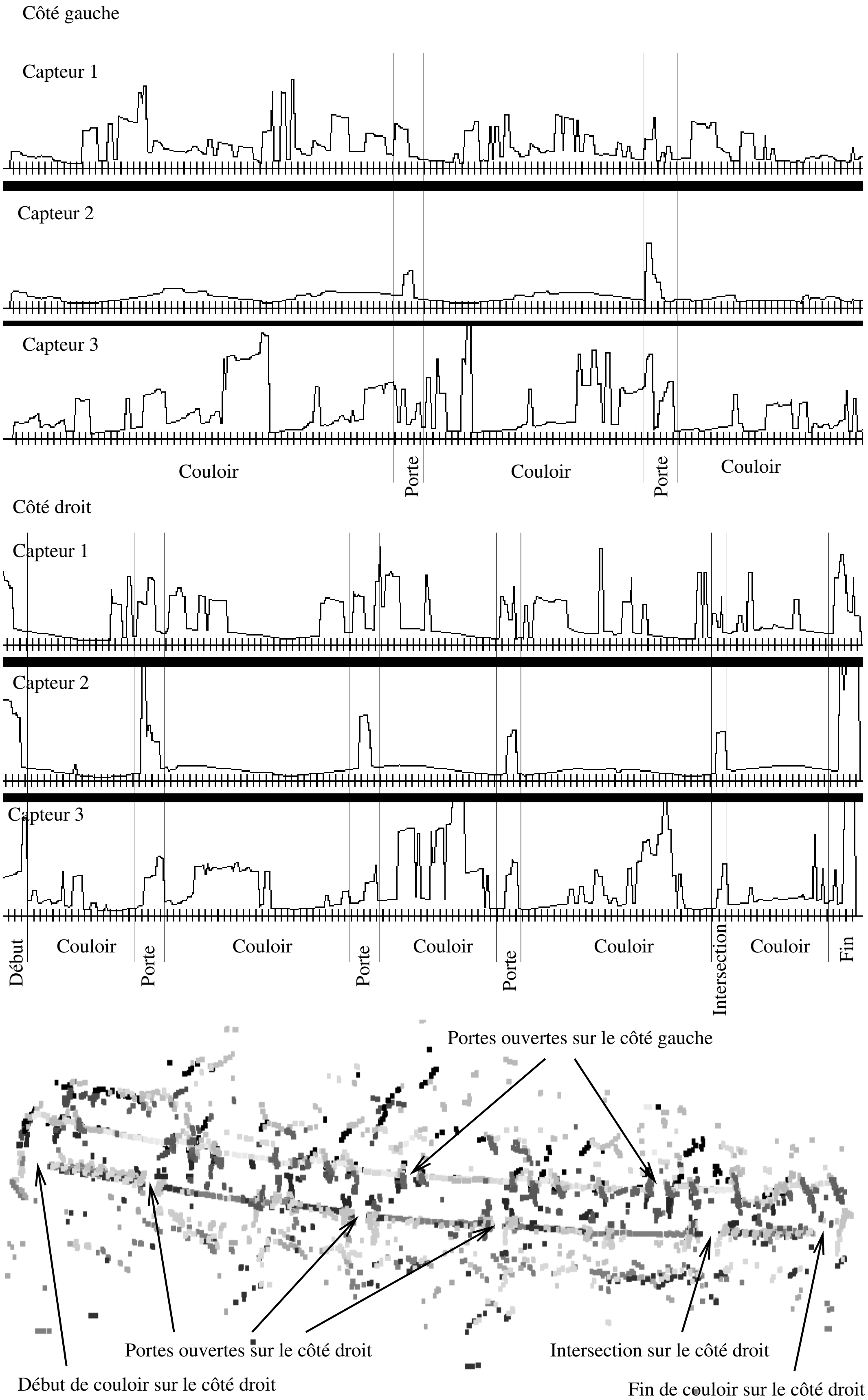


FIG. 10.10 – *La segmentation et l'étiquetage correspondantes à une traversée de couloir*

d'une traversée de couloir, le robot se déplaçant de la gauche vers la droite. Cette figure est composée de trois parties : la première partie représente les variations dans le temps du capteur perpendiculaire au mur à gauche du robot (capteur 2) et des deux capteurs adjacents. La seconde correspond aux variations du capteur dans le temps du capteur perpendiculaire au mur à droite du robot (capteur 2) et des deux capteurs adjacents. Les traits pleins représentent la segmentation des deux traversées, et l'étiquetage correspondant pour chaque côté est donné en dessous des variations des capteurs. La partie en bas de la figure correspond à la représentation géométrique des valeurs de tous les capteurs tout au long du déplacement du robot. Nous y voyons clairement pour chaque balise la correspondance entre les variations des mesures des capteurs et la représentation géométrique.

A la fin de cette phase, à chaque traversée de couloir est associée deux segmentations : une pour le côté droit, et une pour le côté gauche.

10.4 La phase d'apprentissage

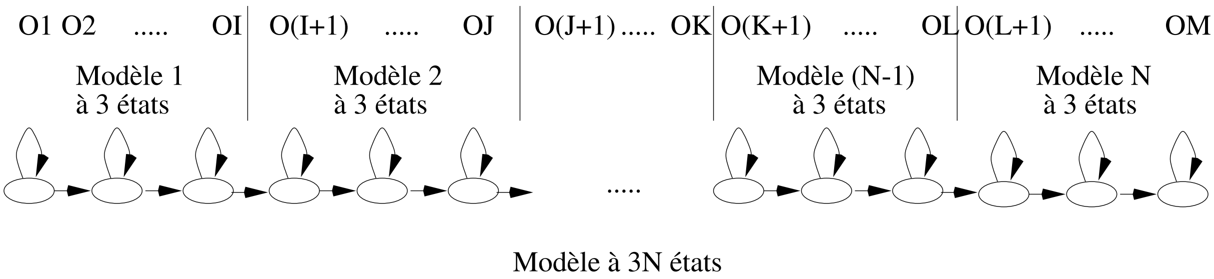


FIG. 10.11 – *La deuxième partie de la phase d'apprentissage*

Pour entraîner un modèle, nous avons besoin du nombre et des valeurs des capteurs qui vont constituer le vecteur d'observation. Comme la segmentation de chaque balise a été faite en utilisant les variations du capteur perpendiculaire au mur du couloir et les variations des deux capteurs adjacents à celui-ci, nous utilisons la dérivée première des mesures de ces trois capteurs comme vecteur d'entrée du modèle. La dérivée première des mesures d'un capteur est calculée en faisant la différence entre la mesure d'un capteur donné à un instant donné et sa mesure à l'instant précédent.

L'apprentissage des modèles est réalisé en deux étapes :

– Dans la première étape, chaque modèle est entraîné indépendamment des autres avec les séquences d'observations qui le caractérisent. Pour cela, pour chaque traversée de couloir, nous extrayons de la segmentation de chaque côté du couloir, les séquences d'observations correspondantes à la balise à apprendre. Lorsque la balise à apprendre a été "vue" sur le côté droit (resp. gauche), la séquence d'observations est constituée des dérivées premières des mesures des trois capteurs de droite (resp. gauche). Cet ensemble de séquences d'observations extraites des segmentations de traversée de couloir constitue le corpus d'apprentissage de la balise. Cette phase permet d'avoir une première estimation de chacun des modèles. Comme nous l'avons dit dans la section 9.3.5, chaque modèle est réestimé plusieurs fois à partir de son corpus d'apprentissage, jusqu'à ce que les matrices de transitions et d'observations convergent. Expérimentalement, nous avons déterminé qu'après quatre réestimations, les matrices de probabilités de transition et d'observations ne sont plus beaucoup modifiées. Nous réestimons donc quatre fois chaque modèle.

- Lors de la phase de reconnaissance, étant donné la séquence d'observations, il ne faut pas reconnaître une balise, mais la suite de balises correspondantes. Pour cela, nous faisons un deuxième apprentissage. Pour chaque traversée de couloir et pour chaque côté, connaissant les N balises que le robot a "vues", nous chaînons les modèles correspondants afin d'obtenir un modèle à 3N états. Nous utilisons ensuite la séquence d'observation correspondante à la traversée de tout le couloir pour réestimer ce modèle à 3N états. Lorsque le modèle à 3N états correspond à la segmentation à droite (resp. gauche), la séquence d'observations est constituée des dérivées premières des mesures des trois capteurs de droite (resp. gauche). La figure 10.11 représente le modèle à 3N états créé à partir d'une séquence de M observations correspondant à N balises. Cet apprentissage a pour but de tenir compte de l'influence des modèles les uns sur les autres, et d'avoir une estimation de la probabilité de transition entre deux modèles consécutifs. Pour la même raison que dans la première étape, nous réestimons chaque suite de modèles quatre fois.

A la fin de cette phase, nous avons cinq modèles de balises entraînés. Ces cinq modèles ont été entraînés pour reconnaître des balises uniquement à partir d'une séquence d'observations indépendamment des capteurs qui constitue la séquence d'observation. Ils seront donc capables de reconnaître les balises seulement à partir des séquences d'observations constituées des variations des mesures de trois capteurs.

10.5 La phase de reconnaissance

L'environnement du robot est décrit au moyen d'une grammaire (figure 10.12) qui permet certaines séquences de modèles et en restreint d'autres. En tenant compte de la grammaire, tous les modèles de Markov cachés sont chaînés dans un grand modèle de Markov caché sur lequel l'algorithme de Viterbi est utilisé. Ensuite, la meilleure séquence de modèles détermine la liste ordonnée de balises que le robot a "vues" pendant son trajet. Il faut prendre en compte le fait que la liste des modèles est connue quand le trajet est terminé. La figure 10.13 montre tous les enchainements de modèles possibles pour une séquence de onze observations. Une séquence de onze observations peut être représentée par un modèle, deux modèles enchainés ou trois modèles enchainés. Chaque ligne correspond à un modèle de balise naturelle particulier contenu dans l'enchainement de modèles. La première ligne donne par exemple tous les enchaînements de modèles contenant au moins un début de couloir. Tous ces modèles seront donc mis en compétition afin de déterminer celui qui a la plus forte probabilité de reconnaître la séquence de onze observations.

Le but de la reconnaissance est d'évaluer les taux de reconnaissance des quatre balises (un début de couloir, une fin de couloir, une porte ouverte et une intersection) dans un couloir. Les couloirs ne sont pas comptabilisés lors de la reconnaissance, puisque nous

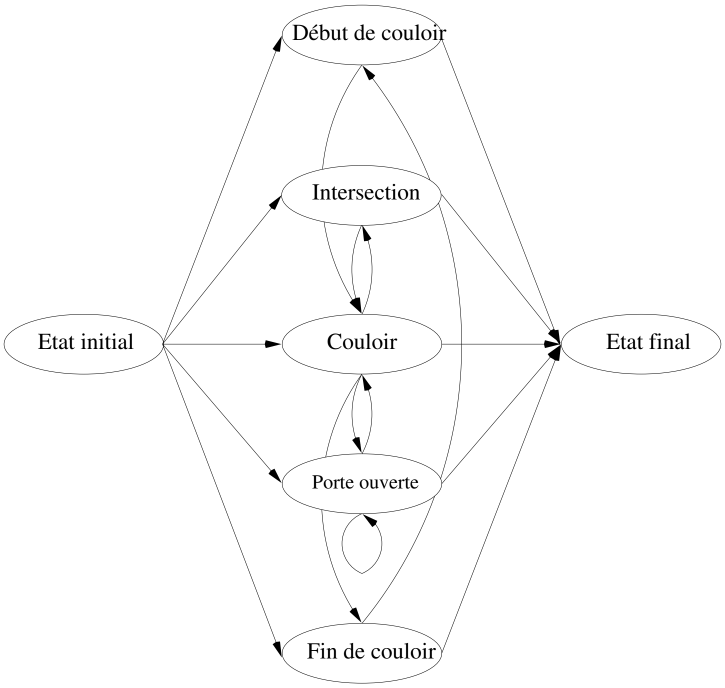


FIG. 10.12 – *La grammaire utilisée pour la reconnaissance*

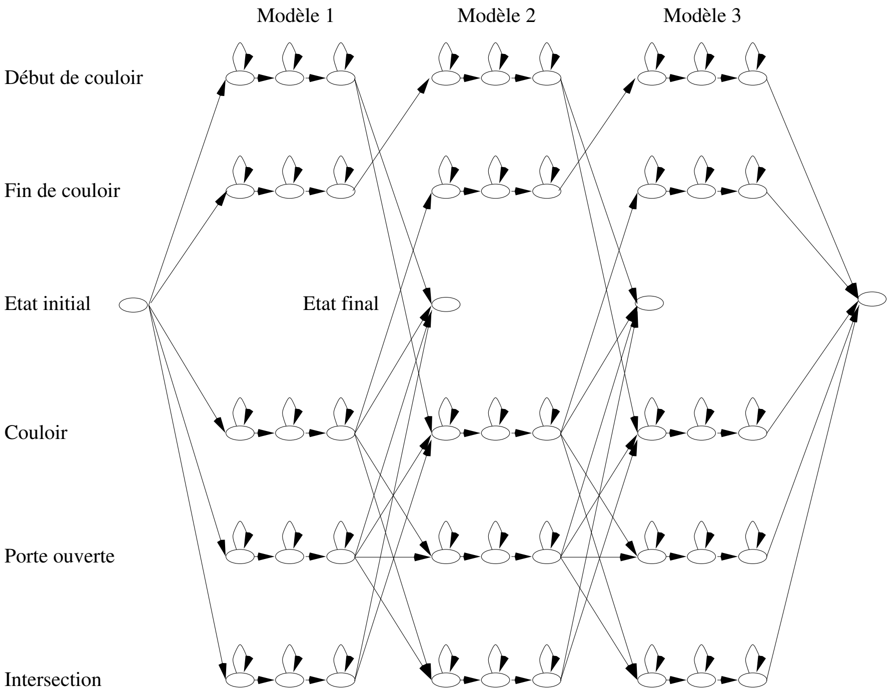


FIG. 10.13 – *Les modèles possibles pour reconnaître une séquence de onze observations*

considérons par défaut que le robot est dans un couloir. Ce modèle ne sert en fait qu'à connecter deux modèles comme le ferait un modèle de silence entre deux mots en reconnaissance de la parole.

Nous réalisons vingt traversées de couloir (dix allers et retours) avec le module de navigation et utilisons les cinq modèles entraînés pour réaliser la reconnaissance. Pour évaluer le taux de reconnaissance, nous comparons pour chaque acquisition, ce que le robot a reconnu, avec ce qu'il aurait dû reconnaître. Pour cela, une fois les traversées de couloir réalisées, nous segmentons et étiquetons manuellement chaque traversée à l'aide de notre éditeur de signal pour déterminer les balises que le robot a “vues”. Puis, nous exécutons l'algorithme de reconnaissance sur chaque traversée afin de déterminer la séquence de balises reconnues par le robot. Enfin nous cherchons à interpréter chacune des balises reconnues par rapport aux balises vues. La reconnaissance est réalisée hors-ligne, puisque le robot fait tout d'abord une traversée de couloir, puis nous segmentons et étiquetons les données acquises, procédons à la reconnaissance de la séquence d'observation correspondante à la traversée de couloir, et comparons ce que la séquence de balises reconnues avec ce que la séquence de balises décrivant la traversée de couloir.

Les critères suivants sont pris en compte pour l'interprétation des balises reconnues :

– Une balise est reconnue si elle a été reconnue par le modèle correspondant.
– Les insertions sont des balises naturelles insérées dans d'autres balises naturelles. Elles correspondent à des balises que le robot a reconnues mais qui en réalité n'existent pas. Par exemple, pendant la reconnaissance d'un couloir (spécialement dans les longs couloirs), quelquefois le robot ne navigue pas complètement parallèlement aux deux murs du couloir, et dans ce cas, les capteurs sont en réflexion, et cette réflexion est interprétée comme une porte ouverte ou une intersection qui en fait n'existe pas. L'interprétation de cette balise est une simple insertion.
– Les omissions sont des balises naturelles que le robot n'a pas reconnues.
– Les substitutions sont des balises naturelles que le robot a confondues avec une autre.

La reconnaissance a été réalisée deux fois pour chaque traversée de couloir, une fois en utilisant les dérivées premières des trois capteurs de droite pour reconnaître les balises du côté droit, et une fois en utilisant les dérivées premières des trois capteurs de gauche pour reconnaître les balises du côté gauche.

La figure 10.14 donne un exemple de reconnaissance effectuée sur une traversée de couloir, ainsi que l'interprétation des balises reconnues. Cette reconnaissance correspond à la reconnaissance des variations des capteurs du côté droit lors de la traversée du couloir de la figure 10.10. Cette figure est composée de deux parties : la première partie présente les variations des capteurs du côté droit dans le temps, et la deuxième partie présente l'interprétation de chaque balise reconnue. Dans la première partie, les traits pleins représentent la segmentation et l'étiquetage correspondant est donné juste en dessous des variations des capteurs. Les traits pointillés représentent les frontières temporelles entre chaque balise déterminées lors de la reconnaissance, et les balises reconnues sont données juste au dessus des variations des capteurs. Nous y voyons clairement la correspondance entre les balises à reconnaître et les balises reconnues. La deuxième partie montre la correspondance entre la séquence de balises à reconnaître et la séquence de balises reconnues,

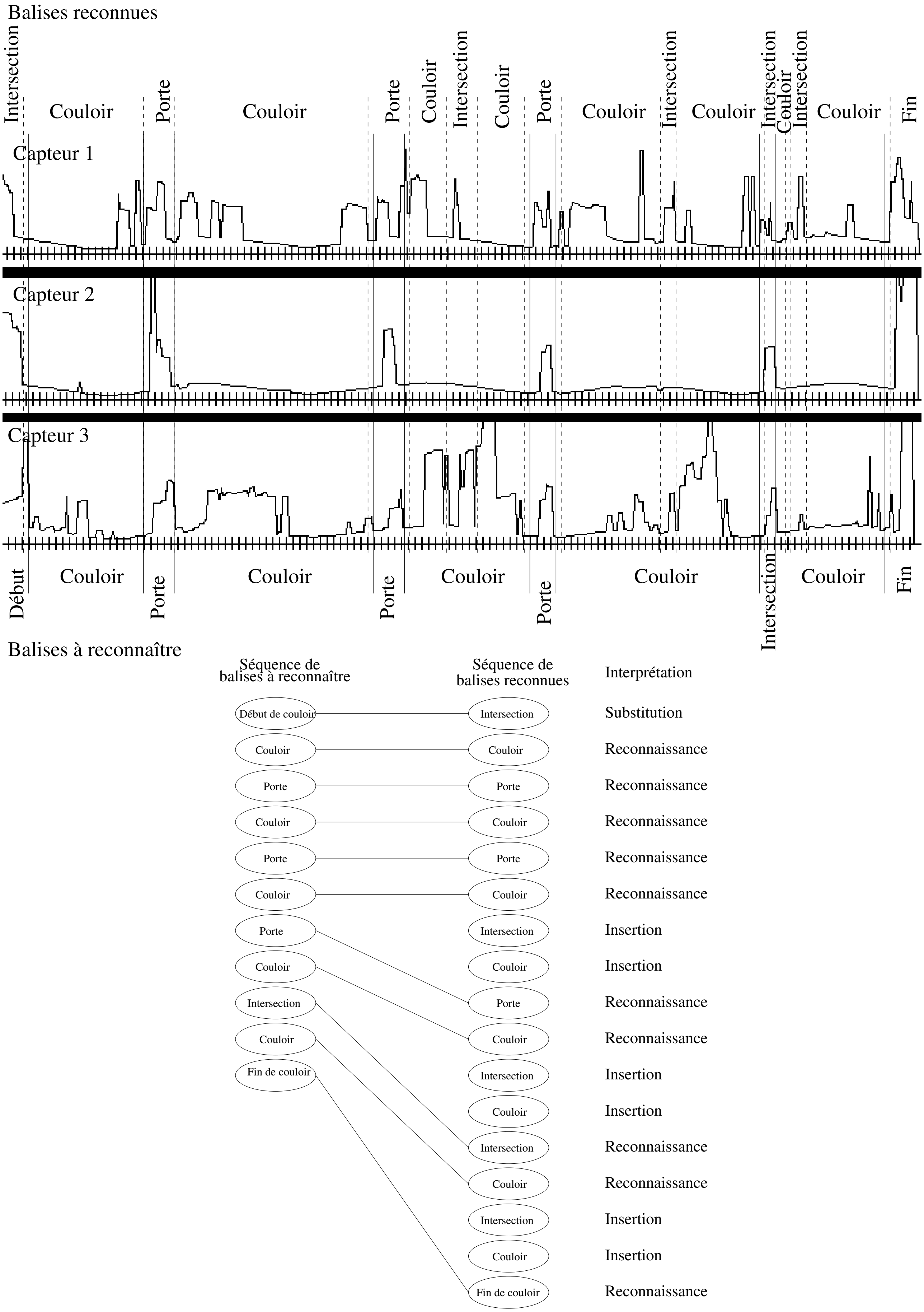


FIG. 10.14 – *L'interprétation des balises reconnues lors d'une traversée de couloir*

et l'interprétation correspondante pour chacune des balises reconnues.

10.6 Résultats et discussion

	début de couloir	fin de couloir	intersection	porte ouver.	Ins.
début de couloir	15	0	0	0	8
fin de couloir	0	19	0	0	17
intersection	1	1	16	38	67
porte ouverte	0	0	2	77	0
oublis	0	0	0	1	0
Total	16	20	18	116	92
% reco.	93	95	89	63	

TAB. 10.1 – *Matrice de confusion*

Les résultats sont présentés dans une matrice de confusion (table 10.1). Chaque ligne correspond à une balise que le robot a reconnue. Chaque colonne correspond à une balise vue pendant le trajet. L'élément c_{ij} à la ligne i et à la colonne j est le nombre de fois que le modèle i a été reconnu quand la bonne réponse était la balise j.

Par exemple, seize débuts de couloir ont été vus sur les vingt trajets. Quinze ont été reconnus comme des débuts de couloir, et une comme une intersection (une substitution). Soit 93% des débuts de couloir ont été reconnus. La dernière colonne de la table 10.1 correspond aux insertions. Par exemple, les intersections ont été insérées soixante sept fois, c'est à dire que sur l'ensemble des vingt traversées utilisées pour la reconnaissance l'algorithme de Viterbi a reconnu soixante sept intersections qui n'existent pas. Comme nous l'avons déjà écrit ces insertions sont provoquées par des réflexions sur les capteurs constituant l'observation.

Les débuts et les fins de couloir sont très bien reconnus. Ces balises ont une forme très particulière, et peuvent difficilement être confondus avec une autre balise.

Les intersections sont globalement bien reconnues, bien que quelquefois confondues avec les portes ouvertes.

Les portes ouvertes sont assez souvent confondues (une fois sur trois) avec les intersections. Ceci pour plusieurs raisons :

- La forme du signal lorsque le robot passe devant une porte est très proche de la forme du signal lorsque le robot passe devant une intersection. De plus, les modèles de Markov n'ayant pas un grand pouvoir de discrimination, il leur est difficile de faire la différence entre les deux modèles.
- Au vu du signal, le seul critère qui permet de distinguer une porte d'une intersection est la taille. En effet, une porte a une taille d'environ quatre-vingt dix cms, et l'intersection une taille de cent vingt cms. La différence de taille est assez faible, et

comme le robot navigue avec une vitesse variable dû au fonctionnement du module de navigation et dépendante de l'encombrement de l'environnement, le critère de taille entre les deux modèles n'est que très peu discriminant. Il est évident que si la taille de l'intersection était plus grande, il y aurait moins de confusion.

– De plus, nous n'avons qu'une seule intersection, mais de nombreuses portes toutes différentes les unes des autres : nous avons travaillé dans des conditions réelles, et les portes étaient soit complètement ouvertes, entreouvertes ou quasiment fermées. Le modèle de porte est donc un modèle qui contient des signaux qui sont très variables, ce qui ne peut qu'accroître le risque de confusion avec une intersection.

La plupart des insertions sont dues au fonctionnement du module de navigation et aux obstacles imprévus. Parfois, le robot mobile doit éviter les personnes ou les obstacles. Dans ce cas, il ne navigue plus parallèlement aux deux murs et au milieu du couloir. Ces conditions provoquent des réflexions correspondantes à de fortes variations sur les trois capteurs qui sont interprétées comme des balises (en général des intersections).

	nombre	%
Balises vues	170	100
Reconnues	127	75
Substituées	42	24
Oubliées	1	1
Inserées	92	54

TAB. 10.2 – *Reconnaissance globale*

Dans la table 10.2, nous présentons les résultats globaux. C'est à dire le nombre et le pourcentage de balises vues, reconnues, substituées, oubliées et insérées.

Les balises sont globalement bien reconnues (cf. Table 10.2) (plus de 85% de reconnaissance, pour chaque balise, à part les portes ouvertes), et 75% de reconnaissance globale. Le problème majeur est le pourcentage d'insertions (54 %) de balises. Les omissions sont très faibles (moins d'1 %).

Remarque

Le nombre total de début de couloir et d'intersection est inférieur à vingt (c-à-d, une par traversée pour dix allers et retours) dans le corpus utilisé pour faire la reconnaissance, car pour réaliser un corpus un peu plus représentatif, nous avons fait des traversées qui n'allaient pas toujours d'une extrémité à l'autre du couloir.

10.6.1 Conclusion

De cette première étude, nous pouvons tirer quelques conclusions :

– Le taux de reconnaissance est bon. Le facteur qui limite le plus ce taux est la confusion fréquente entre les portes ouvertes et les intersections.

- Plus d'une balise reconnue sur trois est une insertion, ce qui est beaucoup trop élevé pour envisager d'utiliser notre approche pour reconnaître des balises naturelles, sans pouvoir détecter celle qui correspondent à des insertions.
- Notre approche remplit les différents critères définis dans le chapitre précédent pour apprendre et reconnaître des balises naturelles. Simplement nous n'utilisons pas les variations de tous les capteurs pour constituer l'observation.

Partant de ces conclusions, dans la section suivante, nous modifions la façon d'apprendre et de reconnaître les balises naturelles de notre environnement afin de diminuer le taux de confusion entre les portes ouvertes et les intersections, et le taux d'insertion.

10.7 Améliorations du taux de reconnaissance

10.7.1 Introduction

Pour augmenter le taux de reconnaissance des balises, nous avons deux problèmes à résoudre :

- Réduire le taux de confusion entre les portes ouvertes et les intersections. Pour cela, nous devons trouver un moyen d'augmenter la discrimination entre ces deux types de balises. La différence de longueur entre ces deux types de balises en est un bon, mais il n'est pas vraiment pris en compte, nous devons donc trouver un moyen pour prendre en compte la longueur des balises pendant l'apprentissage et la reconnaissance. Actuellement, les observations sont faites à chaque fois que le robot s'est déplacé. De part le fonctionnement du module de navigation, le robot a une vitesse variable qui dépend de l'encombrement de son environnement local, donc la distance parcourue entre deux acquisitions n'est pas constante. Par exemple, nous avons remarqué que lorsque le robot passe devant une intersection il parcourt environ 1,80 mètres entre le moment où la valeur du premier capteur augmente brusquement et le moment où la valeur du troisième capteur diminue brusquement. Si l'on regarde l'ensemble des séquences d'observations correspondantes à cette intersection, on se rend compte que le nombre d'observations les composant peut aller de vingt cinq observations à cent vingt observations. Ce qui veut dire que pour la même intersection, le robot peut selon l'encombrement de son environnement local faire d'une acquisition tous les 1,5 cms jusqu'à une acquisition tous les 7,5 cms. Il est donc évident que la longueur de l'intersection ou d'une balise en général ne peut être prise en compte lors de l'apprentissage des modèles puisqu'elle varie beaucoup.
  Pour pouvoir utiliser un critère de longueur, il est donc nécessaire qu'entre deux acquisitions la distance parcourue par le robot soit toujours la même. Sur notre robot mobile, il n'est pas possible de programmer automatiquement une acquisition en fonction de la distance parcourue. Par contre, il nous est tout à fait possible de continuer à acquérir les mesures des capteurs de la même façon que dans la section 10.3 et de les modifier avant les phases d'apprentissage et de reconnaissance pour simuler que la distance parcourue entre deux acquisitions est toujours la même.
  Notre idée est de transformer les observations, pour simuler le fait que le robot se déplace toujours de la même distance entre deux acquisitions. Ceci aidera les modèles

de Markov cachés à faire la discrimination entre une intersection et une porte ouverte en utilisant le critère de longueur d'une balise.

- Réduire le taux d'insertions. En ce moment, les phases d'apprentissage et de reconnaissance sont réalisées indépendamment de chaque côté en utilisant seulement trois capteurs de chaque côté (six capteurs en tout). Cette technique a deux inconvénients :
  - Nous perdons des informations sur l'environnement en utilisant seulement six capteurs (deux paires de trois capteurs). Notre robot mobile a seize capteurs. Les informations fournies par les capteurs situés à l'avant et à l'arrière ne sont actuellement pas prises en compte. Nous avons par exemple des insertions de début et de fin de couloir au milieu du couloir, ce qui n'a pas de sens. La prise en compte des capteurs de devant pourraient éviter ce genre d'insertions. En effet, au début (resp. à la fin) d'un couloir, les valeurs des capteurs situés à l'arrière augmentent (resp. diminuent) progressivement, ce qui ne peut que très rarement arriver lorsque le robot n'est pas au début (resp. à la fin) d'un couloir.
  - Les phases d'apprentissage et de reconnaissance sont réalisées indépendamment de chaque côté. Ceci ne permet pas d'apprendre et de reconnaître l'environnement d'une manière totale (càd, en observant à chaque instant la totalité de la balise). Un apprentissage et une reconnaissance plus totale pourrait être intéressants pour de nombreuses raisons. Par exemple, l'influence de la reconnaissance d'un côté sur l'autre côté pourrait être prise en compte, pour éliminer certaines insertions absurdes.
  - Les apprentissages et les reconnaissances des cinq modèles sont faites en utilisant indifféremment les capteurs de droite ou les capteurs de gauche. Pourtant, il n'est absolument pas évident que les capteurs de droite "voit" une balise de la même façon que les capteurs de gauche, et que le fait de construire le même modèle de chaque balise pour les capteurs de droite et pour les capteurs de gauche soit la meilleure solution.

  Notre idée est de prendre en compte les observations des seize capteurs sonars.

Les deux prochaines sections présentent les deux modifications majeures que nous avons réalisées pour augmenter le taux de reconnaissance global : simuler une fréquence d'acquisition constante des données fonction de la distance parcourue, et utiliser les variations des seize capteurs. Nous présentons ensuite les résultats que nous avons obtenus.

10.7.2 Transformation des observations

Nous avons choisi de fixer la distance parcourue entre deux acquisitions des mesures des capteurs. Pour cela, nous avons analysé quelle était la distance parcourue entre deux observations des mesures des capteurs en supposant que l'erreur sur l'estimée de la position odométrique du robot entre un instant donné et l'instant suivant était négligeable. On se rend compte que :

- Dans une zone encombrée, le robot effectue un déplacement d'environ 1,5 cms entre deux acquisitions.

– Dans une zone libre, le robot effectue un déplacement d'environ 7,5 cms entre deux acquisitions.

Si nous choisissons de simuler (par interpolation des données réelles) que les acquisitions sont faites tous les 7,5 cms au moins, on est sûr d'avoir toujours au moins une acquisition réelle du robot dans l'intervalle de distance choisie et donc des mesures différentes à chaque acquisition simulées. L'algorithme 10.1 que nous avons utilisé est très simple. L'idée principale pour simuler une acquisition pour chaque capteur, est de faire une moyenne des valeurs de chaque capteur chaque 7,5 cms parcourue sur les acquisitions réelles faites pendant ce déplacement. Nous simulons aussi que le robot parcourt exactement 7,5 cms entre deux acquisitions simulées. Pour cela, nous calculons la distance que le robot a parcourue depuis sa dernière acquisition simulée, et nous calculons la position où le robot est supposé se trouver après un déplacement d'exactement 7,5 cms.

`Transformation (O, X, Y, G) :`

entrée : $O_1, O_2, \ldots, O_T$ la séquence de T observations.
$(X_1,Y_1), \ldots, (X_T,Y_T)$ l'estimée de la position du robot associée à l'observation.

sortie : $G_1, G_2, \ldots, G_K$ la séquence de K observations acquises tous les 7,5 cms.

I $\leftarrow$1 (I est l'indice d'observation initiale)
x $\leftarrow X_1$ (x est l'abscisse du robot lors de la dernière observation)
y $\leftarrow Y_1$ (y est l'ordonnée du robot lors de la dernière observation)
K $\leftarrow$1 (K est l'indice d'observation après transformation)
tant que $I \leq T$ **faire**
1) $Observations \leftarrow 0$ (Nombre d'observations réelles entre deux observations simulées)
2) $G_K \leftarrow 0$
3)**tant que** la distance entre (x, y) et (X_I, Y_I) est inférieure à 7,5 cms.
1) $G_K \leftarrow G_K + O_I$
2) $I \leftarrow I + 1$
3) $Observations \leftarrow Observations + 1$
fin-tant-que
4) $G_K \leftarrow G_K \div Observations$
5) $K \leftarrow K + 1$
6) $x \leftarrow x + (X_I - x) \times 7{,}5 \div$ distance entre $(x,y) et (X_I,Y_I)$
7) $y \leftarrow y + (Y_I - y) \times 7{,}5 \div$ distance entre $(x,y) et (X_I,Y_I)$
fin-tant-que

Algorithme 10.1: Algorithme de Transformation

Étape n° 1 et 2 : on initialise le nombre d'observations et la valeur de la nouvelle observation simulée.

Étape n° 3 : on stocke toutes les observations tant que le robot n'a pas parcouru 7,5 cms depuis sa dernière observation simulée.

Étapes n° 4, 5, 6 et 7 : on met à jour l'observation géométrique courante ainsi que l'estimée de la position du robot correspondante.

10.7.3 Utilisation des variations des mesures des seize capteurs comme entrée des modèles de balises

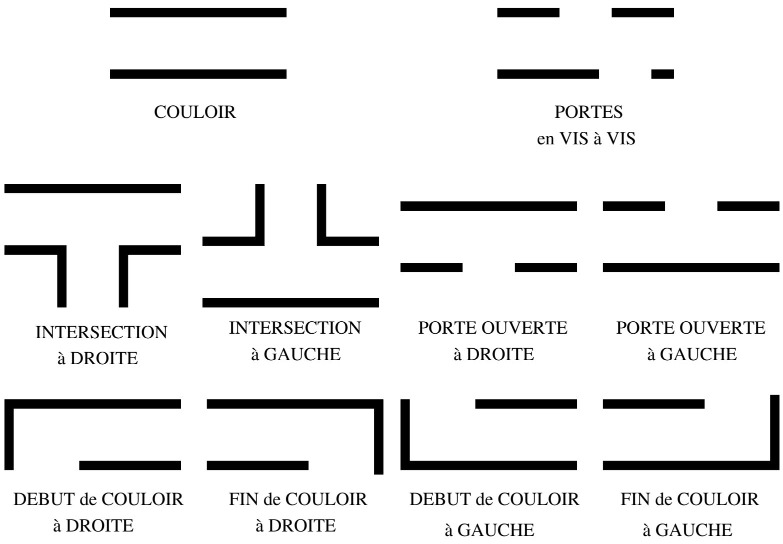


FIG. 10.15 – *Les dix nouvelles balises*

Nous avons choisi d'utiliser les seize capteurs au lieu des deux couples de trois pour construire les modèles de Markov de chaque balise. L'utilisation des seize capteurs permet une prise en compte totale de la forme de chaque balise. Par exemple, il est évident que lorsqu'on construit un modèle de début de couloir et fin de couloir, les capteurs avant du robot ont une importance.

Mais, les balises précédemment définies (figure 10.1) ne sont plus utilisables. Maintenant, comme nous considérons l'environnement d'une façon plus générale, cela n'a pas de sens de considérer que cinq balises peuvent être vues de chaque côté. Nous devons définir des balises plus générales qui prennent en compte ce qui peut être vu du côté droit et ce qui peut être vu du côté gauche. Il faut en fait combiner ce que le robot est susceptible de reconnaître sur sa droite avec ce qu'il est capable de reconnaître sur sa gauche. Ces combinaisons nous donnent donc un ensemble de vingt cinq nouvelles balises possibles. Comme nous ne considérons pas la combinaison de portes ouvertes avec des intersections, des débuts de couloir, ou des fins de couloir car ce genre de combinaisons n'apparaît que très rarement dans notre environnement, nous définissons dix nouvelles balises (figure 10.15) :

- Un couloir, lorsqu'il y a au même endroit un mur vu par les trois capteurs de gauche et un mur vu par les trois capteurs de droite.
- Une intersection à droite (resp. gauche), lorsqu'il y a au même endroit un mur vu par les trois capteurs de gauche (resp. droite) et une intersection vue par les trois capteurs de droite (resp. gauche).
- Une porte à droite (resp. gauche), lorsqu'il y a au même endroit un mur vu par les trois capteurs de gauche (resp. droite) et une porte vue par les trois capteurs de droite (resp. gauche).

– Un début de couloir à droite (resp. gauche), lorsqu'il y a au même endroit un mur vu par les trois capteurs de gauche (resp. droite) et un début de couloir vu par les trois capteurs de droite (resp. gauche).
– Une fin de couloir à droite (resp. gauche), lorsqu'il y a au même endroit un mur vu par les trois capteurs de gauche (resp. droite) et une fin de couloir vu par les trois capteurs de droite (resp. gauche).
– Deux portes en vis à vis, lorsqu'il y a au même endroit une porte vue par les trois capteurs de gauche et une porte vue par les trois capteurs de droite.

10.7.4 La segmentation, l'apprentissage et la reconnaissance

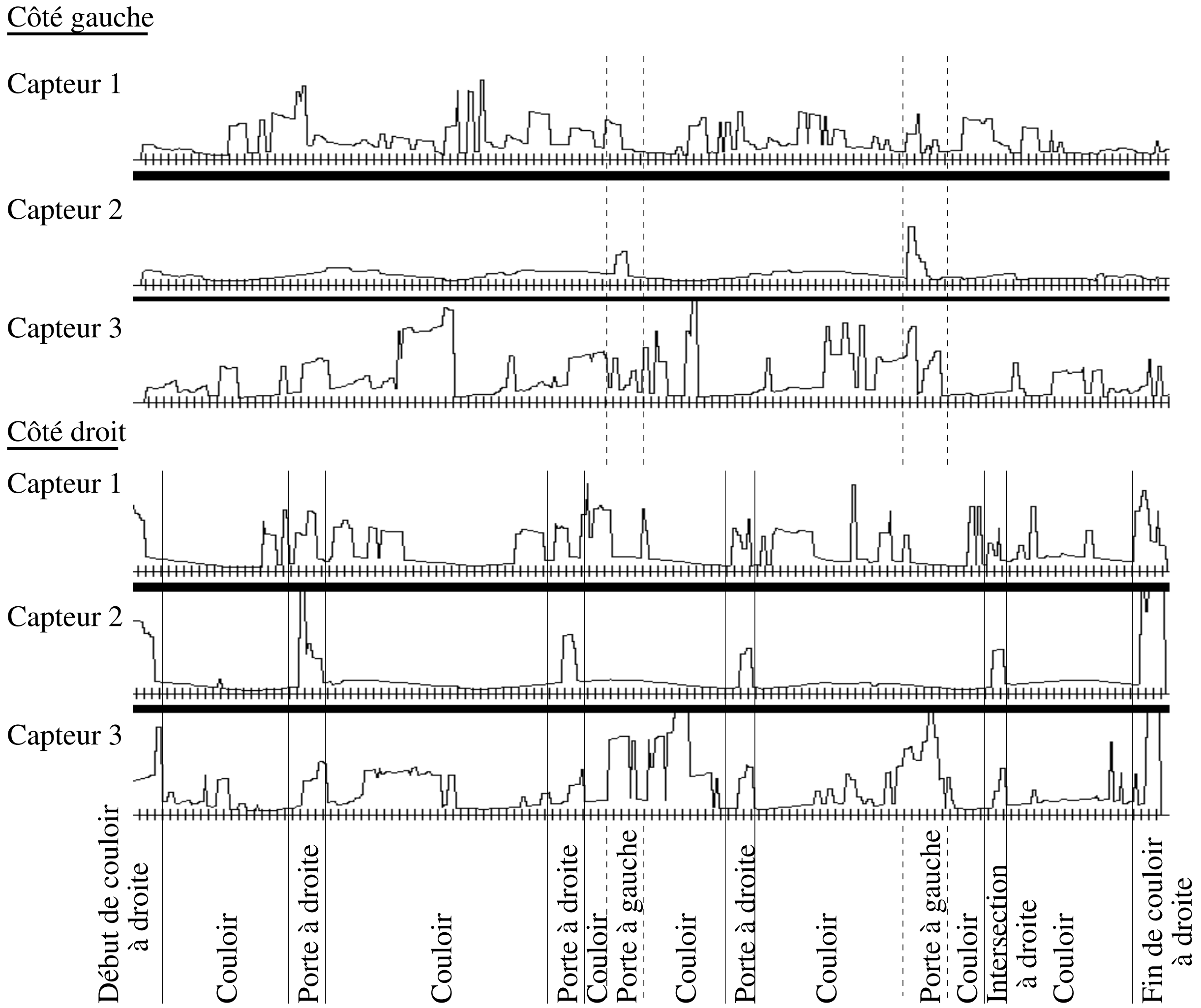


FIG. 10.16 – *La segmentation correspondante à une traversée de couloir*

Pour réaliser les corpus d'apprentissage, nous utilisons les mêmes traversées de couloir que dans la section 10.3. Comme nous considérons l'environnement d'une manière totale, nous n'avons plus deux segmentations (une de chaque côté), mais une seule segmentation faite en utilisant les définitions des dix nouvelles balises, et les deux mêmes couples de trois capteurs. Plutôt que de resegmenter et réetiqueter tout le corpus d'apprentissage, nous avons développé un outil qui à partir des segmentations et des étiquetages précédents d'une traversée de couloir et des nouvelles règles de segmentation donne la nouvelle segmentation et l'étiquetage correspondant. La figure 10.16 montre la nouvelle segmentation de la traversée de couloir de la figure 10.10. La partie supérieure correspond à la segmentation précédente des variations des capteurs gauches du robot. Cette segmentation est matérialisée par des traits pointillés. La partie inférieure correspond à la segmentation précédente des variations des capteurs droits du robot. Cette segmentation est matérialisée par des traits pleins. Tout en bas de la figure, on trouve la nouvelle segmentation ainsi que le nouvel étiquetage.

Nous réalisons la phase d'apprentissage des dix nouvelles balises en deux étapes en réestimant quatre fois chaque balise pour chaque étape de la même façon que dans la section 10.4, et en utilisant les seize dérivées premières des interpolations des seize mesures des capteurs comme entrée de chaque modèle.

La phase de reconnaissance est réalisée en utilisant le même corpus de test que précédemment (ie. les vingt traversées de couloirs utilisés dans la section 10.5 pour évaluer le taux de reconnaissance). Le but de la reconnaissance est d'évaluer le taux de reconnaissance des neuf balises (début à droite, fin à droite, intersection à droite, porte ouverte à droite, début à gauche, fin à gauche, intersection à gauche, porte ouverte à gauche, et portes en vis à vis) dans le couloir.

10.7.5 Nouveaux résultats

Les résultats sont donnés sous forme d'une matrice de confusion (table 10.3).

	début droit	fin droit	intersection droit	porte droit	début gauche	fin gauche	intersection gauche	porte gauche	porte porte	Ins.
début droit	7	0	0	0	0	0	0	0	0	0
fin droit	0	7	0	0	0	1	0	1	0	0
inter. droit	0	0	7	0	0	0	0	0	0	0
porte droit	0	0	0	42	0	0	0	1	1	25
début gauche	0	0	0	0	8	0	0	0	0	0
fin gauche	0	0	0	0	0	6	0	0	0	0
inter. gauche	0	0	0	0	0	0	8	0	0	0
porte gauche	1	0	0	4	0	0	1	43	1	34
porte porte	0	0	1	0	0	0	0	1	2	1
oublis	0	1	0	0	0	0	0	0	0	0
Total	8	8	8	46	8	7	9	46	4	
% reco.	88	88	88	91	100	86	89	93	50	

TAB. 10.3 – *Matrice de confusion des balises reconnues en fonction des balises vues*

On constate que le taux de confusion entre les portes ouvertes (à droite ou à gauche) et les intersections (à droite ou à gauche) a nettement diminué. En effet, du fait de la transformation des observations, la forme du signal et le nombre d'observations pour les portes ouvertes et pour les intersections est presque toujours compris dans un faible intervalle quelque soit l'encombrement de l'environnement et la vitesse du robot. Comme la forme du signal et le nombre d'observations pour les portes ouvertes est différent de la forme du signal et du nombre d'observations pour les intersections, la discrimination est plus facile entre ces deux types de balises.

Auparavant, nous avions un nombre important d'insertions d'intersections. Nous n'avons actuellement plus d'insertions d'intersection, mais un nombre plus élevé d'insertions de portes ouvertes. En effet, une intersection a une taille plus grande qu'une porte ouverte, et les réflexions sur les capteurs étant de petite taille, elles ne peuvent être reconnues que comme des portes ouvertes. D'autre part, nous avions soixante sept insertions d'intersections dues aux réflexions et pas d'insertions de portes ouvertes dues aux réflexions sans transformation des observations, contre cinquante neuf insertions de portes ouvertes et pas d'insertion d'intersection actuellement. On constate qu'il y a une diminution du nombre d'insertions dûes aux réflexions. Suite à la transformation des observations, les réflexions de petite taille ne peuvent plus être reconnues comme des portes ouvertes ou des intersections, et ne provoquent plus d'insertions.

La prise en compte globale de l'environnement grâce à l'utilisation des seize capteurs lors de l'apprentissage a permis d'éliminer complètement les insertions de début et de fin de couloir lorsque le robot n'est pas à une extrémité du couloir. Les observations des capteurs situés à l'avant (resp. à l'arrière) du robot sont très différentes selon que le robot se trouve au milieu d'un couloir ou à la fin (resp. au début) d'un couloir. Les deux modèles de début (resp. fin) de couloir ne pourront être reconnus que lorsque les variations des capteurs situés à l'arrière (resp. avant) du robot correspondront à un début (resp. fin) de couloir, c'est à dire qu'elles diminueront progressivement (resp. diminueront progressivement) ce qui n'arrivera jamais au milieu d'un couloir. Il ne pourra donc jamais y avoir d'insertions de début (resp. fin) de couloir au milieu d'un couloir.

Le taux de reconnaissance des portes en vis à vis est très moyen (50%) pour plusieurs raisons :

- L'apprentissage et la reconnaissance ont été réalisées sur de petits corpus, car cette balise n'apparaît pas très souvent dans notre environnement.
- Nous ne faisons pas la différence quand la première porte ouverte est du côté gauche et la seconde du côté droit ou le contraire.
- Nous ne prenons pas en compte si les portes se chevauchent partiellement ou quasiment complètement.
- Comme nous l'avons dit précédemment, nous travaillons dans un environnement réaliste avec des portes différentes les unes des autres. De plus, certaines portes sont complètement ouvertes et d'autres partiellement. Le modèle prend en compte cette diversité.
- Il arrive aussi lorsque les portes ne se chevauchent que très peu, qu'au lieu de reconnaître des portes en vis à vis, on reconnaisse tout d'abord une porte d'une côté puis une porte de l'autre côté.

Toutes ces raisons font que le modèle de cette balise est un modèle appris avec peu d'observations qui sont très différentes les unes des autres. Il n'est donc pas entraîné pour reconnaître un modèle de balise bien spécifique.

Dans la table 10.4, nous présentons les résultats globaux. C'est à dire le nombre et le pourcentage de balises vues, reconnues, substituées, oubliées et insérées.

	nombre	%
Balises vues	144	100
Reconnues	130	90
Substituées	11	9
Oubliées	2	1
Inserées	60	42

TAB. 10.4 – *Reconnaissance globale*

Suite aux modifications, le taux de reconnaissance globale passe de 75% à 90%. Les insertions de balises diminue de 54% à 42%. Les omissions restent très faibles (moins d'1 %).

10.7.6 Conclusion

Les deux modifications que nous avons apportées ont permis au taux de reconnaissance de passer de 75% à 90% et de faire diminuer le taux d'insertion de 54% à 42%. Le taux de reconnaissance est très bon, et le taux d'insertion reste encore un peu élevé. Le choix de modèles de portes plus spécifique devrait sans doute permettre de distinguer les différentes portes de notre environnement et en même temps de les distinguer des réflexions.

De plus, notre approche vérifie les critères définis pour un bon apprentissage et une bonne reconnaissance de balises : prise en compte de la totalité de la balise (càd, utilisation des variations de tous les capteurs pour observer la balise), prise en compte de la globalité de la balise (càd, utilisation d'une séquence d'observations pour observer la balise), et interprétation symbolique des séquences d'observations.

10.8 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle méthode pour apprendre et reconnaître des balises dans un environnement intérieur structuré avec des modèles de Markov cachés. Nous avons dans une première étude construit un modèle de cinq balises à partir d'observations réelles, et évaluer le taux de reconnaissance global de ces cinq modèles. Cette première étude donne de bons résultats (75% de reconnaissance globale), et l'analyse de ces résultats permet de comprendre quels sont les facteurs limitants le taux de reconnaissance. Dans un deuxième temps, nous avons donc tenu compte des premiers résultats pour modifier la façon d'apprendre et de reconnaître une balise. Ces modifications portent sur deux critères : le nombre de capteurs à prendre en compte lors des phases d'apprentissage et de reconnaissance, et la simulation d'une fréquence constante d'acquisitions des mesures des capteurs fonction de la distance parcourue par le robot. Le taux de reconnaissance final obtenu est très bon (90%). Les résultats pourraient être améliorés en ajoutant plus de modèles pour prendre en compte différentes classes de la même balise, en particulier différents classes de portes ouvertes ou portes en vis à vis.

Un modèle de Markov cherchant toujours à se rapprocher de la situation la plus proche qu'il connaît, il sera capable de reconnaître les balises qu'il a apprises mêmes si elles ne correspondent pas parfaitement au modèle qu'il en a construits. Cette propriété des modèles de Markov permet de reconnaître des balises même si elles ne sont pas parfaitement observables, ou si des obstacles perturbent la reconnaissance. On constate aussi que les modèles de balises que nous avons construits ont réussi à extraire des caractéristiques beaucoup plus complexe que les simples variations d'un ensemble de capteurs entre deux instants. La longueur d'une séquence d'observations (càd, le nombre d'observations la composant) a été prise en compte pour faire la différence entre une intersection et une porte ouverte. La diminution (resp. l'augmentation) progressive des valeurs des capteurs avant (resp. arrière) au cours du temps a été utilisée pour caractériser une fin (resp. un début) de couloir. D'autre part, l'utilisation des dérivées premières permet de prendre en compte uniquement les variations des capteurs pour apprendre et reconnaître des balises, ce qui fait que nous pouvons utiliser nos modèles de balises dans des couloirs de largeurs différentes sans que les reconnaissances en soient affectées.

On peut donc imaginer d'étendre notre approche en augmentant le nombre de balises afin d'obtenir des balises encore plus spécifiques que celles que nous avons à l'heure actuelle ou de rajouter des balises dont nous ne tenons pas compte à l'heure actuelle. Les balises "portes ouvertes" ou "portes en vis en vis" peuvent correspondre à plusieurs balises plus spécifiques, où l'on tient compte du sens d'ouverture de la porte, et du fait qu'elles sont complètement ou partiellement ouvertes, et pour les portes en vis à vis, du fait qu'elles se chevauchent partiellement ou complètement, ou du fait que la porte située à droite doit être reconnue avant celle située à gauche ou inversement. On peut aussi ajouter un ou plusieurs modèles d'obstacles en tenant compte du sens d'évitement, ou même un modèle de rétrécissement de couloirs.

De manière plus générale, on peut envisager d'utiliser notre approche pour apprendre et reconnaître des balises que l'on trouve dans d'autres types d'environnements intérieurs tels que par exemple une pièce ou un hall[26]. On peut par exemple définir :

- un modèle de passage d'une porte lorsque le robot entre dans une pièce ou en sort, caractérisé par tout d'abord une importante diminution des mesures des trois capteurs de chaque côté du robot (correspondant au moment où le robot se trouve devant la porte), puis pas ou peu de variations sur la mesure de ces six capteurs (correspondant au moment où le robot est en train de franchir la porte) et pour finir par une importante augmentation des mesures des capteurs (correspondant au moment où le robot vient de passer la porte).
- un modèle de coin à droite (resp. gauche), lorsque le robot se trouve dans un coin d'une pièce, caractérisé par une diminution progressive des capteurs de droite (resp. gauche) et des capteurs à l'avant.
- un modèle de rapprochement vers le mur situé à droite (resp. gauche) du robot, caractérisé par une diminution progressive des capteurs droits (resp. gauches) du robot.

26. Nous définissons un hall comme une pièce de grande dimension.

- un modèle d'éloignement d'un mur situé à droite (resp. gauche) du robot, caractérisé par une augmentation progressive des capteurs droits (resp. gauche) du robot.
- un modèle de rapprochement vers le mur situé face au robot, caractérisé par une diminution progressive des capteurs situés à l'avant du robot, et pas ou peu de variation des capteurs situés sur les côtés du robot.

Le nombre de balises que l'on peut apprendre et reconnaître ne dépend que de la caractérisation que l'on en fait. Nous venons de montrer qu'il était possible de caractériser de nombreuses balises dans plusieurs types d'environnements simplement en utilisant les variations (brusques ou progressives) de certains capteurs. Nous savons que les variations d'un ensemble de capteurs sont tout à fait apprenables par un modèle de Markov, puisque nous les avons utilisés pour apprendre et reconnaître des balises naturelles. Nous avons montré dans ce chapitre que d'autres critères tels que la longueur d'une balise, la forme du signal d'une balise peuvent être utilisés. Ces divers critères, ainsi que d'autres, apprenables et reconnaissables par un modèle de Markov permettent donc d'apprendre et de reconnaître de nombreuses balises dans de nombreux environnements.

Nous avons vu que lorsque le nombre de balises augmente, cela n'a aucune influence sur le taux de reconnaissance, puisqu'en passant de cinq à dix balises notre taux de reconnaissance est passée de 75% à 90%. Donc même si le nombre de balises devient important, il sera toujours possible de reconnaître correctement les balises.

On peut aussi utiliser des informations supplémentaires afin d'apprendre et de reconnaître une balise en fonction de l'environnement dans lequel le robot se trouve. On sait que dans un couloir, les valeurs des capteurs qui n'interviennent pas dans la caractérisation des balises sont faibles (généralement inférieur à un mètre). Dans une pièce, les valeurs des capteurs qui n'interviennent pas dans la caractérisation des balises sont moyennes (généralement comprises entre un mètre et trois mètres). Dans un hall, les valeurs des capteurs qui n'interviennent pas dans la caractérisation des balises sont élevées (généralement comprises entre trois mètres et la limite de portée). Si on ajoute aux variations des capteurs, les valeurs des capteurs comme composante de l'observation, on peut déterminer pour chaque balise reconnue le type d'environnement associé. Par exemple, on caractérise une balise pour un couloir (resp. une pièce, resp. un hall) grâce aux variations sur certains capteurs et par une valeur faible (resp. moyenne, resp. élevée) des capteurs n'intervenant pas dans la caractérisation.

L'apprentissage des balises constitue le point le plus délicat. En effet, pour entraîner un modèle de balise, il faut un grand nombre de séquences d'observation de cette balise afin de constituer le corpus d'apprentissage. Comme nous l'avons vu dans la section 10.3 l'acquisition des données, leur segmentation et étiquetage se font manuellement et sont très longues.

Actuellement, pour une utilisation en temps réel sur notre robot mobile, notre approche pose deux problèmes :

- D'une part, le taux de reconnaissance nous montre que lorsqu'une balise est reconnue, on ne peut se fier complètement à cette reconnaissance et qu'il faut chercher à savoir si cette reconnaissance correspond à une balise vue en réalité, à une mauvaise reconnaissance ou simplement à une insertion.
- D'autre part, l'algorithme de Viterbi ne permet une reconnaissance des balises

qu'une fois le trajet du robot terminé. La reconnaissance des balises est actuellement hors ligne, ce qui ne peut convenir dans le cas d'une utilisation réelle sur un robot mobile.

Dans le chapitre suivant, nous présentons comment nous avons utilisé notre approche pour reconnaître des balises au fur et à mesure que le robot progresse dans son environnement, et comment nous avons utilisé le taux de reconnaissance des balises afin de savoir comment interpréter chaque balise reconnue. Nous présentons une application de notre approche à la localisation du robot dans son environnement et au contrôle d'exécution d'actions.

11

Application à la localisation et au contrôle d'exécution d'actions

11.1 Introduction

Comme nous l'avons vu dans le chapitre 8, l'apprentissage et la reconnaissance de balises peuvent servir à localiser un robot dans son environnement, à construire des cartes topologiques de l'environnement et à contrôler l'exécution d'actions.

Dans ce chapitre, nous nous intéressons à l'utilisation de notre approche pour localiser le robot dans son environnement au fur et à mesure qu'il reconnaît des balises, puis une fois sa position connue comment il contrôle l'exécution des actions pour atteindre son but. Pour cela, nous avons deux problèmes à résoudre :

– D'une part, le taux de reconnaissance des balises n'est pas suffisamment élevé pour que nous puissions nous fier entièrement à la reconnaissance. En effet, lorsque nous reconnaissons une balise, nous ne savons pas si c'est effectivement cette balise que le robot a "vue", si c'est une mauvaise reconnaissance ou une simple insertion.

  Nous pourrions détecter ces erreurs en comparant (comme nous l'avons fait pour évaluer les taux de reconnaissance) ce que le robot a reconnu avec ce qu'il a vu. La solution serait de construire un modèle de l'environnement contenant les balises que le robot doit reconnaître et de le comparer avec ce que le robot a reconnu. Actuellement, cette comparaison se fait lorsque le trajet du robot est terminé. Pour cela, une fois le déplacement du robot terminé, nous segmentons et étiquetons manuellement (comme nous le faisons pour les données d'apprentissage) la séquence d'observations pour déterminer quelles balises le robot doit reconnaître. Cette séquence d'observations segmentée et étiquetée constitue le modèle de référence par rapport auquel la séquence de balises reconnues est comparée. L'environnement dans lequel évolue le robot étant dynamique, cette segmentation et cet étiquetage ne sont réalisables qu'*a posteriori*, puisqu'il est par exemple impossible de savoir *a priori* si une porte est ouverte ou fermée. Il est donc impossible de prévoir *a priori* ce que le robot doit "voir" et donc de construire la suite de balises que le robot doit reconnaître. De plus, pour pouvoir localiser le robot dans son environnement nous devons savoir non pas à quel type de balise de l'environnement, mais à quelle balise précise de l'environnement correspond chaque balise reconnue. Dans la section 11.2 nous décrivons [Aycard *et al.*, 1998] à partir des balises reconnues et de la confiance accordée à cette reconnaissance, le processus d'interprétation de chacune des balises reconnues et ainsi déterminons l'état de l'environnement (c-à-d, les balises de l'environnement que le robot a vues).

- D'autre part le fonctionnement de l'algorithme de Viterbi : la reconnaissance de balises ne peut être réalisée que lorsque le robot a parcouru son trajet. Actuellement, le robot parcourt son trajet en mémorisant les données des capteurs, puis lorsqu'il a terminé son parcours, une reconnaissance est effectuée sur l'ensemble des données qu'il vient de mémoriser. Dans la section 11.3, nous présentons l'algorithme de Viterbi-Bloc qui permet une reconnaissance des balises naturelles au fur et à mesure que le robot se déplace dans son environnement.

Dans la section 11.4, nous détaillons comment au fur et à mesure que le robot reconnaît et interprète les balises nous déterminons sa position dans l'environnement. Pour terminer dans la section 11.5, nous revenons sur le fonctionnement du contrôleur d'exécution du niveau physique afin de montrer comment, chaque fois que la position du robot est postée par le module de localisation dans la mémoire partagée du niveau physique, il active le module de navigation pour atteindre le but, et ainsi gère la réactivité tactique en faisant le lien entre la planification et l'exécution des actions.

11.2 Interprétation des balises reconnues et détermination de l'état de l'environnement

11.2.1 Introduction

Lorsque Gaston se déplace d'un point à un autre de l'environnement, à partir de la carte topologique de l'environnement (figure 4.10), il est facile de déterminer la suite de N balises B $= \{b_1, b_2, ..., b_N\}$ que le robot est susceptible de voir au cours de son déplacement. En réalité, il va reconnaître une suite de M balises R $= \{r_1, r_2, ..., r_M\}$, car :

- premièrement, la reconnaissance n'est pas parfaite. Nous avons vu que les balises peuvent être confondues ou omises, et que certaines balises reconnues peuvent être de simples insertions et ne correspondre à rien en réalité.
- deuxièmement, l'environnement est dynamique. Une partie (voire la totalité) des balises décrivant l'environnement peuvent ne pas être visibles[27]. C'est le cas par exemple si une porte est fermée ou si une intersection est bloquée.

Il nous faut déterminer une correspondance entre la séquence R de M balises reconnues et la séquence B des N balises de l'environnement. Nous disposons pour cela de deux

27. Dans la suite, nous utiliserons indifféremment le terme porte ouverte (resp. fermée) et le terme porte visible (resp. non visible). De même, nous utiliserons indifféremment le terme intersection (resp. début de couloir, resp. fin de couloir) non bloquée (resp. bloquée), et le terme intersection (resp. début de couloir, resp. fin de couloir) visible (resp. non visible). De manière plus générale, nous parlerons de balises visibles ou non visibles.

informations :

- En utilisant la matrice de confusion de la table 10.3, nous savons quels sont les taux de confusion entre les différentes balises reconnues et nous pouvons en déduire comment accorder un taux de confiance à chaque balise reconnue.
- Expérimentalement, nous avons remarqué que lorsque une porte est fermée, une intersection est bloquée, un début de couloir est bloquée ou une fin de couloir est bloquée, lorsque le robot passe devant ces balises il y a peu ou pas de variations sur la mesure des capteurs servant à caractériser ces balises. A ce moment-là, ces balises sont reconnues comme des couloirs.

A partir de ces informations, il est possible d'interpréter à quoi correspondent les balises reconnues et ainsi déterminer l'état de l'environnement. En effet, notre problème d'interprétation de la séquence R de M balises reconnues se ramène à un problème classique de modèle de Markov caché. Nous pouvons le définir de la façon suivante : étant donné la séquence R de M balises reconnues par le robot, le but est de trouver la séquence optimale de balises (sous-ensemble de B) correspondante à l'état de l'environnement ayant pu générer cette séquence R de M balises. Nous avons vu dans les chapitres précédents que ce problème était typiquement un problème de reconnaissance qui relève de l'algorithme de Viterbi.

Dans cette section, nous définissons dans le premier paragraphe le modèle de Markov caché qui permet de résoudre notre problème d'interprétation de la séquence de balises reconnues. Nous montrons dans le deuxième paragraphe comment à partir de la séquence de balises que le robot peut voir au cours de son déplacement nous construisons la topologie du modèle décrivant l'environnement. Dans le troisième paragraphe, nous expliquons comment nous définissons la matrice de probabilités de transitions et celle d'observations correspondantes à notre modèle. Ensuite, nous montrons comment nous avons utilisé l'algorithme de Viterbi pour interpréter la séquence de balises reconnues, et ainsi déterminer l'état de l'environnement. Pour terminer cette section, nous présentons l'algorithme "forward-backward" qui est un autre algorithme de reconnaissance, et expliquons pourquoi dans le problème que nous traitons, il est plus intéressant que l'algorithme de Viterbi. Dans le dernier paragraphe, nous donnons quelques exemples et résultats pour évaluer notre méthode.

11.2.2 Définition du modèle de Markov caché

Dans notre problème, le modèle de Markov caché représente toutes les séquences de balises que le robot peut voir lorsqu'il se déplace d'un point à un autre de son environnement, (c-à-d toutes les descriptions possibles de l'état de l'environnement). Lorsque le robot se déplace vers un but, il peut voir une séquence B de N balises. Mais certaines de ces balises peuvent ne pas être visibles. Notre modèle comporte un ensemble de (N+1) états appelé S = $\{s_1,s_2,...,s_N,s_{N+1}\}$, où l'état s_i correspond à l'état du modèle après avoir vu la $(i-1)^{eme}$ balise de la séquence B. Nous choisissons un modèle de Markov caché d'ordre 1, car nous considérons que la position du robot dans l'environnement au temps t+1 (c-à-d, l'état du modèle au temps t+1) ne dépend que la position du robot dans l'environnement au temps t (ie, l'état du modèle au temps t).

Les observations du modèle sont en fait les balises que le robot reconnaît au fur et à mesure de son déplacement. Les observations possibles V du modèle sont donc les dix types de balises différentes que le robot peut reconnaître, c'est à dire V = {couloir, début de couloir à droite, début de couloir à gauche, fin de couloir à droite, fin de couloir à gauche, intersection à droite, intersection à gauche, porte à droite, porte à gauche et porte en vis à vis}.

11.2.3 Construction de la topologie du modèle

A partir de la séquence B de N balises que le robot peut voir lorsqu'il se déplace, la topologie de notre modèle à (N+1) états se construit en trois étapes :

- Si toutes les balises sont visibles, alors le robot peut les reconnaître. Nous construisons donc un chemin direct qui va du premier état au $(N+1)^{eme}$ état, et où chaque transition correspond à la balise à reconnaître pour passer de l'état courant à l'état suivant.
- Une balise reconnue peut être une simple insertion, et dans ce cas le robot n'a pas "vu" de nouvelle balise, il reste à la même position dans le couloir. Nous ajoutons donc une transition qui permet de rester dans chaque état, tant que la balise reconnue correspond à une insertion.
- Toutes les balises ne sont pas forcément visibles. En effet, une porte peut être fermée, ou une intersection non visible et dans ce cas, lorsque le robot passe devant ces balises, il reconnaît un couloir. Lorsque le robot a reconnu un couloir, il faut tenir compte du fait qu'une ou plusieurs balises ne sont peut être pas visibles. Il faut ajouter une transition correspondant à une reconnaissance de couloir, et qui permet d'aller de l'état courant à n'importe lequel des états suivants.

Le robot se trouve au départ dans l'état 1, puisqu'il se trouve devant la première balise qu'il est susceptible de voir. Le vecteur de probabilités initiales est égal à $\pi = \{1,0,0,...,0\}$.

Exemple

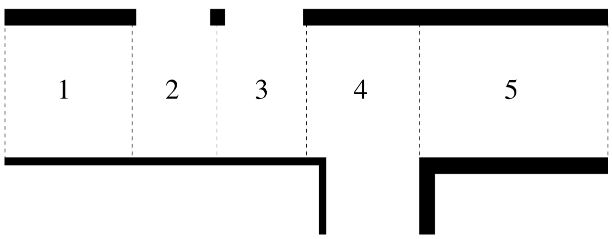


FIG. 11.1 – *La séquence de cinq balises que le robot peut voir*

Nous montrons ici comment lorsque le robot doit se déplacer de la gauche vers la droite dans le couloir de la figure 11.1, nous construisons la topologie du modèle associé.

- Nous construisons un chemin direct (figure 11.2) à six états.
- Pour tenir compte des insertions, nous ajoutons une transition qui permet de rester dans chaque état (figure 11.3).

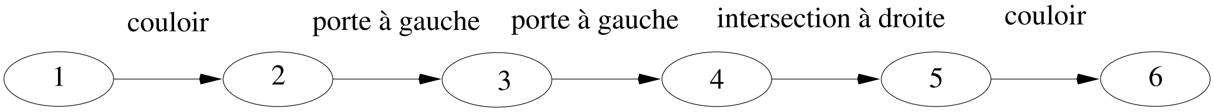


FIG. 11.2 – *Construction de la topologie (1)*

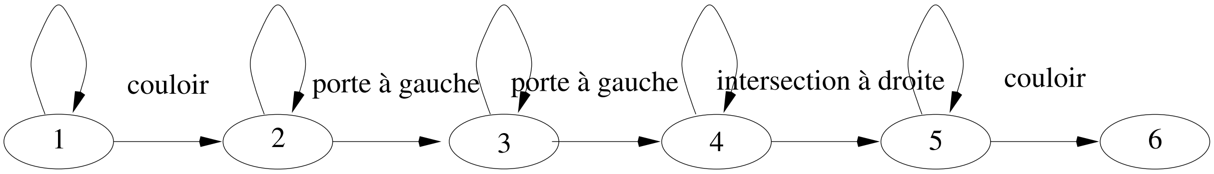


FIG. 11.3 – *Construction de la topologie (2)*

– Pour tenir compte des balises non visibles qui seront donc reconnues comme des couloirs, nous ajoutons des transitions issues de chaque état qui lorsque nous reconnaissons un couloir permettent d'atteindre les états supérieurs (figure 11.4). Cette figure représente la topologie du modèle correspondant à l'environnement de la figure 11.1. De plus, elle détermine tous les séquences de balises que le robot peut reconnaître.

11.2.4 Construction de la matrice de probabilités de transitions et d'observations

Contrairement aux modèles que nous avons utilisés jusqu'à maintenant, les transitions effectuées dépendent de l'interprétation que nous faisons de la balise reconnue. En effet, nous n'effectuons pas la même transition selon que nous considérons que la balise reconnue corresponde à une insertion, une bonne ou mauvaise reconnaissance. En terme de modèle, la transition effectuée dépend de l'observation. Pour répondre à cette spécificité, nous allons construire une matrice de probabilités de transitions qui tienne compte de l'interprétation que nous voulons faire de la balise reconnue (c-à-d, de l'observation). La matrice de probabilités de transitions A = $\{a_{ij}(r)\}$ à trois dimensions sur S x S x V, où $a_{ij}(r) = Prob(q_t = s_j, q_{t-1} = s_i)$, où q_t est l'état actuel au temps t, et r la balise reconnue. Comme nous l'avons vu lors de la définition de la topologie du modèle, chaque transition

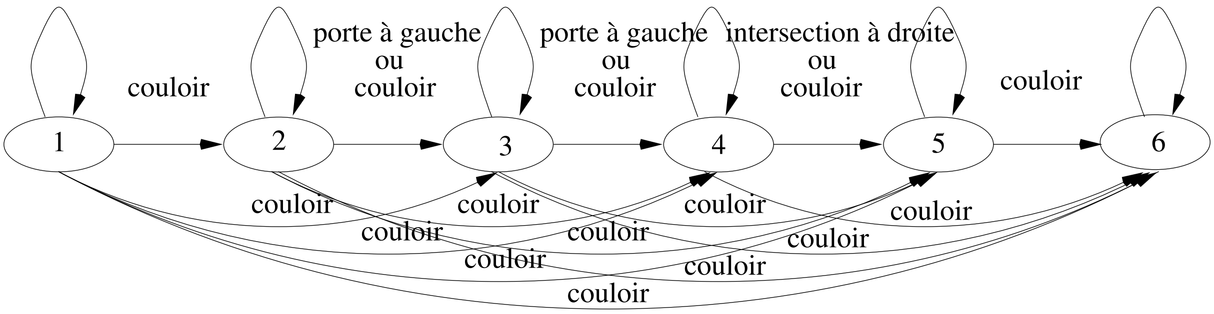


FIG. 11.4 – *Construction de la topologie (3)*

correspond à une interprétation différente de la balise reconnue. Pour définir les éléments de cette matrice, il faut pour chaque balise reconnue et chaque transition (c-à-d, chaque interprétation de la balise reconnue) déterminer la probabilité de cette interprétation.

	couloir	début droit	fin droit	intersection droit	porte droit	début gauche	fin gauche	intersection gauche	porte gauche	porte porte	Ins.
couloir	126	0	1	1	0	0	2	0	0	0	24
début droit	0	7	0	0	0	0	0	0	0	0	0
fin droit	1	0	7	0	0	0	1	0	1	0	0
inter. droit	1	0	0	7	0	0	0	0	0	0	0
porte droit	4	0	0	0	42	0	0	0	1	1	25
début gauche	0	0	0	0	0	8	0	0	0	0	0
fin gauche	0	0	0	0	0	0	6	0	0	0	0
inter. gauche	0	0	0	0	0	0	0	8	0	0	0
porte gauche	1	1	0	0	4	0	0	1	43	1	34
porte porte	0	0	0	1	0	0	0	0	1	2	1
oublis	0	0	1	0	0	0	0	0	0	0	0
Total	133	8	9	9	46	8	9	9	46	4	
% reco.	94	88	77	77	91	100	67	89	93	50	

TAB. 11.1 – *Matrice de confusion des balises reconnues en fonction des balises vues*

	couloir	début droit	fin droit	inter. droit	porte droit	début gauche	fin gauche	inter. gauche	porte gauche	porte porte	oublis
couloir	81%	0%	10%	13%	5%	0%	0%	0%	1%	0%	0%
début droit	0%	100%	0%	0%	0%	0%	0%	0%	1%	0%	0%
fin droit	1%	0%	70%	0%	0%	0%	0%	0%	0%	0%	12%
inter. droit	1%	0%	0%	87%	0%	0%	0%	0%	0%	20%	0%
porte droit	0%	0%	0%	0%	58%	0%	0%	0%	5%	0%	0%
début gauche	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
fin gauche	1%	0%	10%	0%	0%	0%	100%	0%	0%	0%	0%
inter. gauche	0%	0%	0%	0%	0%	0%	0%	100%	4%	0%	0%
porte gauche	0%	0%	10%	0%	1%	0%	0%	0%	51%	20%	0%
porte porte	0%	0%	0%	0%	1%	0%	0%	0%	1%	40%	0%
insertion	16%	0%	0%	0%	35%	0%	0%	0%	37%	20%	0%

TAB. 11.2 – *Matrice de confiance des balises vues en fonction des balises reconnues*

Dans le chapitre précédent, pour évaluer notre approche nous avons construit une matrice de confusion qui donne, en fonction de ce que le robot a "vu", ce qu'il va reconnaîre. Actuellement, il s'agit de déterminer en fonction de ce qu'il a reconnu ce qu'il a réellement "vu", c'est à dire de savoir étant donné une balise reconnue, quelles sont les possibilités d'interprétation de cette balise et leur vraisemblance respective. Ceci revient en fait à faire le contraire de ce que nous avons fait pour la construction de la matrice de confusion 10.3. Donc, pour construire la matrice de confiance accordée aux diverses balises reconnues (table 11.2), il suffit de prendre la transposée de la matrice de confusion (table 10.3) dans laquelle nous avons ajouté les confusions correspondantes à un couloir (table 11.1). En effet, nous devons ici savoir pour chaque balise reconnue quel est le pourcentage qu'elle corresponde à une balise "couloir". Pour obtenir des pourcentages nous normalisons cette matrice de confiance. L'élément c_{ij} à la ligne i et à la colonne j de la matrice de confiance (table 11.2) est une probabilité conditionnelle qui représente la probabilité que la balise i ait été vue sachant que la balise reconnue est j.

A partir de la topologie du modèle, nous savons pour chaque transition à quelle interprétation de la balise reconnue elle correspond. Donc, pour construire la matrice de transitions, la probabilité de chaque transition est une probabilité conditionnelle égale à la probabilité de la balise reconnue sachant à quelle interprétation elle correspond. Pour chaque balise reconnue, nous avons trois types de transitions possibles :

– La balise r reconnue correspond à une insertion. Dans ce cas, si avant la reconnaissance de la balise r le robot se trouve dans l'état s_i, il reste dans cet état. Etant dans l'état s_i, la probabilité de rester dans l'état s_i ayant reconnu une balise r, sachant que cette balise correspond à une insertion est donnée par la probabilité conditionnelle suivante :

$$a_{ii}(r) = Pr(r|insertion)$$

D'autre part, nous avons remarqué dans les sections 10.6 et 10.7.5 que lorsque le robot doit reconnaître une balise, il peut ne pas la voir, c'est à dire l'omettre. Nous devons donc considérer qu'avant d'avoir inséré la balise r, le robot a pu omettre une ou plusieurs balises. Dans ce cas, si le robot se trouve dans l'état s_i, il omet tous les balises comprises entre l'état s_i et l'état s_j, et après l'insertion de la balise r, il reste dans l'état s_j. Etant dans l'état s_i, la probabilité de passer à l'état s_j, sachant que toutes les balises à reconnaître entre l'état s_i et l'état s_j ont été omises, et qu'ayant reconnu la balise r nous restons dans l'état s_j sachant que cette balise correspond à une insertion, est donnée par le produit des probabilités conditionnelles suivantes :

$$a_{ij}(r) = \prod_{k=i}^{j} Pr(b_k|omission) \times Pr(r|insertion)$$

– La balise r reconnue correspond à la balise b_i à reconnaître. Dans ce cas, si le robot se trouve dans l'état s_i, il passe à l'état s_{i+1}. Etant dans l'état s_i, la probabilité de passer dans l'état s_{i+1} ayant reconnu une balise r et sachant que cette balise correspond à la balise i est donnée par la probabilité conditionnelle suivante :

$$a_{i(i+1)}(r) = Pr(r|b_i)$$

Pour considérer les possibilités d'omissions, nous supposons qu'avant d'avoir reconnu la balise r, le robot a pu omettre une ou plusieurs balises. Dans ce cas, si le robot se trouve dans l'état s_i, il omet tous les balises comprises entre l'état s_i et l'état s_{j-1}, et après la reconnaissance de la balise r, il se trouve dans l'état s_j. Etant dans l'état s_i, la probabilité de passer à l'état s_j, sachant que toutes les balises à reconnaître entre l'état s_i et l'état s_{j-1} ont été omises, et qu'ayant reconnu la balise r nous passons dans l'état s_j sachant que la balise r correspond à une reconnaissance, est donnée par le produit des probabilités conditionnelles suivantes :

$$a_{ij}(r) = \prod_{k=i}^{j-1} Pr(b_k|omission) \times Pr(r|b_j)$$

– La balise r reconnue correspond à des balises non visibles (c-à-d, une porte fermée ou un début de couloir, une fin de couloir ou une intersection bloquée). La balise r reconnue correspond donc à un couloir. Dans ce cas, si le robot se trouve dans l'état s_i, il passe à l'état s_j avec i<j. Etant dans l'état s_i, la probabilité de passer dans l'état s_j ayant reconnu une balise r et sachant que cette balise correspond à un couloir est donnée par la probabilité conditionnelle suivante :

$$a_{ij}(r) = Pr(r|couloir)$$

Pour considérer les omissions, nous supposons qu'avant d'avoir reconnu la balise r, le robot a pu omettre une ou plusieurs balises. Dans ce cas, si le robot se trouve dans l'état s_i, il omet tous les balises comprises entre l'état s_i et l'état s_k, et ensuite toutes les balise comprises entre l'état s_k et s_j ne sont pas visibles. Etant dans l'état s_i, la probabilité de passer à l'état s_j, sachant que toutes les balises à reconnaître entre l'état s_i et l'état s_k ont été omises, et qu'ayant reconnu la balise r nous passons dans l'état s_j sachant que la balise r correspond à toutes les balises non visibles entre s_k et s_j, est donnée par le produit des probabilités conditionnelles suivantes :

$$a_{ij}(r) = \prod_{l=i}^{k} Pr(b_l|omission) \times Pr(r|couloir)$$

L'ensemble de ces paramètres $\{N,S,A,\pi,V\}$ constitue le modèle de Markov caché λ.

Remarque

Toutes les probabilités de transitions dépendent de la balise r reconnue. Il faudra donc lors de l'interprétation d'une séquence R de M balises reconnues, recalculer la matrice de probabilités de transitions pour interpréter chaque balise reconnue.

L'observation étant associée à la transition, nous avons ici une seule matrice qui est à la fois la matrice de probabilités de transitions et d'observations.

11.2.5 Interprétation des balises reconnues et détermination de l'état de l'environnement

L'algorithme de Viterbi

Nous modifions le calcul de $\delta_t(j)$ pour tenir compte de notre matrice de probabilités de transitions et d'observations, et pour normaliser, nous divisons par la somme de tous les alignements possibles au temps t.

$$\delta_t(j) = \frac{max_{1\leq i\leq N+1}[\delta_{t-1}(i) \cdot a_{ij}(r)]}{\sum_{i=1}^{N+1}[\delta_{t-1}(i) \cdot a_{ij}(r)]} \quad 2 \leq t \leq T, \;\; 1 \leq j \leq N+1 \tag{11.1}$$

avec l'initialisation suivante :

$$\delta_1(i) = \pi_i, \;\; 1 \leq i \leq N+1$$

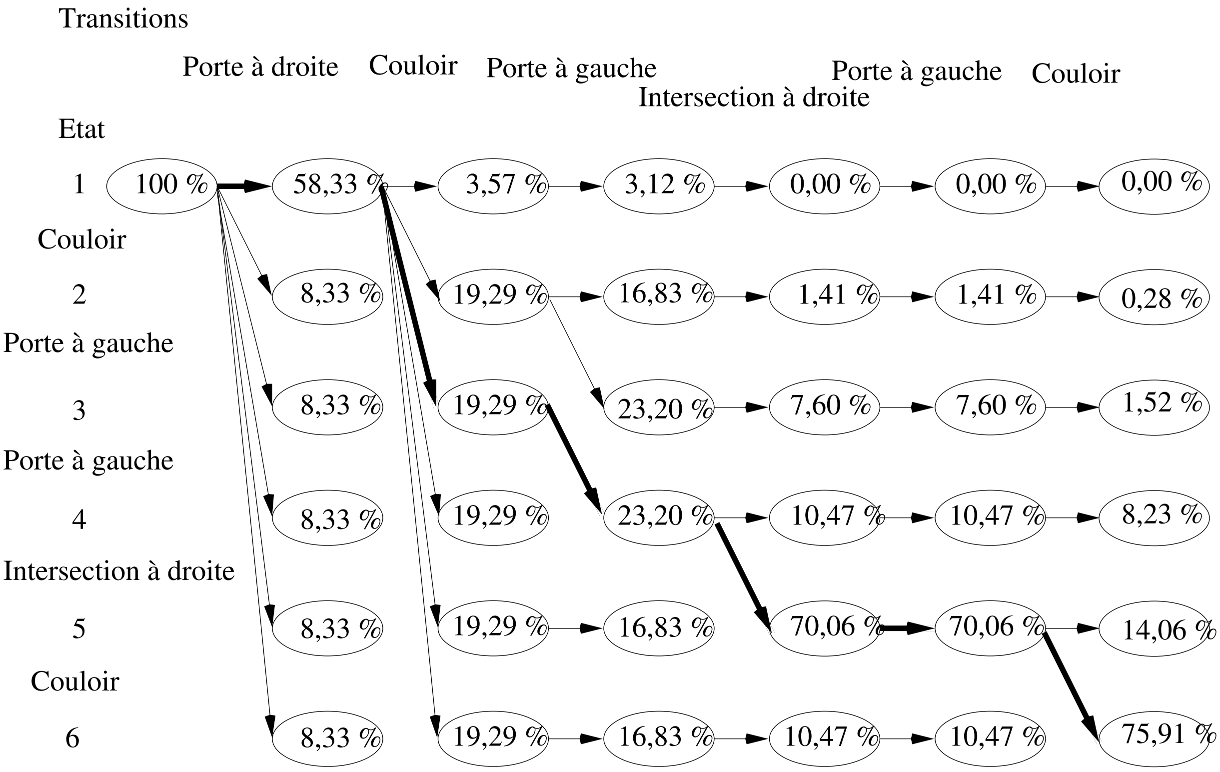


FIG. 11.5 – *Interprétation des balises reconnues en utilisant l'algorithme de Viterbi*

Exemple

Admettons que le robot se déplace dans le couloir de la figure 11.1, il est susceptible de voir la séquence de cinq balises suivantes : couloir, porte à gauche, porte à gauche, intersection à droite, couloir. Admettons maintenant qu'il est reconnu la séquence R de six balises : porte à droite, couloir, porte à gauche, intersection à droite, porte à gauche, couloir. En utilisant la topologie du modèle de la figure 11.4 et l'algorithme de Viterbi (figure 11.5), nous pouvons déterminer que le robot est passé par la suite d'états suivants : 1, 1, 3, 4, 5, 5, 6. Nous en déduisons l'interprétation de chaque balise reconnue :

- La première balise reconnue est une porte à droite. Partant de l'état 1, le robot reste dans l'état 1. Nous en déduisons donc une insertion de cette balise.
- La deuxième balise reconnue est un couloir. Partant de l'état 1, le robot se rend dans l'état 3. Nous en déduisons donc que le robot n'est pas passé par l'état 2, et donc que la première porte à gauche est fermée.
- La troisième balise reconnue est une porte à gauche. Partant de l'état 3, le robot se rend dans l'état 4. Nous en déduisons donc que la deuxième porte à gauche vient d'être reconnue, et donc qu'elle est ouverte.
- La quatrième balise reconnue est une intersection à droite. Partant de l'état 4, le robot se rend dans l'état 5. Nous en déduisons donc que l'intersection vient d'être reconnue, et donc qu'elle est visible.
- La cinquième balise reconnue est une porte à gauche. Partant de l'état 5, le robot reste dans l'état 5. Nous en déduisons donc une insertion de cette balise.

- La dernière balise reconnue est un couloir. Partant de l'état 5, le robot se rend dans l'état 6. Nous en déduisons donc que le dernier morceau de couloir vient d'être reconnue.

A chaque instant, pour un état donné, il y a six alignements temporels différents qui se terminent dans cet état selon l'interprétation que nous faisons de la balise r reconnue. En effet, pour calculer la probabilité de l'alignement se terminant dans un état s_i donné au temps t, nous pouvons considérer :

- la probabilité de l'alignement se terminant dans l'état s_i au temps t-1, et que la balise r reconnue corresponde à une simple insertion.
- la probabilité de l'alignement se terminant dans l'état s_j au temps t-1 avec j<i, et que les balises comprises entre s_j et s_i soient omises et que la balise r reconnue corresponde à une simple insertion.
- la probabilité de l'alignement se terminant dans l'état s_{i-1} au temps t-1, et que la balise r reconnue corresponde à la balise b_{i-1} à reconnaître.
- la probabilité de l'alignement se terminant dans l'état s_j au temps t-1 avec j<i-1, et que les balises comprises entre s_j et s_{i-1} soient omises et que la balise r reconnue corresponde à la balise b_{i-1} à reconnaître.
- la probabilité de l'alignement se terminant dans l'état s_j avec j<i, et que la balise r reconnue corresponde à un couloir, c'est à dire que les balises comprises entre l'état s_j et l'état s_i ne soiennt pas visibles.
- la probabilité de l'alignement se terminant dans l'état s_k avec k<i, et que les balises comprises entre s_k et s_j avec k<j<i sont omises et que la balise r reconnue corresponde à un couloir, c'est à dire que les balises comprises entre l'état s_j et l'état s_i ne sont pas visibles.

Pour ne pas surcharger la figure 11.5, nous n'avons pour chaque état présenté que l'alignement temporel correspondant à l'interprétation des balises reconnues la plus probable se terminant dans chaque état.

Sur l'ensemble des $6N$ alignements temporels possibles à chaque instant, l'algorithme de Viterbi recherche l'alignement de probabilité maximum, c'est à dire celui qui correspond à l'interprétation la plus probable des balises reconnues. Intuitivement, ceci se justifie par le fait que pour une séquence de balises reconnues, il n'y a qu'une interprétation qui correspondent à l'état réel de l'environnement. Par contre, en pratique cette recherche pose quelques problèmes, car dans certains cas l'algorithme de Viterbi trouve plusieurs alignements temporels qui ont la même probabilité, ce qui signifie que nous avons plusieurs interprétations de l'environnement aussi vraisemblables les unes que les autres, ce qui est totalement irréaliste. Dans notre exemple, lorsque le robot reconnaît le premier couloir, les alignements se terminant dans l'état 2, 3, 4, 5 ou 6 ont la même probabilité. En effet, le meilleur alignement se terminant dans ses états, est le produit du meilleur alignement se terminant dans l'état 1 à l'instant précédent et de la probabilité de reconnaître un couloir sachant qu'on doit reconnaître un couloir. Pourtant l'interprétation de la balise reconnue (c-à-d, le couloir) n'est pas la même pour tous les alignements. Pour l'alignement se terminant dans l'état 2, la balise reconnue correspond à une bonne reconnaissance (c-à-d, une transition directe entre l'état 1 et l'état 2), alors que pour les autres alignements, elle correspond à une ou plusieurs balises qui ne sont pas visibles. Si la séquence de balises reconnues n'était composée que des deux premières balises reconnues, alors il ne serait pas possible de déterminer l'état de l'environnement. Ce problème vient du fait que nous considérons indépendamment les six interprétations possibles associées à une balise reconnue, et donc qu'à un instant donné pour chaque état nous avons six alignements temporels possibles. En considérant l'ensemble des alignements temporels pour chaque état, nous aurions sans doute eu à chaque instant et pour chaque état des alignements différents, et nous aurions pu déterminer l'alignement le plus probable et l'interprétation de chaque balise à chaque instant. Intuitivement, ceci se justifie par le fait que pour une balise reconnue, nous ne savons pas *à priori* quelle est son interprétation et donc l'alignement temporel pour chaque état possible à un instant donné doit prendre en compte l'ensemble des interprétations possibles.

Pour tenir compte de ce genre de problème, nous présentons dans le paragraphe suivant l'algorithme "forward-backward" qui prend en compte toutes les interprétations possibles pour le calcul de l'alignement se terminant dans un état donné à un instant donné. Avec cet algorithme, à chaque balise reconnue, chaque alignement a une probabilité différente, et à chaque instant, on peut déterminer l'alignement le plus probable et interpréter les balises au fur et à mesure qu'elles sont reconnues.

L'algorithme "forward-backward"

En utilisant l'algorithme "forward-backward", la probabilité que le robot soit dans l'état i au temps t peut être exprimée comme :

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(j)}{\sum_{i=1}^{N+1} \alpha_t(i)\beta_t(j)}$$

Comme nous cherchons à savoir quel est l'état où le robot a la plus grande probabilité de se trouver après chaque balise naturelle reconnue, nous n'avons besoin que de la probabilité "forward". Notre problème se résume donc à :

$$\gamma_t(i) = \frac{\alpha_t(i)}{\sum_{i=1}^{N+1} \alpha_t(i)}$$

Comme nous l'avons vu dans la présentation de l'algorithme de Baum-Welch, la probabilité "forward" peut être calculée récursivement :

$$\alpha_t(j) = \sum_{i=1}^{N+1} \alpha_{t-1}(i).a_{ij}(r), \;\; 2 \leq t \leq T, \;\; 1 \leq j \leq N+1$$

avec l'initialisation suivante :

$$\alpha_1(i) = \pi_i, \;\; 1 \leq i \leq N+1$$

D'autre part, lors de la définition de la matrice de probabilités de transitions, nous avons considéré que la probabilité qu'une balise soit non visible est aussi importante que la probabilité qu'une suite de balises soient non visibles, ce qui n'est pas très réaliste. En effet, la probabilité P_N que N balises ne soient pas visibles est égale au produit de la probabilité P_{N-1} que les (N-1) premières balises ne soient pas visibles et de la probabilité que la N^{eme} balise ne soit pas visible. P_N est donc inférieur à P_{N-1}. Pour tenir compte de cela, lors du calcul de $a_{ij}(r)$, nous divisons la probabilité conditionnelle $\Pr(r|\text{couloir})$ par le nombre de balises non visibles. La probabilité conditionnelle que la balise r soit reconnue sachant que cette balise correspond à un couloir est donnée par :

$$a_{ij}(r) = \frac{Pr(r|couloir)}{j-i}$$

Dans le cas où il y a des omissions, la probabilité conditionnelle que la balise r soit reconnue sachant que cette balise correspond à un couloir et qu'une ou plusieurs balises ont été omises est donnée par :

$$a_{ij}(r) = [\prod_{l=i}^{k} Pr(l|omission)] * \frac{Pr(r|couloir)}{j-k}$$

Exemples

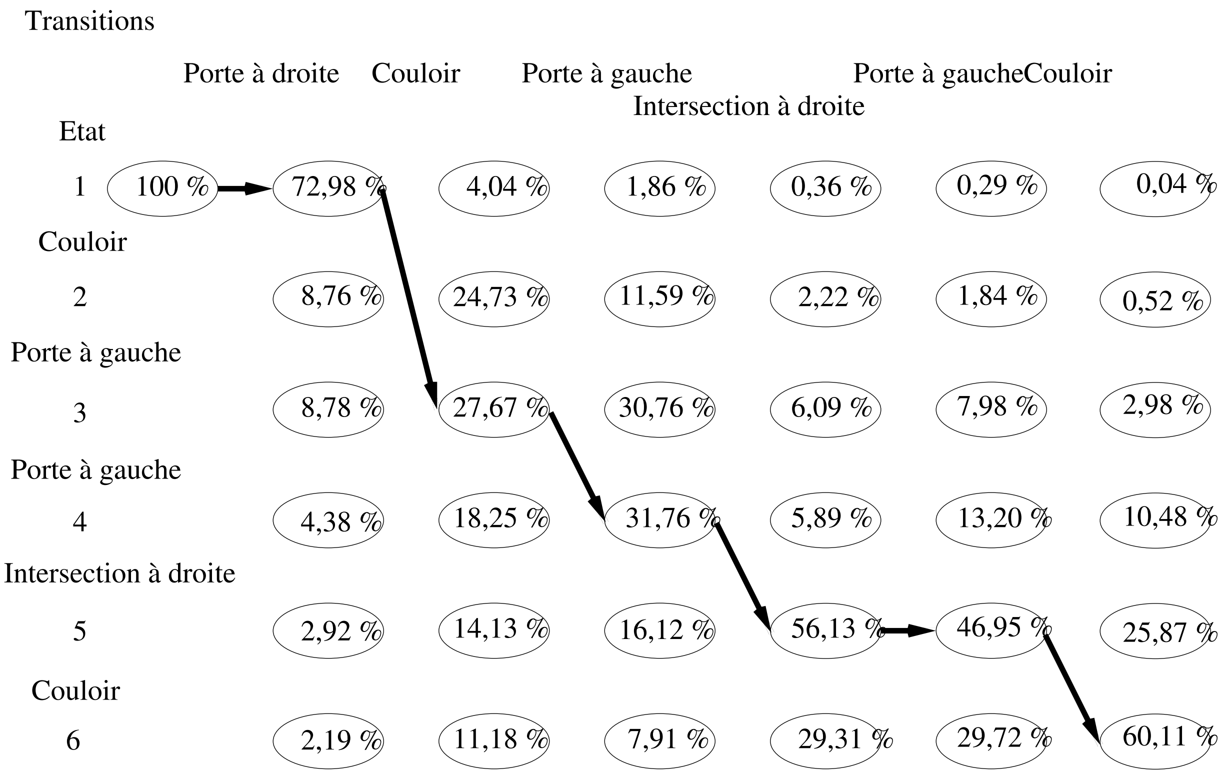


FIG. 11.6 – *Interprétation des balises reconnues en utilisant l'algorithme "forward-backward"*

Reprenons l'exemple précédent. En utilisant l'algorithme "forward-backward" (figure 11.6), nous pouvons déterminer que le robot est passé par la suite d'états suivants : 1, 1, 3, 4, 5, 5, 6 qui est la même séquence que celle déterminée par l'algorithme de Viterbi. Par contre à un instant donné, les probabilités des alignements se terminant dans chaque état sont toutes différentes. La séquence de balises correspondante à l'état de l'environnement (c-à-d, la séquence de balises "vues" par le robot) est déterminée en prenant à chaque instant, l'état correspondant à l'alignement le plus probable. En effet, à chaque instant l'alignement temporel se terminant dans un état donné est calculé à partir de l'ensemble des alignements temporels à l'instant précédent et des interprétations possibles, et non plus à partir d'un alignement temporel et d'une interprétation comme précédemment. Ceci est un avantage indéniable puisque que contrairement à l'algorithme de Viterbi où nous sommes obligés d'avoir toute la séquence de balises reconnues pour déterminer la séquence d'états du modèle correspondante à l'interpretation des balises reconnues, avec l'algorithme "forward-backward" à chaque fois qu'une balise est reconnue, nous savons immédiatement comment l'interpréter (grâce à la transition reliant l'alignement le plus probable avant sa reconnaissance avec l'alignement le plus probable après sa reconnaissance), et à quelle balise de l'environnement elle correspond.

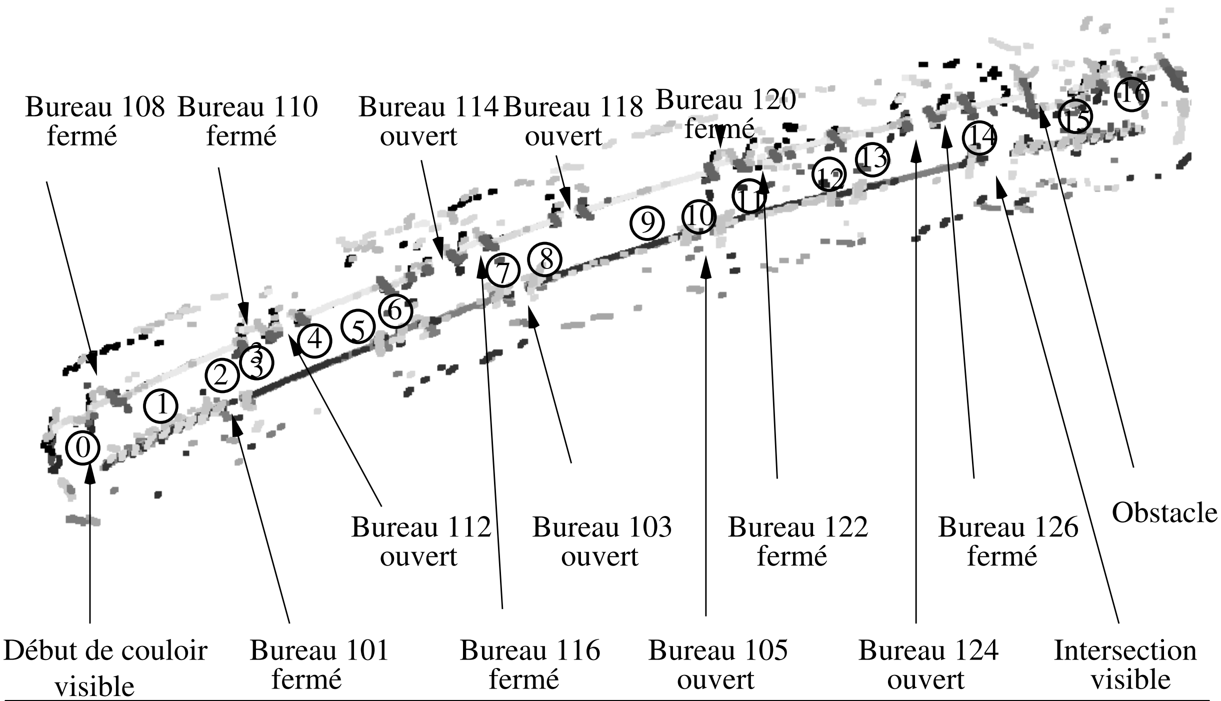
Dans un deuxième exemple, nous nous intéressons à l'interprétation de toutes les balises reconnues lors d'une traversée de couloir. La figure 11.7 donne le détail d'une traversée de couloir (le robot se déplaçant de la gauche vers la droite). Le contenu des deux premières colonnes sera détaillé dans la section 11.3 lors de la description du fonctionnement de l'algorithme de Viterbi-bloc. La colonne six sera détaillée dans la section 11.4. La troisième colonne donne la suite de balises reconnues lorsque l'algorithme de Viterbi est exécuté le robot se trouvant à la position seize de la figure. Les colonnes quatre et cinq donnent l'interprétation des balises reconnues et l'état réel des balises du couloir.

Nous constatons que l'interprétation des balises reconnues est très bonne et correspond presque complètement à l'état réel du couloir lors de la traversée. L'intersection et le début de couloir ont été reconnues. Mise à part, la porte ouverte du bureau B112, toutes les autres portes ouvertes ont été reconnues. En fait, la porte ouverte du Bureau B110 a été reconnue à la place de la porte ouverte du Bureau B112. De plus, le robot a reconnu deux portes ouvertes (celle du bureau 108, et celle du bureau 122) alors qu'elles étaient fermées.

11.2.6 Résultats expérimentaux

Pour évaluer l'interprétation des balises reconnues, nous comparons l'interprétation des balises reconnues des vingt traversées de couloir de la section 10.5 avec l'état réel (c-à-d, visible ou non visible) des balises du couloir lors de ces vingt traversées. Cette évaluation est réalisée en deux phases : premièrement nous procédons à la reconnaissance des balises en utilisant l'algorithme de Viterbi, puis nous contruisons un modèle de Markov caché représentant les balises que le robot peut voit lors de la traversée du couloir, et utilisons ce modèle et l'algorithme "forward-backward" pour déterminer l'interprétation de la séquence de balises reconnues avec Viterbi. Cette interprétation est comparée avec l'état réel du couloir lorsque le robot le traverse. Si l'on regarde par exemple la figure 11.7, il y a un début de couloir et une intersection visibles, et qui lors de l'interprétation des balises reconnues ont été reconnues comme étant visibles. Il y a six portes ouvertes lors de la traversée de couloir ; cinq ont été reconnues comme des portes ouvertes et une comme une porte fermée. Il y a sept portes fermées lors de la traversée de couloir, et quatre ont été reconnues comme des portes fermées et trois comme des portes ouvertes.

Pour chaque balise, les résultats sont donnés sous la forme d'une matrice de confusion.



Position du robot	Balises reconnues avec Viterbi-bloc	Balises reconnues avec Viterbi	Interprétation des balises reconnues	Etat réel du couloir	Recalage (en cms)
1	Couloir	Couloir	Insertion		
2	Début de couloir à droite	Début de couloir à droite	Début de couloir visible	Début de couloir visible	7,5 cms
3	Porte à gauche Couloir	Porte à gauche Couloir	Bureau B108 ouvert	Bureau B108 fermé	90 cms 15 cms
4	Porte à gauche	Porte à gauche	Insertion		
5	Couloir	Couloir	Bureau B101 fermé	Bureau B101 fermé	0 cms
6	Porte à gauche	Porte à gauche	Bureau B110 ouvert	Bureau B110 fermé	97,5 cms
7	Couloir	Couloir	Bureau B112 fermé	Bureau B112 ouvert	-15 cms
8	Porte à gauche	Porte à gauche	Bureau B114 ouvert	Bureau B114 ouvert	0 cms
9	Porte à droite	Porte à droite	Bureau B116 fermé Bureau B103 ouvert	Bureau B116 fermé Bureau B103 ouvert	7.5 cms 0 cms
10	Couloir Porte à gauche	Couloir Porte à gauche	Bureau B118 ouvert	Bureau B118 ouvert	15 cms 0 cms
11	Couloir	Couloir			
12	Porte à droite	Porte à droite	Bureau B105 ouvert Bureau B120 fermé	Bureau B105 ouvert Bureau B120 fermé	15 cms 0 cms
13	Porte à gauche	Porte à gauche	Bureau B122 ouvert	Bureau B122 fermé	45 cms
14	Couloir	Couloir			22.5 cms
15	Porte à gauche Couloir	Porte à gauche Couloir	Bureau B124 ouvert Bureau B126 fermé	Bureau B124 ouvert Bureau B126 fermé	0 cms 0 cms
16	Intersection à droite	Intersection à droite	Intersection visible	Intersection visible	7.5 cms

FIG. 11.7 – *Balises reconnues lors d'une traversée de couloir*

Chaque colonne correspond à l'état réel (c-à-d, visible ou non visible) de la balise reconnue, et chaque ligne correspond à son état donné par l'interprétation. Pour chaque table :

- L'élément $c_{1,1}$ à la ligne 1 et à la colonne 1 est le nombre de fois où une balise visible de l'environnement a été interprétée comme étant visible.
- L'élément $c_{2,1}$ à la ligne 2 et à la colonne 1 est le nombre de fois où une balise visible de l'environnement a été interprétée comme n'étant pas visible.
- L'élément $c_{1,2}$ à la ligne 1 et à la colonne 2 est le nombre de fois où une balise non visible de l'environnement a été interprétée comme étant visible.
- L'élément $c_{2,2}$ à la ligne 2 et à la colonne 2 est le nombre de fois où une balise non visible de l'environnement a été interprétée comme n'étant pas visible.

Il n'y a évidemment pas d'insertions ni d'omissions puisqu'ici il s'agit de déterminer si l'interprétation d'une balise reconnue correspond à l'état réel de la balise correspondante de l'environnement ou non.

Les quatres paragraphes suivants donnent la comparaison entre l'état réel des quatres types de balises, à savoir les portes, les intersections, les débuts de couloir et les fins de couloirs, et leur interprétation par l'algorithme "forward-backward". Nous ne distinguons pas les balises à droite des balises à gauche.

Comparaison entre l'état réel des portes de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues

	porte ouverte	porte fermée
porte ouverte	89	31
porte fermée	16	78
% reco.	85%	72%

TAB. 11.3 – *Table de confusion entre l'état réel des portes ouvertes et leur état déterminé lors de l'interprétation des balises reconnues*

La table 11.3 donne la comparaison entre l'état réel des portes de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues. Nous constatons que 28% des portes fermées de l'environnement sont, lors de la reconnaissance et de l'interprétation des balises reconnues, prises pour des portes ouvertes. Il y a deux raisons à ce problème :

- Dans une séquence de balises reconnues, il y a toujours un grand nombre de portes qui ne sont que des insertions. Pourtant de nombreuses portes ouvertes de la séquence de balises reconnues n'ont pas été interprétées comme des insertions mais comme des portes ouvertes de l'environnement. C'est le cas par exemple dans la figure 11.7 de la porte à gauche (troisième reconnaissance) qui n'est en fait qu'une insertion. Cette mauvaise interprétation détermine la porte du bureau B108 comme ouverte alors qu'en réalité elle est fermée.
- Dans 4% des cas, lorsque le robot doit reconnaître un couloir correspondant à une porte fermée, il reconnaît une porte ouverte.

Nous avons remarqué que lorsque plusieurs portes du même côté du couloir sont l'une à côté de l'autre (c-à-d, elles sont reconnues les unes après les autres), et qu'elles sont toutes fermées sauf la dernière, le robot va reconnaître un couloir puis une porte, en considérant qu'il n'y a pas d'insertions. Etant donné que nous n'utilisons aucune information sur la position des balises, l'interprétation a souvent tendance à reconnaître que l'une d'entre elles est ouverte et la dernière est fermée. C'est le cas par exemple dans la figure 11.7, où la porte du Bureau B110 est fermée et celle du Bureau B112 est ouverte, et que la porte du Bureau B110 est interprétée comme étant ouverte et celle du Bureau B112 est interprétée comme étant fermée. Ceci a pour effet d'interpréter la reconnaissance d'une porte ouverte comme une porte fermée, et vice versa. C'est pour cela que 15% des portes fermées de l'environnement sont reconnues comme des portes ouvertes. Ceci entre bien évidemment en compte pour les 28% de portes fermées qui lors de l'interprétation sont déterminées comme ouvertes.

De plus, le couloir utilisé est quasiment composé exclusivement de portes. Comme nous n'utilisons aucune information sur la position des balises dans le couloir, lorsque le robot reconnaît une séquence composée exclusivement de couloirs et de portes ouvertes, il est difficile d'interpréter cette séquence pour déterminer quelles portes sont ouvertes et quelles portes sont fermées. Dans tous les cas, la détermination de l'état des portes de l'environnement est juste dans 78% des cas, ce qui est tout à fait honorable.

Comparaison entre l'état réel des intersections de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues

	intersection visible	intersection non visible
intersection visible	15	0
intersection non visible	3	5
% reco.	83%	100%

TAB. 11.4 – *Table de confusion entre l'état réel des intersections et leur état déterminé lors de l'interprétation des balises reconnues*

La table 11.4 donne le taux de confusion entre l'état réel des intersections et leur état déterminé lors de l'interprétation des balises reconnues. 17% des intersections visibles sont reconnues comme des couloirs, et sont donc interprétées comme des intersections non visibles :

- D'après la table 11.1, nous constatons que lorsque le robot doit reconnaître une intersection à droite, dans 11% des cas, il reconnaît un couloir. Dans ce cas, les intersections à droite qui sont visibles sont reconnues comme des couloirs et sont donc interprétés comme des intersections non visibles.
- De plus, 11% des intersections à gauche sont reconnues comme des portes à gauche et sont donc interprétées comme des portes ouvertes, alors qu'en fait elles correspondent à des intersections visibles.

Lorsque le robot doit reconnaître un couloir (c-à-d, une intersection non visible), il y a un peu moins d'1% des chances qu'il reconnaisse une intersection. C'est pour cela que 100% des intersections non visibles sont bien interprétées.

Comparaison entre l'état réel des débuts de couloirs de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues

	début de couloir visible	début de couloir non visible
début de couloir visible	16	0
début de couloir non visible	0	3
% reco.	100%	100%

TAB. 11.5 – *Table de confusion entre l'état réel des débuts de couloirs et leur état déterminé lors de l'interprétation des balises reconnues*

Les taux de reconnaissance des débuts de couloirs sont très bons. Donc à chaque fois que le robot reconnaît un début de couloir, cette reconnaissance correspond à un début de couloir de l'environnement. Il n'y a donc qu'un très faible taux de confusion possible au moment de l'interprétation d'un début de couloir. Si dans la séquence de balises reconnues, nous avons un début de couloir, c'est qu'il correspond à un début de couloir de l'environnement, et si nous n'en avons pas, c'est que le début de couloir n'est pas visible.

Comparaison entre l'état réel des fins de couloirs de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues

	fin de couloir visible	fin de couloir non visible
fin de couloir visible	16	0
fin de couloir non visible	2	5
reco.	89%	100%

TAB. 11.6 – *Table de confusion entre l'état réel des fins de couloirs et leur état déterminé lors de l'interprétation des balises reconnues*

Dans 22% des cas, lorsque le robot doit reconnaître une fin de couloir à gauche, il reconnaît un couloir. Il en est de même dans 11% des cas, lorsque le robot doit reconnaître une fin de couloir à droite. Ce couloir reconnu est bien évidemment interprété comme une fin de couloir non visible alors qu'en fait elle est visible. C'est pour cela que 11% des fins de couloirs visibles de l'environnement sont au moment de l'interprétation déterminées comme étant non visibles.

Par contre, lorsqu'un couloir doit être reconnu (c-à-d, une fin de couloir non visible) il ne peut que très rarement être reconnu comme une fin de couloir.

Comparaison entre l'état réel des balises de l'environnement et leur état déterminé lors de l'interprétation des balises reconnues

	balise visible	balise non visible
balise visible	135	31
balise non visible	22	91
reco.	86%	75%
reco globale	81%	

TAB. 11.7 – *Table de confusion entre l'état réel des balises et leur état déterminé lors de l'interprétation des balises reconnues*

La table 11.7 donne les taux de confusion entre l'état réel des balises et leur état déterminé lors de l'interprétation des balises reconnues. Dans l'ensemble l'interprétation des balises reconnues est relativement fiable. Plus de quatre interprétations sur cinq correspondent à la réalité. Le problème majeur est la reconnaissance des portes ouvertes interprétées comme des portes ouvertes de l'environnement (11% des cas) alors qu'en fait elles ne correspondent qu'à des insertions. Ceci est d'autant plus gênant que ces balises serviront pour le recalage du robot lors d'une traversée de couloir. Par contre, les balises visibles reconnues comme des balises non visibles de l'environnement (8% des cas) ne poseront pas de réel problème. En effet, le robot croyant qu'elles ne sont pas visibles ne les utilisera pas pour la localisation.

11.2.7 Conclusion

Dans cette section, nous avons présenté une méthode pour déterminer l'état des balises de l'environnement à partir d'une séquence de balises reconnues. Cette interprétation est réalisée en deux phases : une fois le trajet du robot terminé, l'algorithme de Viterbi donne la séquence de balises reconnues. Puis, nous construisons un modèle de Markov caché représentant toutes les séquences de balises que le robot peut voir lors du déplacement réalisé. A partir de la séquence de balises reconnues, nous déterminons en utilisant l'algorithme "forward-backward" la séquence de balises de l'environnement qui correspond le mieux à cette séquence de balises reconnues qui constituent en fait la séquence d'observations du modèle. D'autre part, le calcul de cette séquence de balises de l'environnement correspond tout à fait à l'exécution d'un plan dans un cadre POMDP. Au fur et à mesure que le robot se déplace et que les balises sont reconnues et interprétées, l'ensemble des états possibles[28] du modèle est mis à jour et permet de déterminer les différentes balises réelles auxquelles correspond cette reconnaissance ainsi que leurs vraisemblances respectives. La séquence de balises est calculée en prenant après chaque reconnaissance, l'état correspondant à l'alignement le plus probable du modèle. De plus, ce calcul permet à chaque fois qu'une balise est reconnue de l'interpréter. Il a l'avantage par rapport à

28. belief states d'un POMDP

l'algorithme de Viterbi que nous ne sommes pas obligés d'attendre d'avoir la séquence complète de balises reconnues pour procéder à l'interprétation de ces balises. Il faut bien évidemment dans ce cas que le robot soit capable de reconnaître les balises au fur et à mesure qu'il les voit, et non pas seulement à la fin de son parcours. Dans la section suivante, nous présentons justement l'algorithme de Viterbi-bloc qui permet au robot de reconnaître les balises au fur et à mesure qu'il évolue dans son environnement.

Pour terminer, nous pouvons dire que les résultats sont bons puisqu'à partir d'une balise reconnue, nous déterminons à quelle balise de l'environnement elle correspond avec une fiabilité de 81%.

11.3 L'algorithme de Viterbi-Bloc

11.3.1 Introduction

Nous avons vu dans la section précédente comment à chaque fois qu'une balise est reconnue, nous pouvons l'interpréter et déterminer à quelle balise de l'environnement elle correspond. Actuellement, les balises ne sont reconnues qu'une fois le déplacement du robot terminé, et donc l'interprétation des balises reconnues ne se fait qu'une fois le déplacement du robot achevé. Dans cette section, nous présentons l'algorithme Viterbi-bloc qui permet une reconnaissance des balises au fur et à mesure que le robot les voit.

11.3.2 Présentation de l'algorithme

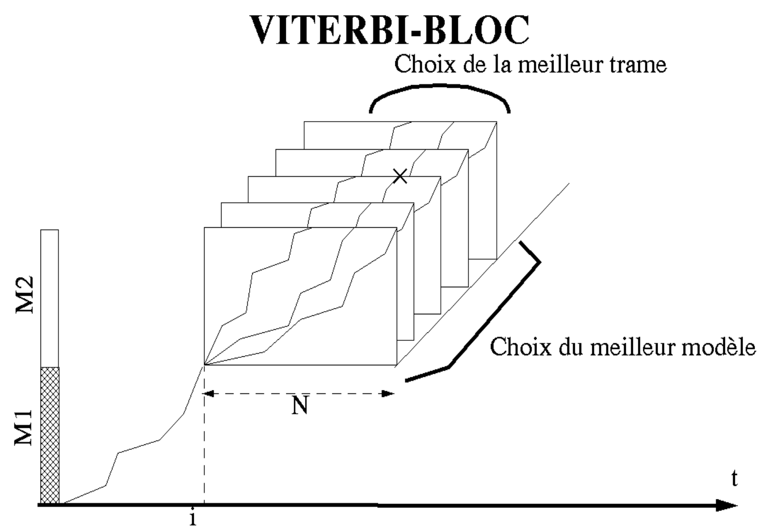


FIG. 11.8 – *Schéma de principe de l'algorithme de Viterbi Bloc*

L'algorithme de Viterbi-Bloc que nous proposons, provient de notre désir d'utiliser pendant la reconnaissance des critères basés sur une notion d'optimalité locale plutôt que globale [Haton, 1974]. Rappelons que l'algorithme de Viterbi, qui est un algorithme de programmation dynamique, construit et préserve à chaque instant, tous les chemins partiels s'achevant sur chaque état du modèle alors qu'un seul chemin, issu de l'état final, sera utilisé pour retrouver la suite des états empruntés. En procédant localement, les algorithmes mettent en œuvre des structures moins volumineuses puisque l'algorithme ne conserve qu'une partie des données nécessaires aux retours-arrières.

Cet algorithme effectue une segmentation simultanément avec la reconnaissance et donne une réponse en temps réel au fur et à mesure des acquisitions. Le principe général consiste à utiliser une fenêtre glissante (*cf.* figure 11.8) de N observations dans laquelle une décision est prise localement. Le processus de reconnaissance de Viterbi-bloc peut être décrit de la façon suivante (algorithme 11.1).

`ViterbiBloc (O, T, M, K ) :`

entrée : $O_1, O_2, \ldots, O_T$ la suite de T observations représentant ce que le robot a vu

sortie : $m_1, m_2, \ldots, m_K$ la suite de K modèles qui maximise le critère choisi

I $\leftarrow$1 (I est le debut de la fenêtre glissante de longueur N)

tant que $I + N \leq T$ **faire**

1) execution de l'algorithme de Viterbi entre I et $I + N$ pour chaque modele de reference
2) memorisation des probabilites d'alignement dans la fenêtre de comparaison pour chaque modele
3) comparaison de ces probabilites
4) choix du segment correspondant au modele favorable (soit t_1 son extremite) au sens d'un critere heuristique de choix predefini
5) eliminer tous les autres alignements
6) I $\leftarrow I + t_1 + 1$

fin-tant-que

Algorithme 11.1: Algorithme de Viterbi-bloc

Étape n° 1 : on applique l'algorithme de Viterbi entre les trames $[I,I+1]$, $[I,I+2]$, $\ldots [I,I+N]$.

Étape n° 2 : on sauvegarde les N probabilités d'alignement de chaque fenêtre.

Étape n° 3 : on étudie la dynamique de ces probabilités pour faire apparaître le meilleur modèle ainsi que la meilleure trame dans ce modèle. Soit t_1 cette trame. Les comparaisons sont faites à l'aide de critères heuristiques pour effectuer le choix du chemin localement optimal.

Étapes n° 4, 5 et 6 : on retient un modèle et on élimine les autres. Le processus continue à partir de la trame $I + t_1 + 1$.

De par la nature même des choix faits dans l'étape n° 3, l'algorithme de Viterbi-bloc est sous-optimal en ce sens qu'il ne garantit pas de retenir ainsi le chemin globalement optimal pour une séquence d'observations complète, c'est à dire, maximisant la probabilité à posteriori de la suite de modèles connaissant la séquence d'observations.

11.3.3 Utilisation pour la reconnaissance de balises

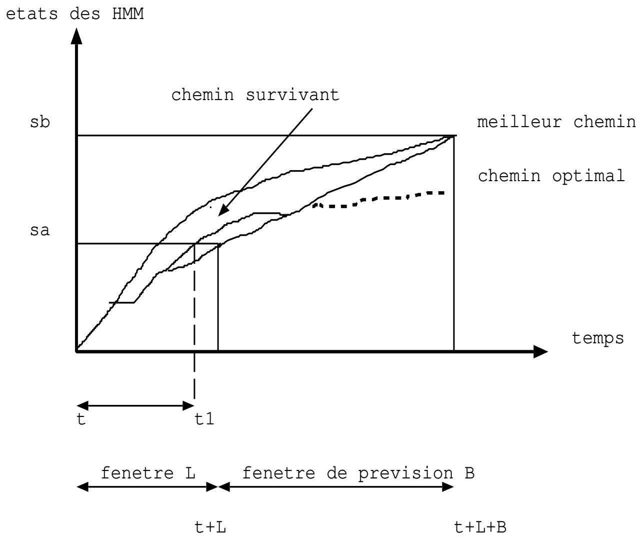


FIG. 11.9 – *Sélection du chemin optimal (d'aprés [Boulianne* et al., *1994])*

Nous avons adapté cet algorithme à la reconnaissance de balises naturelles. Dans une première passe, l'algorithme construit à l'aide de deux fenêtres : l'une, dite fenêtre de construction, de longueur L analogue à la fenêtre glissante de Viterbi bloc et l'autre de longueur L+B, dite fenêtre de prévision, dans laquelle une ou deux balises suivantes sont recherchées (figure 11.9). Il n'y a plus de comparaison de probabilités le long de l'arête supérieure de la fenêtre.

Supposons la reconnaissance effectuée jusqu'à la trame t. Nous appelons s_a l'état final de la balise suivante pour laquelle on cherche les frontières temporelles. Nous savons seulement que cette balise débute sur la trame t. Nous appliquons l'algorithme de Viterbi entre les trames $[t,t+1]$, $[t,t+2]$, $\ldots [t,t+L]$. Pour chaque trame de sortie $t+1$, $t+2$ $\ldots t+L$, alignée sur l'état final s_a, on poursuit le processus de construction à l'aide des balises suivantes jusqu'à la trame $t+L+B$. Soit s_b l'état possédant le meilleur score sur la trame $t+L+B$. Dans la fenêtre de prévision, nous venons de construire les chemins correspondants à deux balises supplémentaires.

A cet endroit, rien n'assure que l'on soit sur le chemin optimal donné par un classique algorithme de Viterbi. Mais, si la fenêtre de prévision est suffisamment grande –au plus, elle est égale à la portion restante de la séquence d'observations– on retrouvera le chemin optimal en reculant dans cette fenêtre car l'hypothèse expérimentale qui a motivé la spécification de l'algorithme de Viterbi-bloc est que le meilleur chemin s'achevant au point $(s_b,t+L+B)$ est optimal entre t et $t+L$. Finalement, on se retrouve à l'instant t_1 dans l'état s_a. t_1 est l'extrémité recherchée de la première balise qui s'achève sur l'état s_a. Le début de la fenêtre de construction est déplacé sur la trame t_1 et on continue le processus de prévision à partir de la trame t.

Expérimentalement, nous avons déterminé qu'une fenêtre de prévision équivalente à quarante observations simulées et une fenêtre glissante équivalente à cent observations réelles, permet de reconnaître avec sécurité les balises qui se trouve à plus de quarante observations géométriques (c-à-d, trois mètres) derrière le robot.

Exemple

La première colonne de la table de la figure 11.7 donne la position du robot (entouré d'un cercle) dans le couloir au moment où l'algorithme Viterbi-bloc a été exécuté. La deuxième colonne donne les balises reconnues à chaque fois que l'algorithme de Viterbi-bloc a été exécuté. La troisième colonne donne les balises reconnues par l'algorithme de Viterbi au moment où le robot a traversé le couloir (c-à-d, quand il se trouve en seizième position). Nous constatons que la séquence de balises reconnues par l'algorithme de Viterbi-bloc est la même que celle de reconnue par l'algorithme de Viterbi. Simplement, les balises reconnues par l'algorithme de Viterbi-bloc sont connues au minimum trois mètres (c-à-d, quarante observations simulées qui correspondent à la taille de la fenêtre de prévision) après avoir été vues, alors que dans le cas de l'algorithme de Viterbi, nous sommes obligés d'attendre que le robot atteigne la position seize (c-à-d, la fin de la traversée du couloir) pour reconnaître les balises.

11.4 Le module de localisation

11.4.1 Introduction

Comme nous l'avons dit dans la définition des modules de notre architecture de contrôle, le rôle du module de localisation est de permettre au robot de connaître sa position dans son environnement au fur et à mesure que les balises sont reconnues et interprétées, et de détecter qu'un but est atteint.

Il ne s'agit pas ici de déterminer une position précise du robot par rapport aux deux extrémités du couloir qu'il traverse, ni par rapport aux deux murs du couloir. Nous ne cherchons pas non plus à savoir l'orientation exacte du robot dans le couloir. Nous ne limitons ici à déterminer la position du robot par rapport aux deux extrémités du couloir. C'est en effet la seule information nécessaire pour déterminer si le robot a atteint son but. Pour cela, nous nous contentons de recaler l'odométrie de façon grossière à chaque fois qu'une balise est reconnue et interprétée pour éviter que l'estimée se dégrade trop au cours du temps. Par la suite, quand nous parlerons de position ou de localisation, ce sera toujours par rapport aux deux extrémités du couloir que le robot est en train de traverser.

11.4.2 Détection du but

A chaque instant, le robot doit savoir s'il a atteint son but. Pour cela, nous comparons la position du robot avec celle de son but. Le but est considéré comme atteint si la position du robot est proche de la position du but. A ce moment-là, le module de localisation poste un événement dans la mémoire partagée afin que le contrôleur d'exécution puisse arrêter le module de navigation.

11.4.3 Calcul de la position du robot au fur et à mesure que les balises sont reconnues et interprétées

Dans la section 11.2, nous avons vu comment à partir d'une séquence R de M balises reconnues, nous interprétons ces balises et déterminons à quelles balises elles correspondent en réalité. Nous avons vu dans la section 11.3 que l'algorithme de Viterbi-bloc reconnaît les balises au fur et à mesure qu'elles sont vues. Pour réaliser notre module de localisation, l'algorithme de Viterbi-bloc est exécuté toutes les cent acquisitions réelles (taille de la fenêtre glissante), et chaque balise reconnue est interprétée à l'aide du modèle de Markov de l'environnement et de l'algorithme "forward-backward". Connaissant les balises que le robot a vues, nous les utilisons pour le localiser au fur et à mesure de ses déplacements. Le seul problème est que Viterbi-bloc permet de reconnaître les balises au minimum trois mètres (c-à-d, quarante observations simulées qui sont égales à la taille de la fenêtre de prévision) après les avoir vues. Nous savons donc qu'à chaque fois que le robot reconnaît une balise, il se trouve au moins à trois mètres de cette balise. Connaissant la position de chaque balise de l'environnement, nous pouvons facilement déterminer la position réelle du robot au moment où il la reconnaît. Grâce à cette information, nous déterminons où se trouve le robot au fur et à mesure qu'il reconnaît et interprète les balises sachant qu'il a parcouru une distance d'au moins trois mètres depuis l'instant où il a reconnu la dernière balise. Pour obtenir une position du robot plus précise, à chaque acquisition, en plus de stocker les valeurs des capteurs, nous mémorisons la position estimée correspondante du robot. Lors de la reconnaissance, nous savons à quelles observations correspond chaque balise reconnue, nous pouvons facilement en déduire la position estimée du robot au moment où il a reconnu une balise. Comme nous connaissons la position estimée actuelle du robot (donnée par l'odométrie), nous en déduisons à partir de la position estimée du robot au moment où il a reconnu la dernière balise une estimée de la distance parcourue. Nous déterminons la position actuelle réelle du robot à partir de la position réelle du robot au moment où il a reconnu une balise et de l'estimée de la distance parcourue, en supposant qu'il a parcouru cette distance depuis la reconnaissance de la balise.

Cette position ainsi que l'état (visible ou non visible) de la balise interprétée est postée dans la mémoire partagée du niveau physique. De plus, cette nouvelle position permet de recaler l'odométrie et devient donc la nouvelle estimée de la position du robot.

Exemples

Dans la figure 11.10, nous montrons un exemple de localisation du robot suite à une exécution de l'algorithme de Viterbi-bloc. Au moment où l'algorithme de Viterbi-bloc est exécuté, le robot possède une estimée de sa position (position 1). De plus, connaissant les observations qui lui ont servi à reconnaître cette balise, il possède une estimée de la position où il se trouvait au moment où il a reconnu la balise (position 2). Il en déduit donc une estimée de la distance qu'il a parcourue depuis qu'il a reconnu la balise. Puis à partir de la position réelle où il se trouvait au moment de la reconnaissance (position 3), et de l'estimée de la distance parcourue nous déterminons la position réelle actuelle du robot (position 4).

Revenons à la colonne six de la figure 11.7. Cette colonne représente le recalage au

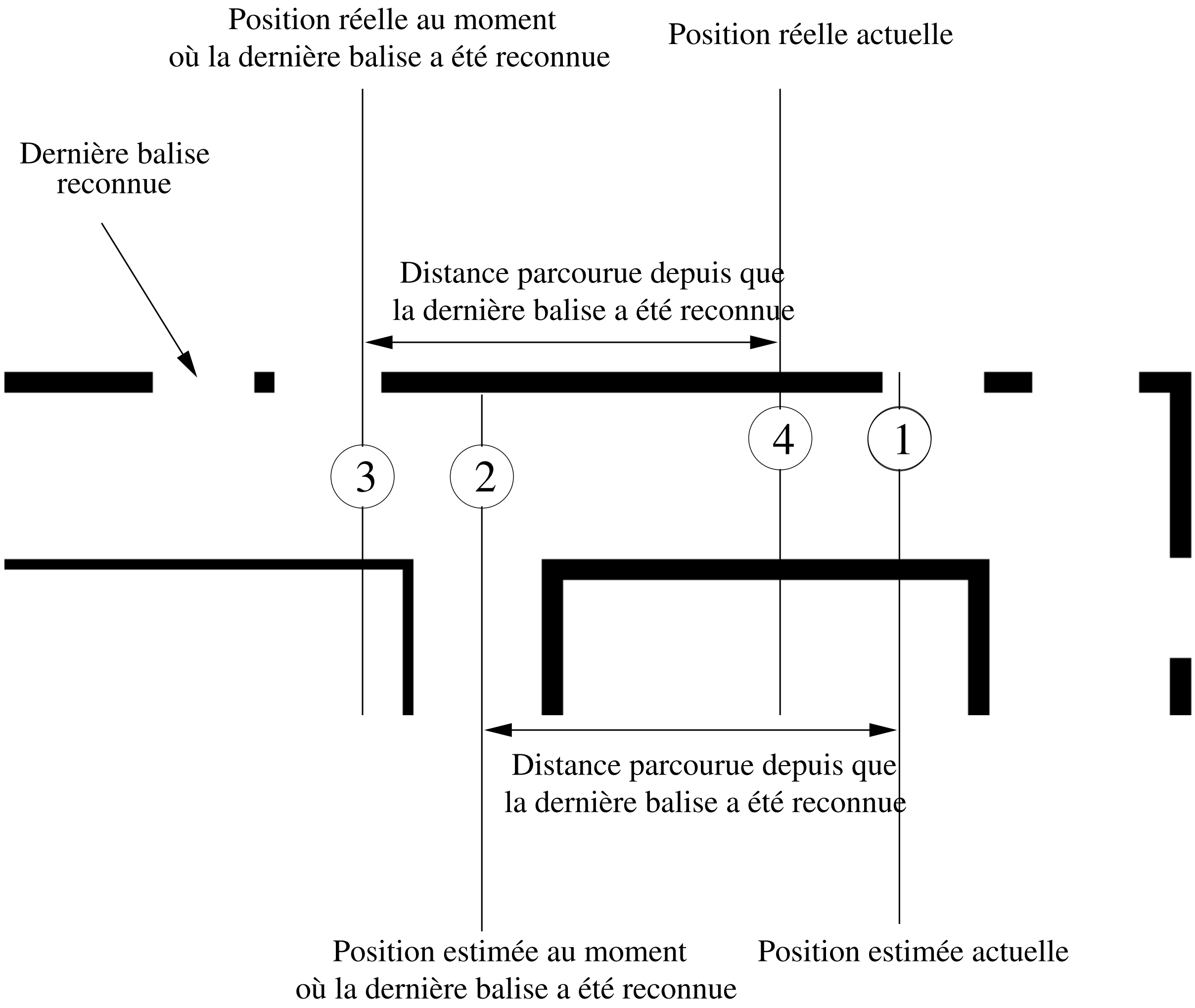


Fig. 11.10 – *Exemple de localisation*

fur et à mesure que le robot reconnaît et interprète les balises de la quatrième colonne de cette même figure. Nous remarquons qu'à chaque fois que les balises reconnues ont été correctement interprétées le recalage est faible (c-à-d, inférieur à trente centimètres). Quand deux balises reconnues et correctement interprétées sont très proches l'une de l'autre, la dérive odométrique est quasiment nulle, donc le recalage est faible. C'est par exemple le cas lors du recalage de la suite de bureau B114, B116 et B103. Les trois balises représentant ces bureaux sont reconnues et bien interprétées l'une après l'autre, et la distance parcourue entre chaque reconnaissance est faible, la dérive odométrique est donc très faible. Par contre quand l'interprétation est mauvaise, le recalage est important : c'est par exemple le cas lorsqu'à la position 3, le robot reconnaît une porte ouverte à gauche et l'interprète comme la porte du bureau B108 qui est ouverte alors que cette reconnaissance correspond à une insertion. De manière générale, le recalage (quand il est bon) est presque toujours positif, ceci vient du fait que l'odométrie a tendance à délivrer une position qui se trouve à quelques centimètres derrière le robot.

Résultats

Pour évaluer notre approche, nous avons vérifié sur plusieurs déplacements si le robot s'arrête devant la balise correspondante à son but à atteindre. Il ne s'arrête que très rarement devant une autre balise. D'autre part, dans la majorité des cas la position du robot donnée par notre module de localisation est à moins de quinze centimètres de la position réelle du robot. En effet, la position du robot est souvent recalée puisque quasiment tous les trois mètres il a une nouvelle balise à sa disposition pour se localiser. En trois mètres, l'estimée ne se dégrade qu'assez peu, et donc entre deux interprétations des balises de l'environnement, l'estimée donnée par notre module de localisation reste satisfaisante. D'autre part, l'estimée de la distance parcourue est suffisamment fiable pour savoir où il se trouve au moment où il réalise la reconnaissance, connaissant la position où il se trouvait au moment il a reconnu la dernière balise par rapport à laquelle il se recale.

Le problème majeur provient des balises reconnues qui sont mal interprétées. Comme nous l'avons dit dans la section 11.2, certaines balises reconnues sont interprétées comme des balises visibles alors qu'en fait, elles ne le sont pas. Ces balises provoquent des recalages qui n'ont aucun sens, puisqu'en fait la position estimée au moment où le robot reconnaît la balise ne correspond pas du tout à la position estimée où il se serait trouvé s'il avait pu reconnaître la balise. L'estimée de la distance parcourue depuis la reconnaissance sera fausse, et la position réelle du robot délivrée, dégradera la position estimée plutôt que de recaler. Pour résoudre ce problème, nous avons testé deux solutions qui paraissent aussi valables l'une que l'autre :

– Nous avons remarqué que dans la majorité des cas, ce genre de recalage est relativement plus important (c-à-d, il y a une grosse différence entre la position estimée actuelle délivrée par l'odométrie et la position actuelle délivrée par le module de localisation) qu'un recalage par rapport à une balise reconnue qui a été bien interprétée. En effet, comme nous l'avons dit précédemment un recalage par rapport à une balise reconnue et bien interprétée est souvent minime (moins de quinze centimètres). Nous pouvons donc détecter les balises reconnues et mal interprétées puisqu'elles provoquent des recalages beaucoup plus important que celles qui sont bien interprétées. Il suffit pour cela de fixer un seuil de recalage au dessus duquel nous considèrons que le recalage est probablement faux.

– Le recalage est réalisé à chaque fois qu'une nouvelle balise est reconnue et interprétée, ce qui se produit environ tous les trois mètres. Si un recalage est faux à cause d'une mauvaise interprétation, nous pouvons le laisser, en supposant que de toute façon dès qu'une balise reconnue sera bien interprétée nous pourrons recaler correctement le robot.

11.4.4 Conclusion

Dans cette section, nous avons décrit le fonctionnement de notre module de localisation. La localisation est réalisée en quatre étapes :

1. A partir du but à atteindre et de la carte topologique de l'environnement, nous construisons un modèle de Markov caché du premier ordre représentant toutes les séquences de balises que le robot peut voir lors de son déplacement vers le but.
2. A chaque fois que le robot a réalisée cent acquisitions réelles, l'algorithme de Viterbi-bloc est exécuté sur l'ensemble des observations non encore reconnues. Cette première phase permet de reconnaître les balises au fur et à mesure que le robot évolue dans son environnement.
3. Pour chaque balise reconnue, nous utilisons le modèle de Markov défini dans l'étape 1 et l'algorithme "forward-backward" pour déterminer à quelle balise de l'environnement elle correspond.
4. Connaissant la position de la balise de l'environnement que le robot vient de reconnaître, et possédant une estimée de la distance qu'il a parcouru depuis cette reconnaissance, nous déterminons sa position dans l'environnement.

A proprement parler, le module de localisation réalisée est plus un module de recalage périodique de la position estimée délivrée par l'odométrie qu'un "réel" module de localisation. En effet, à chaque fois qu'une balise est reconnue et interprétée, à partir de sa position dans l'environnement et de la distance parcourue depuis cette reconnaissance, nous recalons l'odométrie. D'autre part, nous ne nous intéressons qu'à la position du robot par rapport aux extrémités du couloir qu'il est en train de traverser.

Comme nous l'avons dit ce recalage périodique est suffisant pour le type de localisation que nous recherchons et permet de connaître la position du robot avec une erreur de moins de quinze centimètres.

11.5 Le contrôleur d'exécution du niveau physique

Lors d'un déplacement, le rôle du contrôleur d'exécution est de déterminer à chaque instant la meilleure action à exécuter par le robot. C'est en effet ce module qui gère la réactivité tactique.

Lors de l'exécution d'un déplacement nous avons vu dans la section 11.4 qu'au fur et à mesure que les balises sont reconnues et interprétées, la position du robot dans son environnement est postée par le localisateur dans la mémoire partagée. A ce moment-là, le contrôleur d'exécution d'actions fournit cette position au module de navigation et lui demande à nouveau d'atteindre le but. Si le robot ne se trouve plus face au but, le module de navigation l'arrête et le place à nouveau face au but et dans le cas contraire il continue simplement à déplacer réactivement le robot vers son but.

Ce processus se repète à chaque fois que la position du robot est postée dans la mémoire partagée. Au moment où le but est atteint le localisateur poste un compte rendu d'exécution dans la mémoire partagée, et le contrôleur d'exécution demande au navigateur d'atteindre le but suivant fourni par le planificateur de chemins. Ce processus permet au robot d'atteindre une suite de sous-buts le menant d'une position initiale à une position finale tout en gérant les incertitudes.

Au delà de cet interaction simple, la gestion des déplacements du robot correspond tout à fait à l'interaction entre la planification et l'exécution des déplacements réalisées à l'aide d'un POMDP.

En effet, dans la phase d'interprétation des balises reconnues, nous pouvons associer à chaque état de l'environnement la probabilité que le robot a de s'y trouver, ce qui correspond tout à fait aux états possibles[29] d'un POMDP, et le modèle de Markov représentant l'environnement correspond à la fonction d'observation d'un POMDP. Simplement nous n'avons pas de fonction de transitions, puisque nous considérons que le module de navigation fait avancer le robot réactivement devant lui, et que les beliefs states sont mis à jour à chaque fois qu'une balise est reconnue et interprétée. Comme nous l'avons vu dans la section précédente, depuis le moment où le robot a reconnu la dernière balise, il a parcouru une certaine distance. En tenant compte de cette distance parcourue, nous pouvons déterminer les positions possibles du robot dans l'environnement ainsi que leur vraisemblance. Il suffit donc à ce moment là de déterminer une stratégie qui permet de choisir l'action à effectuer en fonction de ces états possibles. Dans la littérature, nous avons vu que la stratégie généralement retenue est de choisir l'action associée à l'état le plus probable, et c'est en fait celle que nous utilisons : à chaque interprétation de balise, seule la position du robot correspondante à l'état le plus probable est retenue et postée dans la mémoire partagée. Une fois cette position postée, le navigateur détermine automatiquement s'il se trouve toujours face au but, ce qui permet de déterminer la meilleure action à réaliser pour atteindre le but. De plus, notre approche est relativement proche de [Simmons et Koenig, 1995] et [Nourbakhsh *et al.*, 1993] : notre environnement est représentée sous forme d'une carte topologique (assimilable à un POMDP), et nous avons vu que pour définir la liste des sous-buts permettant d'atteindre un but donné, nous utilisons l'algorithme A*. L'atteinte de ces buts et la prise en compte des incertitudes lors des déplacements sont gérées par l'interaction entre le localisateur qui donne périodiquement la position où le robot a la plus grande probabilité de se trouver, et à partir de là le contrôleur d'exécution fournit cette nouvelle position au navigateur et lui demande d'atteindre à nouveau le but. Le navigateur vérifie par la même occasion s'il se trouve toujours face au but.

Contrairement à [Simmons et Koenig, 1995] et [Nourbakhsh *et al.*, 1993], notre fonction d'observation est basée sur l'évaluation des taux de reconnaissance des balises et pas sur des constatations empiriques.

29. belief states

Exemple

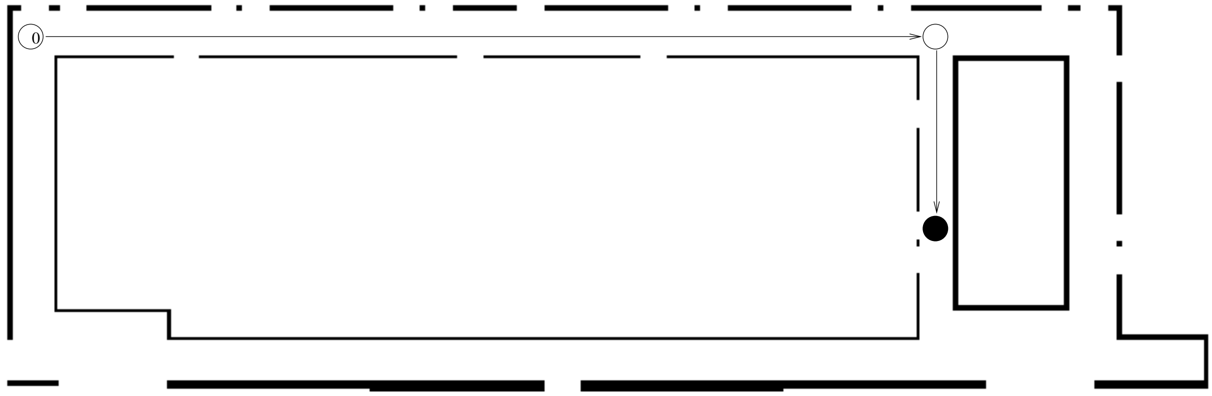


Fig. 11.11 – *Liste de sous-buts à atteindre*

La figure 11.11 montre la liste de sous-buts à atteindre par le robot pour se déplacer de la position 0 (sa position initiale) à la position finale (cercle noir).

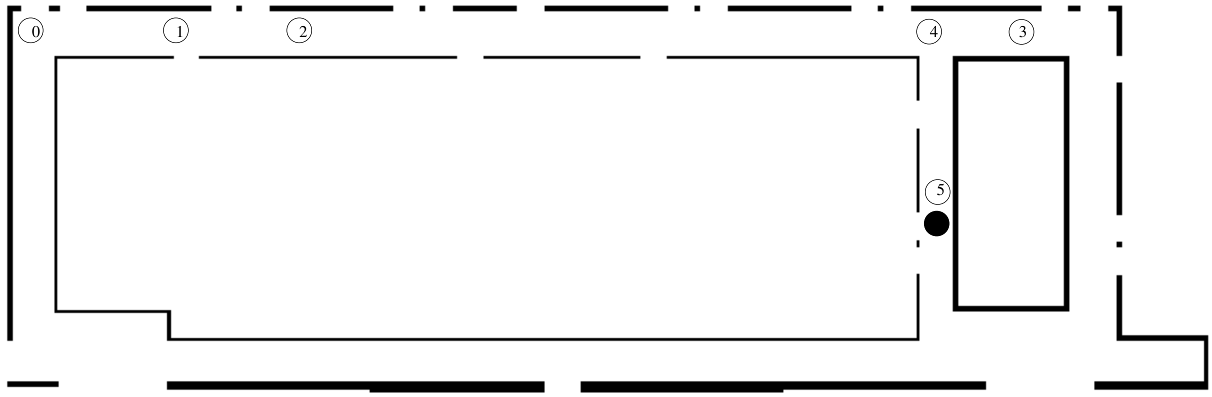


Fig. 11.12 – *Exécution du plan correspondant*

Examinons maintenant l'exécution de ce plan. Supposons de plus que la position initiale du robot soit fausse, mais que le robot se trouve bien entre l'extrémité gauche du couloir supérieur et l'intersection du couloir supérieur. Pour l'interprétation des balises reconnues, nous considérons la partie du couloir supérieur que le robot va traverser. La première action à réaliser est de se déplacer vers la droite. La première balise que le robot reconnaît est une porte ouverte à gauche, et l'interprétation de cette balise lui permet de déterminer sa position (position 1 dans la figure). Le robot continue donc à se déplacer dans la même direction. Il reconnaît ensuite un couloir, et d'après l'interprétation de cette balise, il se trouve dans la position 2, et continue à se déplacer dans la même direction. Il reconnaît ensuite une intersection à droite, et se trouve à ce moment là dans la position 3. Le robot ne se trouve à ce moment là plus face au but. Le navigateur fait donc faire demi-tour au robot pour le placer face à son sous-but (c-à-d, l'intersection dans ce cas-là). Le robot se déplace ensuite jusqu'à l'intersection (position 4). Il s'y arrête puisqu'elle correspond au sous-but à atteindre. A ce moment-là, le contrôleur d'exécution demande au navigateur d'atteindre le sous-but suivant. Le navigateur fait donc tourner le robot de quatre-vingt dix degré dans le sens des aiguilles d'une montre pour le placer face à son sous-but. le robot reconnaît ensuite une porte à droite, qui lui permet de déduire qu'il est dans la position 5, et il continue à avancer pour atteindre le but final (point noir), et s'arrête lorsque le localisateur a detecté qu'il l'avait atteint.

11.6 Conclusion

Pour une utilisation en temps réel sur notre robot mobile, notre approche posait deux problèmes. D'une part, l'algorithme de Viterbi ne permet une reconnaissance des balises qu'une fois la traversée du couloir terminée. D'autre part, la comparaison entre ce que le robot a reconnu et ce qu'il a vu n'est plus possible, puisque nous ne savons pas *à priori* l'état du couloir. Ces deux problèmes nous ont amené à proposer une modification de l'algorithme de Viterbi pour que les balises soient reconnues au fur et à mesure qu'elles sont perçues. De plus, nous avons construit un modèle de Markov qui permet en fonction de la séquence de balises reconnues de déterminer l'état de l'environnement qui correspond le mieux à cette séquence. L'utilisation de l'algorithme de Viterbi-bloc, et de l'algorithme "forward-backward" nous a permis de reconnaître et d'interpréter les balises au fur et à mesure qu'elles sont vues par le robot, et ainsi de déterminer la position du robot.

Dans la dernière section, nous avons expliqué comment interagissent le localisateur, le contrôleur d'exécution et le navigateur pour permettre au robot d'atteindre un but tout en gérant les incertitudes. Cet interaction constitue un POMDP simplifié qui est tout à fait suffisant pour gérer l'incertitude lors des déplacements du robot dans les couloirs du laboratoire. L'incertitude de perception est gérée à l'aide de modèles de Markov cachés du second ordre qui intègre les données bruitées des capteurs à l'apprentissage et à la reconnaissance. L'incertitude de contrôle est gérée par le module de navigation. Gaston évoluant dans un environnement quasiment exclusivement constitué de couloirs, nous n'avons pas besoin de techniques complexes et de fonction de gains pour planifier ses déplacements : le simple algorithme A* et une carte topologique de l'environnement sont suffisants pour déterminer les sous-buts à atteindre. De plus, le module de navigation assure que le robot atteint son but à moins qu'un obstacle bouche le couloir, nous n'avons donc pas besoin d'une fonction de transition. L'interprétation des balises reconnues permet d'obtenir la position la plus vraisemblable du robot, et à partir de là de vérifier le bon déroulement du déplacement vers le but. A chaque fois que la position du robot est réactualisée, le module de navigation vérifie que le robot se trouve toujours face à son but. S'il a dépassé son but, il le replace face à lui, et dans le cas contraire le robot continue à se déplacer réactivement vers son but.

Nous constatons donc que notre approche pour l'apprentissage et la reconnaissance de balises est à la base du contrôle d'exécution d'actions. Elle permet au robot de vérifier qu'il se trouve toujours sur le chemin le menant à son but, et le cas échéant lui permet de faire demi-tour. Cette technique simple permet de gérer l'exécution d'actions pour atteindre un but. Dans un cadre plus général, nous avons vu que notre approche peut être étendue pour apprendre et reconnaître des balises dans d'autres types d'environnement, et qu'elle peut en plus déterminer pour chaque balise reconnue l'environnement associé, et ainsi utiliser automatiquement les techniques de planification et d'exécution d'actions les plus appropriées selon le contexte. Nous avons montré que dans un environnement composé de couloirs, notre approche était rapide et fiable. Nous pensons que dans d'autres contextes d'autres techniques seraient plus appropriées. Par exemple, dans un environnement constitué de pièces (voire de halls), nous pourrions étendre notre approche pour réaliser un véritable POMDP pour planifier et exécuter les actions du robot en prenant en compte une fonction de gain et une fonction de transition. D'autre part, nous pourrions améliorer notre module de navigation pour prendre en compte un but précis à atteindre dans ce type d'environnement, ou définir une méthode pour construire la fonction de transition en considérant le taux d'échec du module de navigation dans une pièce lorsque le robot doit atteindre un but précis.

Troisième partie

Conclusion et perspectives

12

Conclusion et perspectives

Le but de notre travail était d'utiliser des modèles de Markov cachés pour apprendre et reconnaître des balises naturelles. Nous avons ensuite appliqué notre approche à la localisation d'un robot mobile et au contrôle d'exécution d'actions.

Pour réaliser ce travail, nous avons développé une architecture complète de contrôle pour robot mobile. Cette architecture inspirée d'AIBOT, est basée sur la décomposition du contrôle en niveaux. Chaque niveau est spécialisé dans la réalisation d'une tâche. Il est pour cela autonome. La structure de chaque niveau est identique. Il est composé d'un contrôleur qui gère l'activation des composantes du niveau, d'une mémoire partagée qui contient les événements relatifs au niveau, et d'un planificateur qui décompose les tâches du niveau. De plus, chaque niveau dispose de composantes spécifiques qui lui permettent de réaliser la tâche à laquelle il est dédié. La définition des niveaux se fait de manière incrémentale. Chaque niveau peut ainsi être développé et testé indépendamment des autres. La circulation des informations se fait de manière ascendante et descendante entre deux niveaux consécutifs. Le niveau le plus haut dans la hiérarchie envoie des actions à exécuter au niveau juste inférieur, et le niveau inférieur renvoie le résultat de l'exécution de ces actions.

Dans notre cas, nous avons développé une architecture de contrôle à deux niveaux : un niveau cognitif et un niveau physique.

- Le niveau cognitif décompose les missions du robot et gère leurs exécutions. Il reçoit ses missions de l'utilisateur. Il manipule des données symboliques. Le planificateur de ce niveau est un planificateur de missions, qui interprète et décompose en actions élémentaires les missions du robot, et les envoie au niveau inférieur. Il reçoit à la fin de l'exécution d'une action un compte rendu d'exécution du niveau immédiatement inférieur. En fonction de ce compte rendu, le contrôleur d'exécution de ce niveau détermine comment continuer la mission. Tout au long de l'exécution d'une action, le contrôleur d'exécution reçoit deux types d'informations sur l'environnement fourni par le niveau physique : des informations sur l'état des balises (visibles ou non visibles), et sur la position des obstacles découverts lors de l'exécution d'une action.
- Le niveau physique exécute les actions et gère leurs exécutions. Ce niveau reçoit les actions à exécuter du niveau cognitif, et lui renvoie des comptes rendus d'exé-

cution des actions. Le planificateur de ce niveau sert à trouver une liste de sous but à atteindre par le robot pour se déplacer d'un endroit à un autre dans l'environnement. Afin d'exécuter ces actions, ce niveau dispose de plusieurs modules : un module de navigation qui permet au robot de se déplacer de manière réactive dans son environnement, un module de localisation qui permet au robot de connaître sa position dans son environnement, et un module pour saisir et rendre les objets. Le localisateur et le module de saisie et de rendu d'objet postent dans la mémoire partagée du niveau des comptes rendus d'exécution. En fonction de ce compte rendu, le contrôleur détermine si l'action a été exécutée correctement. Si c'est le cas, il exécute l'action suivante postée dans la mémoire partagée du niveau physique par le niveau cognitif, et dans le cas contraire, il fait appel au niveau cognitif pour replanifier ou abandonner la mission.

Au-delà de notre travail, l'étude menée a permis de mettre en évidence les besoins et exigences à respecter pour réaliser une architecture de contrôle. D'autre part, notre architecture propose un cadre d'intégration idéal pour les modules qui à l'avenir seront développés dans l'équipe.

Cette architecture peut être améliorée de nombreuses façons :

- Nous pouvons y ajouter de nouveaux modules pour augmenter les capacités de chacun des niveaux. L'ajout d'un module permettant de prendre en compte des échéances pour la réalisation des actions et des missions ou un module qui à partir des événements découverts lors de l'exécution d'une mission pourrait générer de nouvelles tâches à accomplir. Nous pouvons aussi penser à un module d'exploration de l'environnement ou de construction de cartes topologiques.
- D'autre part, nous envisageons l'ajout d'un troisième niveau capable de coordonner divers robots effectuant une mission.
- L'implantation des divers modules dans un niveau pourrait être facilité grâce à un langage de spécification qui permettrait de savoir comment intégrer un nouveau module, en spécifiant sa fonction dans l'architecture, ses entrées et ses sorties, et les interactions qu'il a avec les autres modules.
- A l'heure actuelle, les événements sur l'état des balises de l'environnement et les obstacles découverts par le robot lors de l'exécution ne sont absolument pas utilisés. Ils sont simplement transmis au niveau cognitif par le contrôleur du niveau physique. Nous pouvons imaginer de prendre en compte ces événements tant au niveau cognitif qu'au niveau physique. Nous pourrions par exemple mettre à jour la carte de l'environnement. Les obstacles découverts y seraient automatiquement ajoutés, et lors de la planification d'un chemin, nous pourrions tenir compte de ces obstacles, et privilégier certains chemins par rapport à d'autres. De la même façon pour les balises vues pendant l'exécution des missions, nous pourrions lors de la planification de missions tenir compte que certaines pièces de l'environnement sont inaccessibles car les portes qui y mènent sont fermées.

Pour réaliser notre approche, nous avons aussi dû nous intéresser à la définition d'un module de navigation répondant à deux contraintes majeures : permettre à notre robot de se déplacer dans les couloirs du laboratoire tout en évitant les obstacles, et éviter que le robot fasse des rotations trop importantes afin que chaque capteur observe à peu près toujours la même partie de l'environnement, pour observer les variations des capteurs dans le temps, et s'en servir pour apprendre et reconnaître des balises naturelles. Pour cela, le robot se place tout d'abord face au but, en vérifiant que le bras manipulateur ne risque pas de toucher un obstacle ou un mur lorsque la tourelle va tourner. Dans le cas où le robot se trouve déjà face à son but, il ne fait rien.

Pour que Gaston atteigne son but tout en évitant les obstacles, nous avons développé un contrôleur flou à deux niveaux. Notre approche a l'avantage de permettre un développement simple d'un contrôleur flou. Au premier niveau, des zones locales et des règles locales sont définies pour concevoir des comportements locaux. Ces règles sont définies à partir du comportement qu'aurait un être humain placé dans la même situation. Au deuxième niveau, un mécanisme d'intégration de ces comportements locaux fournit une action motrice globale au robot mobile. De plus, les comportements locaux peuvent être conçus et testés indépendamment et les mécanismes d'intégrations proposés donnent une cohérence globale au robot. Dans un deuxième temps, nous avons ajouté un comportement de prise en compte d'un but précis. Ce comportement a permis de mettre en place plusieurs mécanismes d'intégration des divers comportements. En effet, les comportements d'évitement d'obstacles sont fusionnés afin de définir le comportement global du robot lorsqu'il évite un obstacle, et sont en compétition avec le comportement de prise en compte du but qui n'est actif que lorsque le robot n'est pas en train d'éviter un obstacle. Malheureusement, cette prise en compte du but est conditionné par l'odométrie ce qui pose problème.

Notre approche a été implantée et testée dans de nombreuses situations. Le robot est capable de faire face à diverses situations d'évitement d'obstacles. L'intégration des divers comportements locaux d'évitements d'obstacles, lui permet de se décaler et de longer un obstacle lorsqu'il le gêne et même de longer un obstacle en naviguant en crabe dans les cas d'urgence. Ces comportements locaux lui permettent d'éviter les obstacles dans la grande majorité des cas. De plus, notre contrôleur donne de bons résultats dans d'autres situations tel que les rétrécissements de couloir ou les passages de portes ouvertes. Il faut aussi noter que notre contrôleur flou est actuellement utilisé par tous les membres de l'équipe travaillant sur le robot.

Notre approche souffre encore de quelques problèmes. Le robot est incapable d'atteindre son but, lorsqu'un obstacle en U se trouve sur son chemin. De plus, les capteurs situés à l'arrière du robot ne sont pas utilisés. Ces capteurs sont justement souvent utilisés pour permettre au robot de faire une marche arrière ou un demi tour lorsqu'il est bloqué.

Dans la deuxième partie du mémoire, nous avons présenté notre approche pour apprendre et reconnaître des balises naturelles à l'aide de modèles de Markov cachés. Le but de notre travail a été de montrer que les HMMs sont tout à faits aptes à apprendre et à reconnaître des balises naturelles dans un environnement intérieur structuré. Les HMMs sont en effet très intéressants puisqu'ils se situent à mi chemin entre une approche où l'on cherche à expliquer les observations, et une approche où l'on cherche à déduire des propriétés statistiques des observations, puisqu'ils apprennent la suite d'événements temporels qui caractérisent une balise naturelle. Pour réaliser notre travail, nous nous sommes servis d'un outil développé dans l'équipe à base de HMM2 avec des observations continues. Pour l'appliquer à notre problème, nous avons dû constituer un corpus d'apprentissage, le segmenter et l'étiqueter pour pouvoir entraîner chaque modèle de balise. Nous avons ensuite mis en place une procédure pour évaluer le taux de reconnaissance en comparant ce que le robot reconnaît lors d'un déplacement avec ce qu'il a réellement vu.

Nous avons dans une première étude construit un modèle de cinq balises à partir d'observations réelles, et évaluer le taux de reconnaissance global de ces cinq modèles. Cette première étude donne de bons résultats (75% de reconnaissance globale), et l'analyse de ces résultats permet de comprendre quels sont les facteurs limitants le taux de reconnaissance. Dans un deuxième temps, nous avons donc tenu compte des premiers résultats pour modifier la façon d'apprendre et de reconnaître une balise. Ces modifications portent essentiellement sur deux critères : le nombre de capteurs à prendre en compte lors des phases d'apprentissage et de reconnaissance, et la simulation d'une fréquence constante d'acquisitions des mesures des capteurs fonction de la distance parcourue par le robot. Le taux de reconnaissance final obtenu est très bon (90%). De ce travail, nous pouvons tirer plusieurs enseignements :

– L'apprentissage intègre le bruit des capteurs lorsque le robot passe devant une balise, et la reconnaissance met en compétition tous les modèles, sachant que seul le modèle le plus probable est retenu. Ces deux propriétés permettent à la fois de ne pas avoir à traiter les données des capteurs avant apprentissage et reconnaissance, et permettent une reconnaissance des balises même si les données des capteurs sont bruitées ou si la balise à reconnaître n'est pas parfaitement observable, ce qui rend la reconnaissance beaucoup plus robuste. De plus, le choix des variations des mesures des capteurs comme composante de l'observation permet d'utiliser notre approche dans des couloirs de largeurs différentes.
– Pour constituer un corpus d'apprentissage pour un modèle de balise, nous définissons quelques règles simples pour segmenter et étiqueter des corpus contenant une suite de balises à reconnaître. Nous constatons que l'apprentissage des modèles permet d'extraire des caractéristiques beaucoup plus complexes que la simple variation d'un ensemble de capteurs dans le temps, telles que par exemple la longueur de la balise à reconnaître. Cette propriété des modèles de Markov caché permet l'apprentissage et la reconnaissance de nombreuses autres balises.

Nous pouvons envisager deux extensions majeures à notre approche :

– Etendre le nombre de balises, afin d'avoir des balises plus spécifiques. Les balises “portes ouvertes” ou “portes en vis en vis” peuvent correspondre à plusieurs balises, où l'on tient compte du sens d'ouverture de la porte, et du fait qu'elles sont complètement ou partiellement ouvertes, et pour les portes en vis à vis, du fait qu'elles se chevauchent beaucoup ou peu, ou du fait que la porte située à droite doit être reconnue avant celle située à gauche ou inversement. Nous pouvons aussi ajouter un ou plusieurs modèles d'obstacles en tenant compte du sens d'évitement (droite ou gauche) de l'obstacle, ou même un modèle de rétrécissement de couloirs. De plus, ces extensions permettraient sans aucun doute d'améliorer encore le taux de reconnaissance. A partir du corpus déjà existant, ces extensions reviennent en fait à définir de nouvelles balises, ainsi que les règles qui permettent d'en tenir compte lors de la segmentation. Pour terminer il faudra refaire la segmentation du corpus, ainsi que l'apprentissage de chaque balise, même celles déjà apprises puisque leur corpus d'apprentissage ont changé.

– De manière plus générale, nous envisageons d'utiliser notre approche pour apprendre et reconnaître des balises que l'on trouve dans d'autres types d'environnements intérieurs tels que par exemple une pièce ou un hall. Nous avons déjà défini des modèles de coin dans une pièce, d'éloignement ou de rapprochement d'un mur. En plus de cela, nous pouvons ajouter les mesures des capteurs en plus des variations de ces mesures comme composante de l'observation afin de reconnaître en plus de la balise le type d'environnement dans lequel se trouve le robot. En effet, lorsque le robot se trouve dans un couloir (resp. une pièce, resp. un hall), les valeurs des capteurs n'intervenant pas dans la caractérisation des balises sont faibles (resp. moyennes, resp. élevées).

La constitution du corpus d'apprentissage des balises constitue le point le plus délicat. En effet, pour entraîner un modèle de balise, il faut un grand nombre de séquences d'observation de cette balise. L'acquisition des données est très longue. Cette phase est inévitable, et ne peut être raccourcie. De plus, la segmentation et l'étiquetage sont réalisées manuellement. La seule idée que nous ayons pour automatiser un peu ce processus est de réaliser manuellement la segmentation d'une partie du corpus, puis de passer à l'apprentissage et ensuite reconnaître la partie restante du corpus à l'aide des modèles entraînés. Cette reconnaissance constituerait une présegmentation du corpus qui permettrait de ne plus qu'à avoir à contrôler la présegmentation de chaque déplacement du robot.

Dans le dernier chapitre, nous avons appliqué notre approche pour définir deux modules de notre architecture de contrôle : un module de localisation et le contrôleur d'exécution du niveau physique. Nous avons tout d'abord expliqué comment utiliser notre approche en temps réel pour reconnaître les balises au fur et à mesure qu'elles sont vues. Pour cela, nous avons utilisé l'algorithme de Viterbi-bloc qui effectue une segmentation simultanément avec la reconnaissance et donne une réponse en temps réel au fur et à mesure des acquisitions. Expérimentalement nous avons déterminé qu'en effectuant une reconnaissance à chaque fois que le robot avait fait cent acquisitions réelles, nous pouvions déterminer les balises qui se trouvent à plus de trois mètres de lui (ie, quarante acquisitions simulées). Nous avons aussi montré comment interpréter les balises reconnues et déterminer à quelle balise de l'environnement elles correspondent. Pour cela, nous avons défini un modèle de Markov caché dont la topologie représente l'ensemble des séquences de balises que le robot peut voir lors d'un déplacement, et étant donné la séquence de balises reconnues, nous avons déterminé la séquence d'états (càd, de balises) du modèle qui lui correspond le mieux (càd, qui est la plus probable). Cette séquence d'états représente l'état de l'environnement que le robot a vu, et permet ainsi de déterminer pour chaque balise reconnue à quelle balise précise de l'environnement elle correspond. Cette interprétation donnent de bons résultats puisque nous savons à quelle balise de l'environnement correspond chaque balise reconnue avec une confiance de 81%.

L'utilisation de ce modèle pour interpréter les balises reconnues et de l'algorithme de Viterbi-bloc permet au robot d'interpréter les balises au fur et à mesure qu'il les reconnaît. Connaissant la position de chaque balise dans l'environnement et possédant une estimée de la distance parcourue par le robot depuis le moment où il a reconnu une balise, nous pouvons déterminer la position du robot par rapport aux extrémités du couloir qu'il est en train de traverser avec une erreur de moins de trente centimètres. Cette position est à chaque instant comparée avec la position du but à atteindre, afin de déterminer si le robot a atteint son but. Pour terminer, nous avons détaillé le fonctionnement du contrôleur d'exécution d'actions dans le cas où le robot se déplace dans son environnement. En effet, la façon dont il va gérer la réactivité tactique est grandement liée à la position du robot que va lui fournir le module de localisation, à chaque fois qu'une balise est reconnue et interprétée. Son rôle est de déterminer à chaque instant, la meilleure action à exécuter par le robot pour qu'il atteigne son but.

A chaque fois que la position du robot est postée dans la mémoire partagée le contrôleur d'exécution du niveau physique doit vérifier que le robot se déplace toujours en direction de son but. Pour cela, il demande au navigateur d'atteindre à nouveau le but. Au moment du placement face au but si le navigateur se rend compte qu'il n'est plus face au but, alors il replacera le robot face au but. Cette procédure très simple permet de prendre en compte la réactivité tactique et d'atteindre un but même s'il y a un problème pendant le déplacement du robot vers son but.

Nous pouvons proposer une extension pour chacun des modules que nous avons définis dans le dernier chapitre :

- En plus de la position du robot par rapport aux deux extrémités du couloir, le module de localisation pourrait calculer la position du robot par rapport aux deux murs constituant le couloir ainsi que son orientation, à l'aide d'un filtre de Kalman ou d'un moindre carré.
- Nous avons vu que les Processus de Décision Markovien Partiellement Observable permettent à un robot mobile de choisir la meilleure action en fonction de la connaissance partielle qu'il a de son environnement et de sa situation et de l'incertitude qu'il a sur celle-ci. D'autre part, les POMDPs appliqués à la robotique mobile constitue actuellement un domaine de recherche très actif. Une extension très intéressante de notre travail serait de modifier notre planificateur de chemins afin de réaliser un véritable POMDP. En effet, nous avons déjà de nombreux éléments qui nous permettraient d'utiliser un POMDP :
  - $\mathcal{S}$: l'ensemble fini d'états de l'environnement *identifiables* par le robot, formé par les états de la carte topologique représentant notre environnement.
  - $\mathcal{A}$: l'ensemble fini d'actions qui se résume à la seule action d'atteindre un but puisque notre navigateur est capable de se placer face à son but avant de se déplacer.
  - $\mathcal{O}$: l'ensemble fini d'observations de l'environnement *identifiables* par le robot, formé par les balises que le robot peut reconnaître.
  - $\mathcal{O}'$: la fonction d'observation qui peut être définie ainsi :

    $$\mathcal{O}' : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{O})$$

    $\mathcal{O}'(s,a,o)$: probabilité d'observer o de l'état s après avoir exécuté l'action a. Nous avons déterminé cette fonction lors de l'interprétation des balises reconnues. En effet connaissant la position du robot (l'état du modèle dans lequel il se trouve), et le déplacement qu'il a réalisé, nous pouvons savoir quelle balise il a "vue" pour y arriver. A partir de là, lorsqu'il va reconnaître une balise nous pouvons facilement déterminer à partir de la matrice de confiance qu'elle est le pourcentage qu'elle corresponde à la balise qu'il aurait dû voir lors de son déplacement.

De plus la procédure d'interprétation des balises reconnues correspond tout à fait à l'exécution d'un POMDP. En effet, à chaque fois qu'une balise est reconnue, nous mettons à jour l'ensemble des états où le robot se trouvait au moment où il a reconnu cette balise. Les états de notre modèle de Markov caché représentant l'environnement et la probabilité associée à chaque état correspondent tout à fait aux beliefs states d'un POMDP. Pour pouvoir utiliser un véritable POMDP, il ne nous manque que la fonction de transition $\mathcal{T}$, et la fonction de gain $\mathcal{R}$. Nous menons actuellement des travaux dans ce sens en collaboration avec Pierre Laroche. L'aboutissement de ces travaux étant la suite logique de notre thèse. L'idée principale est de partir d'une discrétisation classique de l'environnement sous forme de grille d'un mètre sur un mètre pour définir les états de l'environnement, puis progressivement d'agréger les différents états qui correspondent à la même balise comme dans la figure 8.10, et à chaque agrégation de recalculer la fonction de gain. Actuellement, nous ne tenons pas compte de la fonction de transition puisque nous supposons comment nous l'avons fait dans le dernier chapitre de notre mémoire qu'à chaque fois que le robot doit atteindre un but dans son environnement, il l'atteint toujours. C'est la mise à jour des beliefs states au fur et à mesure que les balises sont reconnues et interprétées qui permet de déterminer si les actions se sont bien déroulées.

Publications personnelles

[Aycard *et al.*, 1997a] O. Aycard, F. Charpillet, D. Fohr et J.-F. Mari. Place learning and recognition using hidden markov models. Dans *proceedings of IEEE International Conference on Intelligent Robots and Systems*, pages 1741–1746, 1997.

[Aycard *et al.*, 1997b] O. Aycard, F. Charpillet et J.-P. Haton. A new approach to design fuzzy controllers for mobile robots navigation. Dans *proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 68–73, 1997.

[Aycard *et al.*, 1998] O. Aycard, P. Laroche et F. Charpillet. Mobile robot localization in dynamic environments using places recognition. Dans *proceedings of IEEE International Conference on Robotics and Automation*, pages 3135–3140, 1998.

[Morignot *et al.*, 1997] P. Morignot, O. Aycard et F. Charpillet. A pair of heterogeneous agents in a unique vehicle for object motion. Dans *proceedings of IEEE International Conference on Tools with Artificial Intelligence*, pages 508–513, 1997.

Bibliographie

[Alami *et al.*, 1993] R. Alami, R.Chatila et B. Espiau. Designing an intelligent control architecture for autonomous robots. Dans *International Conference on Advanced Robotics*, pages 435–440, 1993.

[Albus et Lumia, 1987] H.G Albus, J.S McCain et R. Lumia. Nasa/nsb standard refernce model for telerobot control system architecture. Rapport Technique 1235, Us Departement of Commerce. National Bureau of Standards, 1987.

[Arkin, 1987] R.C Arkin. Motor schema based navigation for a mobile robot : an approach to programming by behavior. Dans *proceedings of the 1987 IEEE/ICRA*, pages 264–271, 1987.

[Aycard *et al.*, 1997a] O. Aycard, F. Charpillet, D. Fohr et J.F Mari. Place learning and recognition using hidden markov models. Dans *proceedings of IEEE/IROS*, pages 1741–1746, 1997.

[Aycard *et al.*, 1997b] O. Aycard, F. Charpillet et J-P. Haton. A new approach to design fuzzy controllers for mobile robots navigation. Dans *proceedings of IEEE/CIRA*, pages 68–73, 1997.

[Aycard *et al.*, 1998] O. Aycard, P. Laroche et F. Charpillet. Mobile robot localization in dynamic environments using hidden markov models. Dans *proceedings of IEEE/ICRA*, pages 3135–3140, 1998.

[Baum, 1972] L. E. Baum. An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. *Inequalities*, 3:1 – 8, 1972.

[Bellman, 1957] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[Berenji, 1990] H.R Berenji. *An Introduction to Fuzzy Logic. Applications in Intelligents Systems.* Kluwer Academic Press, 1990.

[Borenstein et Koren, 1989] J Borenstein et Y Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics.*, 19(5):1179–1187, 1989.

[Bouchon-Meunier, 1994] B Bouchon-Meunier. *La Logique Floue.* Collection "Que sais-je?". Presse Universitaire de France, 1994.

[Boulianne *et al.*, 1994] G. Boulianne, P. Kenny, M. Lennig, D.O'Shaughhessy et P. Mermelstein. Books on tape as training data for continuous speech recognition. *Speech Communication*, 14:61 – 70, 1994.

[Brooks, 1986] R.A Brooks. A robust layered control system for a mobile robot. *In IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[Brown, 1987] P.F. Brown. *The Acoustic-Modeling Problem in Automatic Speech Recognition*. PhD thesis, C.S.D., Carnegie Mellon University, 1987.

[Chang et Hsu, 1994] S.W Chang, T.Y Kuo et Y.J Hsu. A two-phase navigation system for mobile robots in dynamic environnements. Dans *IEEE International Conference on Robotics and Systems*, pages 306–313, 1994.

[Chapman, 1987] David Chapman. Planning for Conjunctive Goals. *Artificial Intelligence*, 32:333–377, 1987. Also Tech. Rep. 802, C.S., MIT, 1985.

[Charniak, 1991] E. Charniak. Bayesian networks without tears. *AI Magazine*, Winter 1991.

[Chatila, 1994] R Chatila. Control architectures for autonomous mobile robots. Dans *From Perception to Action Conference*, pages 254–265, 1994.

[Crowley et Sarachik, 1991] P. Crowley, J. Bobet et K. Sarachik. Mobile robot perception using vertical line stereo. *Robotics and Autonomous System*, 7(2-3), 1991.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird et D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[Elfes, 1989] A Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–57, June 1989.

[Fabiani, 1997] L. Fabiani. Analyse de performances pour robots mobiles réactifs. Dans *Septièmes Journées des Jeunes Chercheurs en Robotique*, 1997.

[Feng *et al.*, 1994] L. Feng, J. Borenstein et H.R. Everett. Where am i. Rapport Technique UM-MEAM-94-21, university of Michigan, 1994.

[Fikes et Nilsson, 1971] Richard E. Fikes et Nils J. Nilsson. STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:198–208, 1971.

[Fleury, 1996] S Fleury. *Architecture de contrôle distribuée pour robot mobile autonome : principes, conception et applications*. PhD thesis, Université Paul Sabatier, Toulouse. FRANCE, 1996.

[Forney, 1973] G.D. Forney. The Viterbi Algorithm. *IEEE Transactions*, 61:268–278, March 1973.

[Fraichard, 1992] T Fraichard. *Planification de mouvements pour robot mobile non-holonome en espace de travail dynamique*. PhD thesis, Institut National Polytechnique de Grenoble, Grenoble. FRANCE, 1992. In french.

[Garnier, 1995] P Garnier. *Contrôle d'exécution réactif de mouvements de véhicules en environnement dynamique structuré*. PhD thesis, Institut National Polytechnique de Grenoble, 1995.

[Gaussier, 1994] S. Gaussier, P. Zrehen. A topological neural map for online learning : emergence of obstacle avoidance in a mobile robot. Dans *proceedings of the Third International Conference on Simulation of Adaptative Behavior*, 1994.

[Giralt, 1993] G Giralt. Robots d'intervention : Autonomie, réactivité, programmation.

Rapport Technique 93151, Laboratoire d'Automatique et d'Analyse des Systèmes, Toulouse, France, 1993.

[Hassoun, 1994] M Hassoun. *Contrôle d'exécution des mouvements d'un robot mobile : application à l'assistance à la conduite automobile.* PhD thesis, Institut National Polytechnique de Grenoble, 1994. In french.

[Haton, 1974] J.-P. Haton. *Contribution à l'analyse, la paramétrisation et la reconnaissance automatique de la parole.* PhD thesis, Université de NANCY 1, 1974.

[Hayes-Roth *et al.*, 1993] Barbara Hayes-Roth, Philippe Lalanda, Philippe Morignot, Karl Pfleger et Marko Balabanovic. Plans and Behavior in Intelligent Agents. Rapport technique, Knowledge Systems Laboratory, Stanford, 1993. KSL-93-43.

[Hayes-Roth, 1985] Barbara Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:251–321, 1985.

[Howard, 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachussets, 1960.

[Jelinek, 1976] F. Jelinek. Continuous Speech Recognition by Statistical Methods. *IEEE Trans. on ASSP*, 64(4):532 – 556, April 1976.

[Jenkin et Jasiobedzki, 1993] E Jenkin, M Milios et P Jasiobedzki. Global navigation for ark. Dans *proceedings of the 1993 IEEE International Conference on Intelligent Robots and Systems*, 1993.

[Kambhampati et Nau, 1996] Subbarao Kambhampati et Dana S. Nau. On the Nature and Role of Modal Truth Criteria in Planning. *Artificial Intelligence*, 1996. Also, reduced version in AAAI'94.

[Khatib, 1986] O Khatib. Real-time obstacle avoidance for manipulators and autonomous robots. *International Journal of Robotics Research*, 5(1), 1986.

[Khatib, 1996] M. Khatib. *Sensor-based motion control for mobile robots.* PhD thesis, Laboratoire d'Automatique et d'Analyse des Systèmes, Toulouse. France., 1996.

[Koditschek, 1987] D.E Koditschek. Exact robot navigation by means of potential functions : Some topological considerations. Dans *proceedings of the 1987 International Conference on Robotics and Automation*, pages 1–6, 1987.

[Kortenkamp et Weymouth, 1994] D Kortenkamp et T Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. Dans *Proceedings of AAAI*, 1994.

[Kortenkamp, 1993] D Kortenkamp. *Cognitive maps for mobile robots : a representation for mapping and navigation.* PhD thesis, University of Michigan, 1993.

[Kriegman et Binford, 1989] E. Kriegman, D.J. Triendl et T.O Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Transactions on Robotics and Automation*, 5(6), 1989.

[Krogh et Thorpe, 1986] B.H Krogh et C.E Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. Dans *proceedings of the 1986 International Conference on Robotics and Automation*, pages 1664–1669, 1986.

[Laroche, 1996] Pierre Laroche. Modélisation stochastique pour la planification en robotique : tendances actuelles. Rapport technique, LORIA, 1996.

[Latombe, 1991] J.C Latombe. *Robot motion planning.* 1991.

[Lee, 1990] C.C Lee. Fuzzy logic in control systems : Fuzzy logic controller. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):491–418, March 1990.

[Leonard et Durrant-Whyte, 1992] J.J. Leonard et H.F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation.* Kluwer Academic Publishers, 1992.

[Li, 1994] W Li. Perception-action behavior control of a mobile robot in uncertain environments using fuzzy logic. Dans *proceedings of the 1994 IEEE International Conference on Intelligent Robots and Systems*, pages 439–446, 1994.

[Lin, 1993] L-J Lin. Hierarchical learning of robot skills by reinforcement. Dans *IEEE International Conference on Neural Networks*, pages 181–186, 1993.

[Littman *et al.*, 1995] Michael L. Littman, Anthony R. Cassandra et Leslie P. Kaelbling. Learning Policies for Partially Observable Environments: Scaling up. Dans *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[Lumia et Wavering, 1990] J Lumia, R Fiala et A Wavering. The nasrem robot control system and testbed. *IEEE Journal of Robotic and Automation*, 5(1):20–26, 1990.

[Maes, 1993] P Maes. Behaviour-based artificial intelligence. Dans *proceedings of the second international conference on adaptative behaviour*, rédacteur MIT Press, 1993.

[Mahadevan et Connell, 1991] S. Mahadevan et J. Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. Dans *proceedings of the Eight International Workshop on Machine Learning*, pages 328–332, 1991.

[Mamdani, 1976] E.H Mamdani. Advances in the linguistic synthesis of fuzzy controllers. *International Journal of Man-Machine Studies*, 8(6):669–678, 1976.

[Mari *et al.*, 1997] J.-F. Mari, J.-P. Haton et A. Kriouile. Automatic Word Recognition Based on Second-Order Hidden Markov Models. *IEEE Transactions on Speech and Audio Processing*, 5, Janvier 1997.

[Mari, 1996] J.-F. Mari. *Perception de signaux complexes et interaction Homme-Machine.* PhD thesis, Université Henri Poincaré-Nancy I, 1996. Habilitation à diriger des Recherches.

[Morignot *et al.*, 1997] P. Morignot, O. Aycard et F. Charpillet. A pair of heterogeneous agents in a unique vehicle for object motion. Dans *proceedings of IEEE/ICTAI*, pages 508–513, 1997.

[Morignot, 1994] Philippe Morignot. Embedded Planning. Dans *Proceedings of the Second International Conference on A.I. Planning Systems*, rédacteur Kristian Hammond, pages 128–133, Chicago, IL, 1994.

[Nilsson, 1969] N.J Nilsson. A mobile automation : An application of artificial intelligence techniques. Dans *IJCAI*, 1969.

[Nomadics, 1996] Nomadics. Nomad 200 user´s manual. Rapport technique, 1996.

[Nourbakhsh *et al.*, 1993] I. Nourbakhsh, S. Morse, C. Becker, M. Balabanovic, E. Gat, R.Simmons, S.Goodridge, H.Potlapalli, D.Hinkle, K.Jung et D.Van Vactor. The winnings robots from the 1993 robot competition. *AI Magazine*, 14(4):51–61, 1993.

[Pednault, 1986] Edwin Pednault. *The Synthesis of Plans.* PhD thesis, E.E. Dept., Stanford, Palo Alto, California, 1986.

[Pomerleau, 1992] D.A Pomerleau. *Neural Network Perception for Mobile Robot Gui-*

dance. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1992. CMU-CS-92-115.

[Rabiner, 1989] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech recognition. *IEEE Trans. on ASSP*, 77(2):257 – 286, February 1989.

[Radford et Fang, 1993] K.W Radford, J. Hipel et L. Fang. Strategic and tactical analyses in complex decision situations. Dans *IEEE International Conference on Systems, Man and Cybernetics*, pages 143–146, 1993.

[Reignier, 1994] P Reignier. Fuzzy logic techniques for mobile robot obstacle avoidance. *International Journal on Robotics and Autonomous System*, 12:143–153, 1994.

[Rompais et Pradet, 1995] C Rompais, J.F Barret et G Pradet. Neural classifier for environnement recognition in mobile robotics. Dans *proceedings of the IEEE/ICANN*, pages 463–468, 1995.

[Saffioti *et al.*, 93] A. Saffioti, E.H Ruspini et K. Konolige. Robust control of a mobile robot using fuzzy logic. Dans *proceedings of the First European Congress on Fuzzy and Intelligent Technologies*, 93.

[Simmons et Koenig, 1995] R. Simmons et S. Koenig. Probabilistic robot navigation in partially observable environments. Dans *proceedings of IJCAI*, 1995.

[Song et Tai, 1992] K Song et J Tai. Fuzzy navigation of a mobile robot. Dans *proceedings of the 1992 IEEE International Conference on Intelligent Robots and Systems*, pages 621–627, 1992.

[Sutton et Barto, 1981] R Sutton et G Barto. Toward a modern theory of adaptative networks : expectation and rejection. *Psychological Review*, 88(2):135–170, 1981.

[Sutton, 1988] R.S Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Tesauro, 1992] G Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[Thorpe, 1984] C.E Thorpe. Path relaxation : Path planning for a mobile robot. Dans *proceedings of the 1984 American Association for Artificial Intelligence Conference*, pages 318–321, 1984.

[Tigli, 1996] J.Y Tigli. *Vers une Architecture de contrôle pour Robot Mobile orientée Comportement, SMACH*. PhD thesis, Université de Nice - Sophia Antipolis, 1996.

[Yamauchi, 1995] B Yamauchi. *Exploration and spatial learning in dynamic environnements*. PhD thesis, Case Western Reserve University, 1995.

[Yamauchi, 1996a] B Yamauchi. Mobile robot localization in dynamic environments using dead reckoning and evidence grids. Dans *proceedings of the IEEE/ICRA*, pages 674–680, 1996.

[Yamauchi, 1996b] P Yamauchi, B Langley. Place learning in dynamic real-world environments. Dans *Proceedings of ROBOLEARN-96: International Workshop for Learning in Autonomous Robots*, pages 123–129, 1996.

[Yamauchi, 1997] P Yamauchi, B Langley. Place recognition in dynamic environments. *Journal of Robotic Systems*, 14(2):107–120, February 1997. Special Issue on Mobile Robots.

[Zadeh, 1965] L.A Zadeh. Fuzzy sets. *Information and Control*, 8:338–365, 1965.

[Zhu et Latombe, 1989] D.J. Zhu et J.C Latombe. Constraint reformulation and graph searching techniques in hierarchical path planning. Rapport Technique CS-89-1279, Departement of Computer Science, Stanford University, 1989.

[Zimmer, 1995a] U.R Zimmer. *Adaptatives Approaches to basic mobile robot tasks*. PhD thesis, University of Kaiserlautern, Kaiserlautern.Allemagne, 1995.

[Zimmer, 1995b] U.R. Zimmer. Self-localization in dynamic environnement. Dans *IEEE/SOFT International Workshop BIES '95*, 1995.