Contrôle d'exécution réactif de mouvements de véhicules en environnement dynamique structuré

Philippe Garnier

THÈSE

présentée par

Philippe GARNIER

pour obtenir le grade de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 mars 1992)

(Spécialité : Informatique)

==========

CONTRÔLE D'EXÉCUTION RÉACTIF DE MOUVEMENTS DE VÉHICULES EN ENVIRONNEMENT DYNAMIQUE STRUCTURÉ

==========

Date de soutenance : 21 décembre 1995

Composition du jury :

Président :	M. Augustin Lux
Rapporteurs :	M. François Pin
	M. René Zapata
Examinateurs :	M. Christian Laugier
	M. Michel Parent
	M. Thierry Fraichard

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale et
d'Intelligence Artificielle
46, Avenue Félix Viallet
38031 Grenoble Cedex 1 – France

A ma famille,
avec une pensée particulière pour
mon Grand-père Daurian Nal (1904-1945)

Remerciements

Au terme de ma thèse, je tiens à remercier ici tout d'abord les personnes qui m'ont fait l'honneur d'accepter de faire partie de mon jury :

- Mr Augustin Lux, Professeur à l'Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG) et responsable de la Formation Doctorale en Informatique à l'Institut National Polytechnique de Grenoble (INPG), qui a bien voulu présider mon jury. Cette responsabilité, spontanément acceptée, démontre son attachement à des travaux effectués par une équipe œuvrant dans le même laboratoire que la sienne;

- Mr René Zapata, Maître de Conférences à l'Université de Montpellier, et Mr François Pin, Directeur de Recherche à l'Oak Ridge National Institute (USA), qui se sont acquittés de la lourde tâche de rapporteurs. J'ai pu apprécier la pertinence de leurs remarques, mais aussi leur soutien et leur disponibilité, pour préparer ma soutenance dans de bonnes conditions;

- Mr Michel Parent, Directeur du Programme Praxitèle à l'INRIA, qui, par sa présence, a manifesté l'intérêt qu'il porte à mes travaux effectués au sein de ce programme;

- Mr Thierry Fraichard, Chargé de Recherche INRIA, pour avoir encadré mon travail dans ma dernière année de thèse. J'ai été sensible à la conscience professionnelle de Thierry dans une activité relativement nouvelle pour lui;

- Mr Christian Laugier, Directeur de Recherche INRIA et responsable du projet SHARP, pour m'avoir accepté au sein de son équipe, me permettant de la sorte de concrétiser mon rêve de travailler dans le cadre de la robotique mobile.

Un travail de thèse est indissociable d'un laboratoire et d'une équipe de recherche. De la qualité des rapports humains qui y règnent, dépend le bon déroulement de celui-ci. Selon la formule consacrée, il est impossible de citer toutes les personnes qui, à différents niveaux, ont contribué à établir une atmosphère à la fois studieuse et amicale. Je les prie de bien vouloir m'en excuser. Un merci tout particulier pour leur aide ou simplement leur amitié à Juan Manuel Ahuactzin, Gérard Baille, Moëz Chérif, Anton Deguet, Fernando De La Rosa, Leszek Lisowski, Isabelle Mazon, Didier Pallard, Cyril Novales et Alexis Scheuer du projet SHARP, Patrice de Marconnay et Ambarish Goswami des équipe PRIMA et projet BIP. Il me serait impossible d'oublier James Crowley, Professeur ENSIMAG

et responsable de l'équipe PRIMA pour son dévouement quant à la recherche, malheureusement vaine, d'une date lui permettant de faire partie de mon jury, Bernard Espiau, Directeur de Recherche INRIA et responsable du projet BIP, pour m'avoir fait l'honneur d'assister à ma soutenance, et Mme Vacher, du Bureau des Thèses de l'INPG, pour sa gentillesse et sa compétence lors de mes démarches administratives.

Un travail de thèse dépend également de considérations financières. Je tiens, pour cela, à saluer le Ministère de la Recherche pour m'avoir attribué une bourse et l'INRIA, à travers le programme Praxitèle, pour son soutien lors de ma dernière année de thèse.

Enfin, je me souviens, dès le début de mon cursus universitaire, avoir considéré avec un intérêt certain, ce long chemin estudiantin conduisant au doctorat. Ce parcours aura été, pour moi, jalonné de joies certes, mais aussi parfois de périodes moins riantes. C'est en pensant à ces dernières que je tiens à exprimer ici ma profonde gratitude à mes parents pour leur soutien sans faille et de chaque instant.

Philippe Garnier,
21 décembre 1995

Table des matières

Introduction

Les Trois Lois de la Robotique :
1. Un robot ne peut nuire à un être humain ni permettre qu'un être humain soit lésé du fait de sa passivité.
2. Un robot doit obéir aux ordres qui lui sont donnés par des êtres humains sauf au cas où ces ordres seraient incompatibles avec la Première Loi.
3. Un robot doit protéger sa propre existence aussi longtemps que cela n'est pas incompatible avec la Première ou la Seconde Loi.

Isaac Asimov

Depuis une vingtaine d'années, un effort particulier a été fait dans les domaines de la recherche et de l'industrie pour construire des robots mobiles évoluant avec un minimum d'intervention humaine. Une première génération de robots a consisté en des machines capables d'évoluer dans des environnements parfaitement connus : celles-ci réalisent des missions planifiées à partir d'une modélisation complète de l'environnement (laboratoires) ou se contentent de suivre une trajectoire par un mécanisme de filo-guidage (robots de manutention). Le point commun de ces robots est qu'ils évoluent dans un environnement qui leur est totalement dédié.

Cependant, lorsque l'environnement devient plus complexe (i.e. partiellement connu, dynamique, ...), il apparaît indispensable que le robot soit doté de capacités décisionnelles aptes à le faire réagir aux aléas qui peuvent contrarier ses mouvements (pannes partielles, obstacles imprévus). Cela peut être le cas lorsque le robot mobile évolue dans des environnements hostiles à l'homme (milieu radioactif) ou trop éloignés (exploration spatiale). Pour cela, le robot doit suivre le schéma correspondant au paradigme *Percevoir-Décider-Agir* au sein d'une architecture de contrôle. Bien que, comme nous le verrons par la suite, plusieurs architectures existent au niveau de la satisfaction de ce paradigme, l'activité d'un tel robot se ramène aux tâches énoncées ci-après.

- **Percevoir** : Le robot doit acquérir des informations sur l'environnement dans lequel il évolue par l'intermédiaire de capteurs. Ces informations permettent de mettre à jour un modèle de l'environnement (architectures hié-

rarchiques) ou peuvent être directement utilisées comme entrées de comportements de bas niveau (architectures purement réactives);

- **Décider** : Le robot doit définir des séquences d'actions résultant d'un raisonnement appliqué sur un modèle de l'environnement (architectures hiérarchiques) ou répondant de manière réflexe à des stimuli étroitement liés aux capteurs (architectures purement réactives);

- **Agir** : Le robot doit enfin exécuter les séquences d'actions en envoyant des consignes aux actionneurs par l'intermédiaire des asservissements..

Au sein de la partie *décision*, comme nous l'avons dit plus haut, il est nécessaire de contrôler l'exécution de ces actions afin que le robot s'adapte rapidement à des événements imprévus.

Contrôle d'exécution de mouvements

Dans l'architecture de contrôle sur laquelle nous avons travaillé (figure .1), nous avons mis en évidence les trois fonctions essentielles évoquées précédemment. Chacune de ces trois fonctions sera détaillée par la suite. Comme nous pouvons

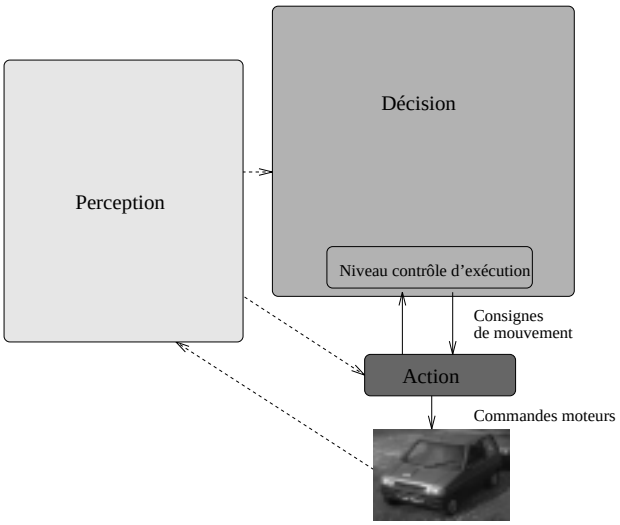


FIG. .1 –: *Notre architecture de contrôle*

le voir sur la figure, au niveau de la partie *décision*, le **contrôle d'exécution** assure l'interfaçage avec la partie *action*. Le rôle du contrôleur est de traduire les séquences d'actions générées par les plus hauts niveaux de la partie *décision* (génération de mission, planification) en consignes de mouvements transmises à la partie *action* (asservissements). En plus de cette tâche, le contrôle d'exécution doit assurer la bonne réalisation des actions en prenant en compte de manière

réactive l'évolution de l'environnement. Le rôle du contrôle d'exécution consiste donc à :

1. envoyer périodiquement les consignes permettant d'exécuter des actions nominales définies par la partie *décision*;
2. contrôler la bonne réalisation des actions nominales;
3. amender, si cela est possible, les actions nominales et proposer de manière réactive des corrections locales;
4. faire appel aux modules de haut niveau de la *décision* en cas d'échec pour proposer de nouvelles actions nominales.

Les corrections locales ont pour but de minimiser les temps de réaction du véhicule devant un événement imprévu. On met ainsi en œuvre une boucle de contrôle plus rapide que celles faisant intervenir les hauts niveaux de la partie *décision*. Cela peut être rendu possible par une action déduite directement des informations délivrées par les capteurs. On s'affranchit ainsi d'un traitement des informations coûteux en temps de calcul, comme cela est nécessaire pour les boucles de contrôle de plus haut niveau. On dispose alors d'une boucle de contrôle permettant des actions rapides, sur des horizons temporels relativement réduits. Ceci vient en complément des boucles de contrôle mettant en œuvre une modélisation relativement fine de l'environnement pour un raisonnement plus poussé sur le comportement du véhicule à moyen terme.

Notre contribution

Les travaux présentés dans cette thèse portent essentiellement sur le développement d'un contrôleur d'exécution de mouvements pour un robot mobile de type *voiture* dans un environnement dynamique partiellement connu. Notre contrôleur d'exécution de mouvements est un système réactif[1] considérant en entrée une trajectoire de référence[2] délivrée par un module de planification et satisfaisant une mission donnée[3].

– Notre contrôleur réactif doit sa propriété à la correspondance étroite entre les capteurs et les actionneurs, i.e. l'utilisation directe d'informations capteurs dans la génération de consignes. Il est constitué d'un ensemble de comportements de base (suivi de trajectoires, évitement d'obstacles, ...) activés

1. "Système apte à réagir continûment à un environnement physique, à une vitesse déterminée par cet environnement" [31, 5].

2. Trajectoire sous la forme d'une séquence de configurations datées, enrichies de la vitesse désirée du véhicule, sous la forme $(x_{ref}(t), y_{ref}(t), \theta_{ref}(t), v_{ref}(t))$.

3. Sous sa forme la plus simple, mission du type "aller de la configuration q_{init} à la configuration q_{but}".

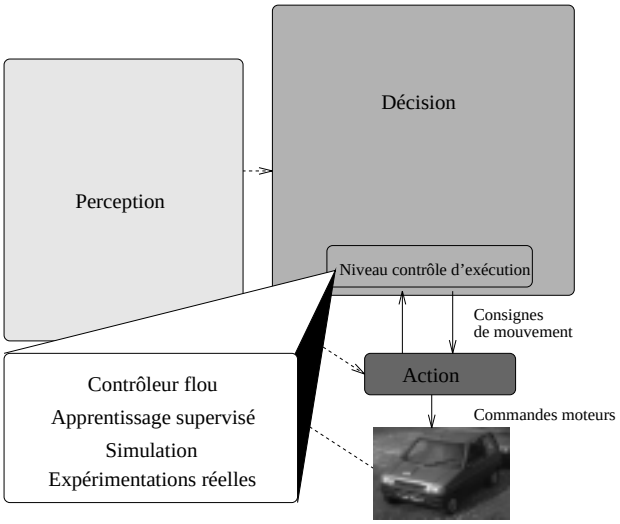


FIG. .2 –: *Notre contribution au niveau du contrôle d'exécution de mouvements*

en parallèle. Ces comportements sont ensuite combinés dans le but d'obtenir un comportement global correspondant à l'exécution des mouvements planifiés. La programmation de notre contrôleur repose sur la logique floue, au travers de l'utilisation d'un contrôleur flou de type *Mamdani* [48, 51]. L'intérêt essentiel de cette approche est de coder les comportements désirés sous la forme de règles pondérées[4] exprimées dans un langage proche du langage humain (intégrant de ce fait naturellement les notions d'incertitude, d'imprécision, d'imperfection); pour cette raison, le contrôleur peut être qualifié de "système expert flou".

– Bien qu'un contrôleur flou se caractérise essentiellement par sa relative facilité de conception pour résoudre un problème précis (pendule inversé par exemple), il apparaît rapidement qu'un "réglage" de celui-ci soit nécessaire lorsque l'on désire intégrer plusieurs critères. Cet aspect multi-critères occasionne en effet un accroissement important du nombre de règles de la base pouvant être activées simultanément et il s'agit de déterminer l'importance relative de chacune. Afin de nous affranchir d'un réglage empirique fastidieux, nous avons opté pour une méthode d'apprentissage supervisé[5] basée sur l'utilisation de l'algorithme du simplexe issu de l'optimisation combinatoire. Cet apprentissage porte sur la détermination automatique des poids de règles. Nous montrerons dans ce manuscrit qu'un problème d'apprentissage supervisé peut être facilement codé sous la forme d'un programme linéaire dont la méthode de résolution la plus efficace est l'algorithme du

4. Un poids associé à chaque règle nous permet de caractériser l'importance relative de cette règle par rapport au reste de la base.

5. Apprentissage effectué à partir d'un ensemble d'échantillons correspondant à des instanciations d'entrées-sorties désirées du contrôleur.

simplexe.

- Afin de valider notre contrôleur flou, nous avons été amené à construire, dans un premier temps, une plate-forme de simulation d'un environnement $2D$ pour un véhicule de type *voiture*. Celle-ci constitue, de par son Interface Homme-Machine, sa modularité, sa bibliothèque d'acquisition d'informations perceptives et la visualisation $3D$ de l'évolution de l'environnement, un outil de validation important [6].

- Les résultats obtenus en simulation nous ont conduit naturellement à une implémentation du contrôleur sur un véhicule réel dans le cadre du projet INRIA/INRETS Praxitèle qui constitue le cadre d'application de notre contrôleur d'exécution de mouvements.

Cadre d'application : le projet Praxitèle

Depuis quelques années, bon nombre de projets concernant les transports routiers ont vu le jour. Ces projets consistent à faire disparaître ou au moins à limiter les désagréments liés à une utilisation sans cesse croissante des voitures. Deux axes principaux se sont dégagés ces dernières années : une volonté d'accroissement de la sécurité au niveau des voitures particulières et l'introduction de transports de substitution. En ce qui concerne le premier axe, nous faisons référence au projet européen Eureka PROMETHEUS [7], en particulier au sous-programme PRO-ART et à son démonstrateur français Prolab. L'objectif consistait en l'élaboration d'un copilote électronique pour l'aide à la conduite [33, 35, 34, 32]. Pour le deuxième axe, nous pouvons citer le projet TULIP [8] du constructeur automobile PSA et plus particulièrement le projet INRIA/INRETS Praxitèle [9]. Ces concepts cherchent à mettre à la disposition des usagers un moyen de locomotion, en l'occurrence des petits véhicules électriques, intermédiaire entre les transports en commun et la voiture particulière. Ces véhicules peuvent être pilotés manuellement (utilisation classique par des usagers) ou de manière automatique (retour à vide dit "haut-le-pied", gestion des véhicules à l'intérieur de parkings entièrement automatisés). Chaque véhicule est donc un robot mobile doté d'une architecture de contrôle, à l'intérieur de laquelle nous avons travaillé au niveau du contrôle d'exécution.

6. Les véhicules évoluent sur un plan mais leur modélisation, ainsi que la visualisation de la scène, se font en $3D$.

7. Acronyme de PROgraMme for a European Traffic with Highest Efficiency and Unprecedented Safety.

8. Acronyme de Transport Urbain Libre Individuel et Public.

9. Action de recherche de l'INRIA sur le concept de TUPI (Transport Urbain Public Individuel) en partenariat avec l'INRETS, la CGEA, Renault et EDF.

Plan du manuscrit

Nous décrivons dans le chapitre I les trois grandes familles d'architectures de contrôle d'un robot mobile autonome à savoir l'approche classique, l'approche purement réactive et enfin l'approche hybride résultant de la prise en compte des deux précédentes. Nous introduisons par la suite l'architecture de contrôle hybride sur laquelle nous avons travaillé. Nous présentons enfin différents contrôleurs d'exécution de mouvements avant d'introduire succinctement notre contrôleur basé sur l'utilisation de la logique floue.

Le concept de logique floue est détaillé dans le chapitre II. En plus de la théorie de base définie par Zadeh, une attention particulière est portée à la notion de contrôle flou à travers la description du contrôleur flou générique de Mamdani que nous avons utilisé dans notre travail.

Notre contrôleur flou d'exécution de mouvements est présenté dans le chapitre III. Celui-ci est basé sur l'utilisation d'une base unifiée de règles pondérées codant un certain nombre de comportements (suivi de trajectoire, prise en compte de l'environnement statique et dynamique, ...). Chaque poids associé à une règle permet de définir l'importance de cette règle par rapport au reste de la base.

Nous présentons ensuite notre méthode d'apprentissage paramétrique supervisé dans le chapitre IV, celui-ci nous permettant de régler automatiquement notre contrôleur flou. L'apprentissage porte sur la détermination des poids associés à chaque règle.

Le chapitre V décrit l'implantation de notre contrôleur d'exécution de mouvements au sein de l'architecture de simulation que nous avons développée, ainsi que sur notre véhicule prototype dans le cadre du projet Praxitèle. Sont décrits notre plate-forme de simulation, le projet Praxitèle avec le concept de TUPI, les caractéristiques du véhicule expérimental et l'outil de spécification de la tâche robotique associée à notre architecture : Orccad. Des résultats issus de la phase de simulation sont également présentés.

Les résultats obtenus au niveau du contrôle d'exécution de mouvements nous ont amené à utiliser la logique floue dans le cadre d'une application précise : la génération de missions au sein d'un parking automatisé, également dans le cadre du projet Praxitèle. Cette contribution est décrite en annexe.

Chapitre I

Architectures de contrôle de robots

Nous décrivons dans ce chapitre les trois grandes familles d'architectures de contrôle d'un robot mobile autonome, à savoir l'approche hiérarchique, l'approche purement réactive et enfin l'approche hybride résultant de la prise en compte des deux précédentes. Nous introduisons par la suite l'architecture de contrôle hybride sur laquelle nous avons travaillé. Nous présentons enfin différents contrôleurs d'exécution de mouvements avant d'introduire succinctement notre contrôleur basée sur l'utilisation de la logique floue.

1. Introduction

Dans la littérature, nous constatons que l'on peut regrouper les architectures de contrôle de robots mobiles en trois catégories. La plus ancienne concerne certainement les architectures hiérarchiques résultant d'une approche "Top-Down" pour un robot autonome. Dans une telle architecture, on cherche avant tout à modéliser la connaissance et le raisonnement sur celle-ci. Les actions du robot ne peuvent être que le résultat de ce raisonnement. Basées sur des considérations éthologiques, les architectures purement réactives s'appuient sur une approche radicalement opposée, appelée "Bottom-Up". L'idée est que les comportements de haut niveau d'un robot résultent de l'activation de sous-comportements de base réactifs, ces derniers pouvant être vus comme des réponses réflexes à des stimuli directement issus des capteurs du robot. Enfin, les architectures hybrides tendent tout naturellement à tirer parti au mieux des avantages des deux approches précédentes tout en minimisant leurs lacunes respectives.

2. Systèmes hiérarchiques

Afin de rendre une machine intelligente, beaucoup ont essayé de copier l'homme. Pour cela, on s'est appuyé sur un schéma d'intelligence artificielle conçu dans les années cinquante par Newell et Simon. L'architecture de contrôle hiérarchique d'un robot autonome repose sur la décomposition fonctionnelle apparaissant dans la figure I.1

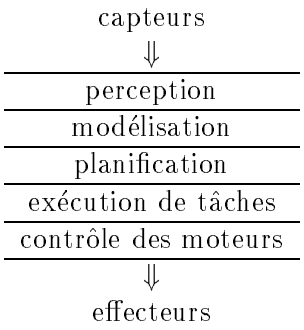


FIG. I.1 –: *L'architecture hiérarchique d'un robot autonome*

Un robot doit commencer par traiter toutes les données recueillies par ses capteurs et concernant son environnement. Ce n'est qu'une fois ce traitement fait qu'il peut identifier les objets qui sont dans son environnement proche. Il lui faut ensuite bâtir une sorte de structure interne des données analysées, pour se représenter la scène dans son ensemble, puis utiliser cette structure pour faire une planification. Après quoi, il doit calculer au mieux une séquence de commandes vers les effecteurs pour exécuter le plan prévu.

Dans une telle architecture, on essaie de construire un modèle de l'environnement le plus complet possible et ensuite de raisonner sur la(les) représentation(s) de celui-ci. On privilégie l'aspect cognitif dans le but de reproduire au mieux l'intelligence humaine.

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 (cartes 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 :

- ces systèmes, même implantés sur de super-calculateurs, sont relativement lents pour un observateur extérieur (notamment pour maintenir le modèle

du monde). De plus, 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 systèmes sont peu adaptatifs dans la mesure où il leur est difficile de réagir à des situations légèrement différentes de celles qu'ils connaissent. Cela exprime une lacune dans la possibilité de généraliser des situations possibles.

- ces systèmes sont en général peu robustes de par leur modèle centralisé. Une défaillance d'un module peut provoquer le blocage de toute l'architecture.

L'intelligence artificielle traditionnelle s'est appuyée sur la volonté de reproduire des systèmes vivants, à travers des concepts plus ou moins valides issus d'observations macroscopiques empiriques, ne correspondant pas aux mécanismes réels régissant les êtres vivants. Certains travaux, par exemple, inspirés des théories de Tinbergen en éthologie, ont prôné une intelligence hiérarchique chez les êtres vivants, appelée *Top-Down* : l'intelligence est associée à un raisonnement qui conditionne les actions du robot.

Les limitations de cette approche ont conduit certains chercheurs à développer des systèmes purement réactifs.

3. Systèmes purement réactifs

Nous pouvons définir la réactivité comme *l'aptitude à réagir continûment à un environnement physique, à une vitesse déterminée par cet environnement* [31, 5]. Les systèmes réactifs s'opposent en cela aux systèmes transformationnels [1] et aux systèmes interactifs [2].

3.1. Idée de base

Des études récentes dans le domaine de l'éthologie, en particulier dans le domaine de l'entomologie, ont été à l'origine d'une approche complètement différente de la précédente. Selon Brooks, il ne s'agit plus de considérer l'intelligence selon un schéma *Top-Down* mais *Bottom-Up*. Les insectes, bien que disposant de par leur taille, de possibilités de modélisation de l'environnement et de raisonnement limitées, n'en sont pas moins aptes à avoir des comportements cohérents et relativement complexes. Cette constatation induit l'idée suivante : pourquoi

1. Systèmes fournissant des sorties uniquement à leur terminaison.
2. Systèmes réagissant continûment à leur environnement, mais à une vitesse qui leur est propre.

ne pas abandonner le concept centré sur la modélisation de l'environnement et le raisonnement, au profit d'une architecture totalement distribuée privilégiant un lien étroit entre perception et action. Arkin parle de modèle "action-oriented perception" précisant que les besoins perceptifs d'un agent doivent être basés sur ses nécessités en action. Cela s'oppose à une approche hiérarchique collectant des informations, puis travaillant à différents niveaux d'abstraction, sans vraiment prendre en considération les utilisations ultérieures qui en seront faites (on privilégie le modèle et sa mise à jour). Brooks va même plus loin qu'Arkin en affirmant qu'il n'y a de meilleur modèle de l'environnement que l'environnement lui-même.

3.2. L'approche comportementale

L'architecture "subsumption" de Brooks [8, 9] peut être définie comme une hiérarchie de comportements plutôt que comme une hiérarchie fondée sur une abstraction des données. Le concept de base est le suivant : un comportement donné, si complexe soit-il, peut être décomposé en comportements élémentaires représentant chacun un niveau de compétence. Ces différents niveaux sont placés en couches (hiérarchie de couches) et chaque compétence couple directement perception et action (figure I.2). On parle alors de théorie *comportementaliste*, par opposition à *cognitiviste*.

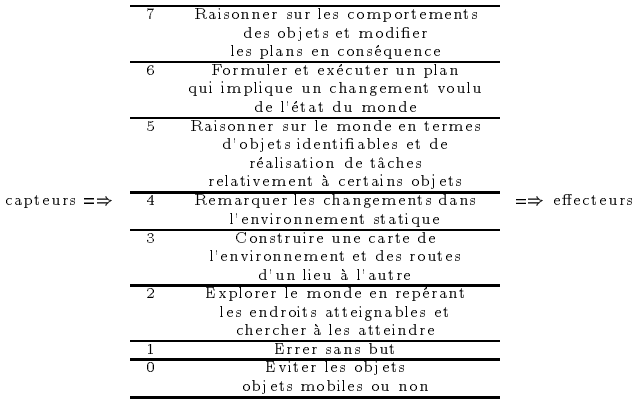


FIG. I.2 –: *L'architecture en couche de Brooks ("subsumption architecture") pour un robot d'exploration*

Ainsi, contrairement aux systèmes hiérarchiques, le comportement global du robot apparaît comme la résultante de plusieurs comportements actifs simultanément. La base de l'architecture est un réseau câblé de machines d'états finis augmentés d'éléments prenant en considération le temps (Augmented Finite State Machine) et communiquant par l'intermédiaire de messages. Une telle machine est représentée dans la figure I.3.

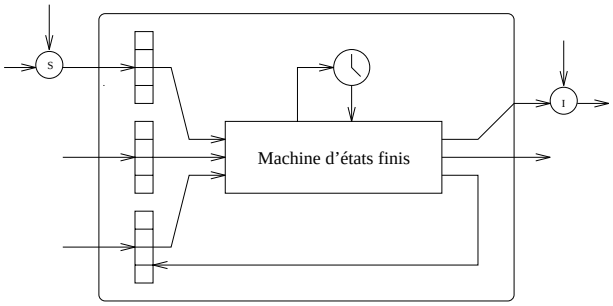


FIG. I.3 –: *Machine d'états finis augmentée (AFSM)*

Une AFSM est donc composée de registres, d'horloges, d'un circuit combinatoire et d'une machine d'états finis. Chaque registre peut être chargé en le reliant au câble d'entrée et en envoyant des messages à partir d'autres machines. Les messages arrivent ainsi dans des registres en écrasant la valeur qui s'y trouve.

L'arbitrage entre les comportements s'appuie sur la hiérarchie des couches : la couche n contrôle seulement la couche $n-1$ à travers des mécanismes d'inhibition et de suppression (figure I.4). La couche n "subsume" la couche $n-1$.

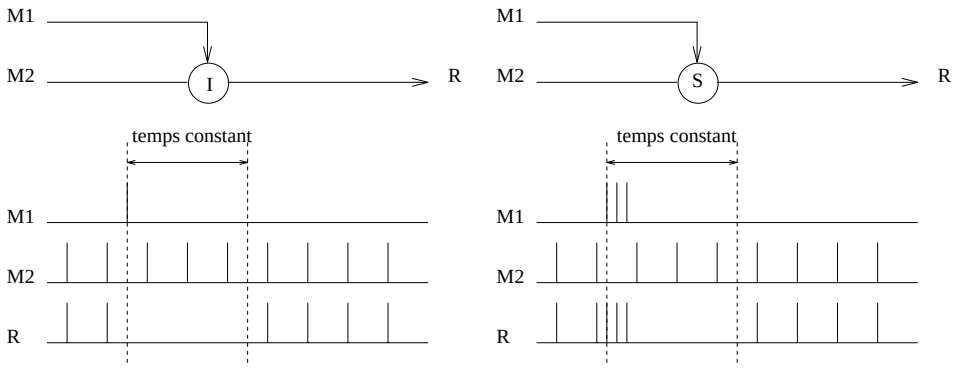


FIG. I.4 –: *Schémas de câblage et chronogrammes des mécanismes (a) d'inhibition, (b) de suppression [11]*

Dans la figure I.4(a), la sortie de M_1 inhibe la sortie de M_2 alors que, dans la figure I.4(b), elle la remplace. Dans les deux cas, l'opération ne dure qu'un temps constant.

L'idée de Brooks est de construire un robot de manière incrémentale : chaque nouvelle compétence est construite comme la couche supérieure (de niveau n) de l'architecture existante avec des liens possibles vers les couches inférieures : entrées issues des niveaux k ($k < n$) et sorties vers le niveau $n-1$. Les couches inférieures n'ont pas à être modifiées.

Nous pouvons illustrer l'interaction entre différents comportements par la des-

cription du fonctionnement de la "main" du robot Herbert du MIT [11]. Il s'agit bien évidemment d'une partie très réduite de l'architecture totale; la main elle-même a été simplifiée (figure I.5).

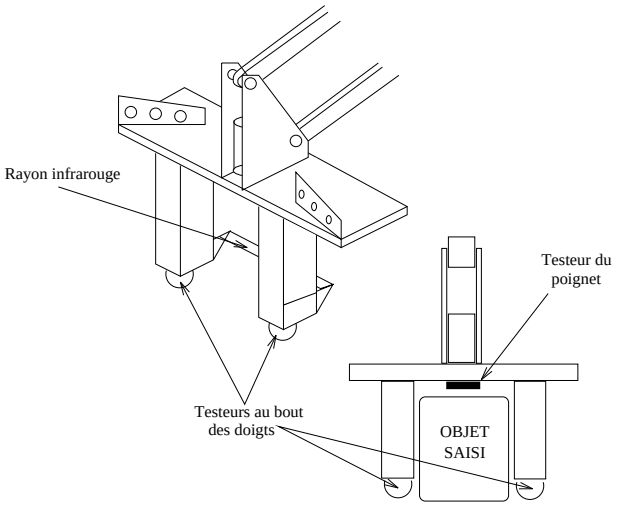


FIG. I.5 –: *Main simplifiée du robot Herbert (MIT) en perspective et en coupe [11]*

Près de l'extrémité des doigts se trouve un rayon infrarouge qui est interrompu lorsqu'un objet se trouve entre les doigts. Les extrémités de ces derniers sont munies de capteurs indiquant si la pince rencontre un objet.

La commande de la main est schématisée en I.6 : le comportement de base *ouvrir* tend à maintenir la main ouverte. Ce comportement est modifié par le comportement *saisir* lorsque le rayon est interrompu. Dans ce cas, la main sera fermée aussi longtemps que le rayon sera ainsi coupé. Mais le comportement *saisir* peut à son tour être modifié par le comportement *déposer* qui force la main à s'ouvrir. C'est de cette manière que la main repose les objets saisis. Elle descend jusqu'à ce que l'objet saisi et le plan de dépose exercent une force activant le testeur du poignet.

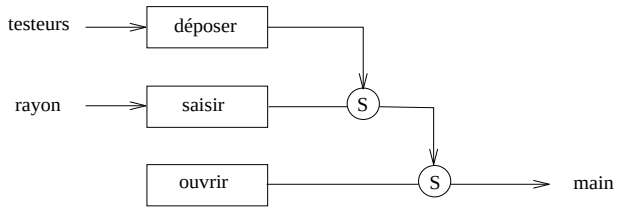


FIG. I.6 –: *Système de contrôle associé à la tâche de saisie de la main du robot Herbert (MIT) [11]*

La figure I.7 représente quelques véhicules utilisant l'architecture "subsumption" de Brooks.

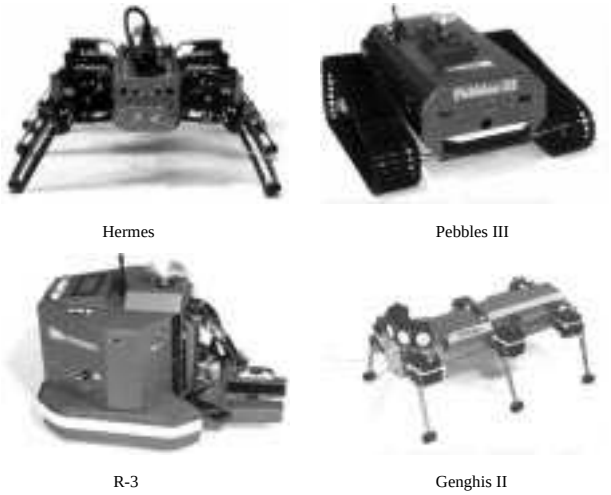


FIG. I.7 –: *Robots utilisant la "Subsumption Architecture" de Brooks (photos IRS)*

Points forts :

- la robustesse
  du fait de la relative indépendance de chacun des comportements les uns par rapport aux autres, si un venait à avoir une défaillance, le robot serait toujours capable de réaliser une tâche, ce qui est très important lorsque l'on a affaire à un robot autonome.

- la rapidité de réponse
  ceci est dû au 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 précédents. Cet aspect est primordial pour la mise au point et la recherche des pannes notamment. Cette approche permet également d'accroître la complexité du comportement global du robot sans pour autant accroître celle du contrôle global puisque celui-ci est complètement distribué.

- faible coût
  un robot avec un telle architecture est d'un coût moindre à celui d'un un robot doté d'une architecture traditionnelle réalisant la même tâche.

- la possibilité de miniaturisation
  d'après Brooks, la relative simplicité de l'architecture de contrôle de ses robots autorise une réduction drastique de ces derniers. Ceci pourrait ouvrir

la porte à de vastes domaines d'applications dont certains peuvent apparaître encore utopiques. On peut notamment citer la microchirurgie [16] et la construction d'une base lunaire où Brooks pense utiliser une multitude de petits robots dont les avantages exposés précédemment permettraient de s'affranchir de l'utilisation de robots plus conventionnels [10].

Points faibles : Il semble cependant qu'une telle approche, bien que modulaire, présente quelques inconvénients pratiques lors de la mise en place d'un nouveau module : pour une application complexe, on peut imaginer les problèmes de communications (câblage, nature des échanges, cohérence avec les communications déjà existantes, ...).

Une critique souvent formulée à l'encontre de la *subsumption architecture* porte sur le fait qu'aucun robot de Brooks n'ait été testé en environnement extérieur. L'argument principal des détracteurs de cette approche réside dans la complète remise en question de l'architecture existante. Cela concerne les capteurs utilisés mais aussi les comportements, même de bas niveau : quelle est en effet la pertinence d'un *suivi de mur* dans un environnement quelconque ?

Enfin, la *subsumption architecture* souffre d'une absence de capacité de raisonnement. Même si ces considérations ont été prises en compte au niveau théorique, Brooks n'a pas dépassé, à notre connaissance, le troisième niveau de son architecture, si l'on se réfère aux expérimentations qui ont été réalisées.

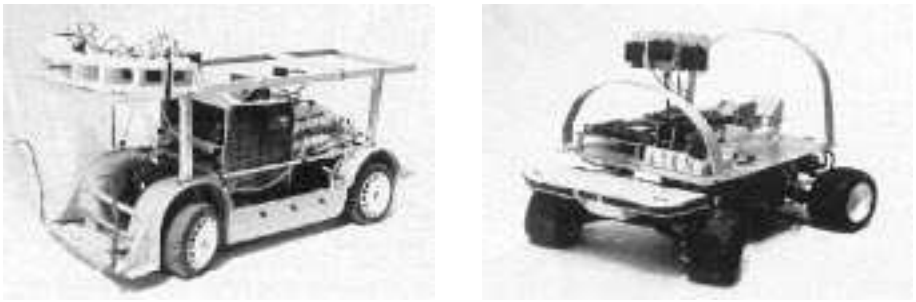
3.3. Les Zones Virtuelles Déformables (Z.V.D.)

Zapata [74] propose une architecture constituée de quatre modules correspondant chacun à un comportement au sein d'une architecture hiérarchique. Ces modules sont, en allant du niveau le plus bas au plus haut : l'arrêt d'urgence, l'évitement d'obstacles, l'exécution des déplacements définis par la mission et le suivi de cible. Les niveaux les plus bas (arrêt d'urgence et évitement d'obstacles) sont basés sur l'utilisation de la notion de Zones Virtuelles Déformables (Z.V.D.). Une Z.V.D. est une enveloppe entourant le robot dont la forme est déterminée par l'état cinématique du robot et les informations perceptives délivrées par les capteurs de celui-ci. On distingue deux sortes de déformations :

- **les déformations contrôlées**, dues aux commandes;
- **les déformations non contrôlées**, dues aux obstacles de l'environnement détectés par les capteurs.

Le principe de fonctionnement est le suivant : il s'agit de générer les commandes aptes à respecter le mieux possible la forme d'une Z.V.D. de manière à ce qu'elle ne dépende que de l'état du véhicule. Ainsi, lorsqu'aucun obstacle n'est détecté, le robot est commandé par les modules de haut niveau. Par contre, lorsqu'un obstacle est détecté, celui-ci provoque une déformation locale de la Z.V.D. associée.

Le module d'évitement d'obstacles inhibe les modules de haut niveau et essaie de minimiser la déformation de la Z.V.D.. Cela peut être réalisé en changeant de cap et/ou réduisant la vitesse de translation du robot. Si, malgré cela, un obstacle pénètre dans la zone d'arrêt d'urgence, contenue dans la zone d'évitement d'obstacle, le robot s'immobilise. Cette approche a été testée sur plusieurs robots mobiles rapides (pouvant évoluer à une vitesse de 7 m/s). On peut citer en particulier les robots SNAKE II et RAT [55] apparaissant dans la figure I.8.



SNAKE II RAT

FIG. I.8 –: *Robots utilisant la notion de Zones Virtuelles Déformables de Zapata (photos [55])*

Points forts : Cette approche semble assez intuitive et permet d'intégrer de manière unifiée l'état du véhicule et de l'environnement local au sein de la seule notion de Z.V.D.. Les informations issues des capteurs sont directement exploitables, ce qui permet des temps de réactions minimaux. De plus, la notion de déformation de Z.V.D. semble permettre une caractérisation plus formelle que l'approche précédente quant à la génération des consignes.

Points faibles : Il semblerait qu'une telle architecture souffre d'une absence de capacité de raisonnement (tout comme la Subsumption Architecture), l'effort ayant porté essentiellement sur l'aspect réactif.

4. Systèmes 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.

4.1. L'architecture de Payton

L'architecture de Payton[60], tout comme la *Subsumption Architecture* est une architecture hiérarchisée de niveaux exécutés en parallèle. Cependant, contrairement à la précédente, il ne s'agit pas d'une hiérarchie de compétences mais d'une hiérarchie de modules effectuant des tâches de planification ordonnancées selon leur aptitude réactive et leurs besoins en traitement d'informations. Cette architecture (figure I.9) est constituée de quatre modules :

- *la planification de missions* transforme une mission en un ensemble de buts géographiques et de contraintes de mouvements.
- *la planification basée sur des cartes* détermine des routes pour rallier les différents objectifs.
- *la planification locale* exécute le parcours des routes.
- *la planification réflexe* effectue le contrôle effectif du robot en temps réel.

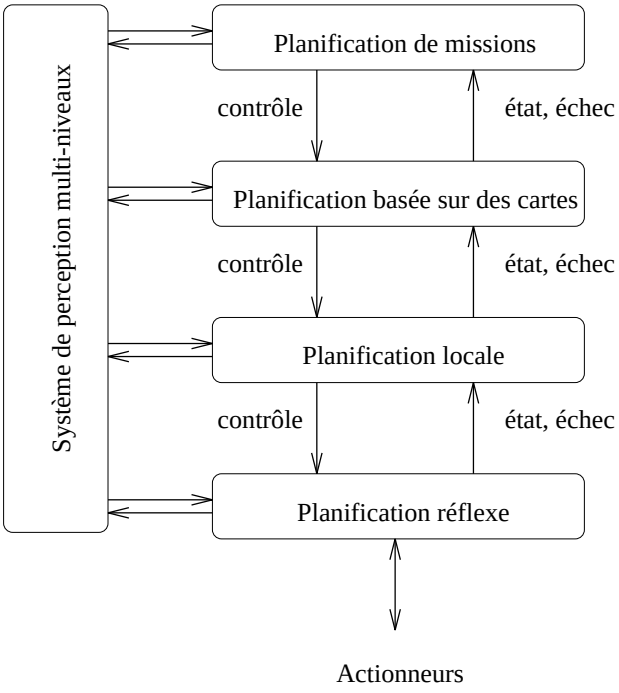


FIG. I.9 –: *L'architecture modulaire de Payton*

La *planification de missions* demande d'importants traitements sur les informations proprioceptives et extéroceptives pour mettre à jour les modèles qui lui sont nécessaires. Par contre, la *planification réflexe* consiste à traduire directement les informations capteurs en commandes sur les effecteurs pour assurer la réactivité du robot. Chaque module de planification est constitué d'un ensemble

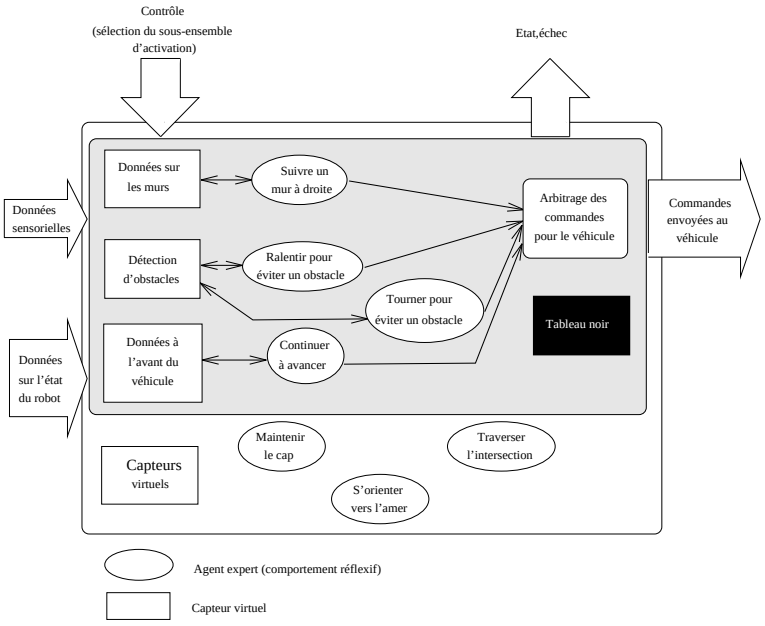


FIG. I.10 –: *Sous-ensemble d'activation du module de planification réflexe dans l'architecture de Payton*

d'agents experts regroupés en sous-ensembles, dits d'activation (figure I.10). Le contrôle d'un module consiste en la sélection par le module supérieur d'un sous-ensemble d'activation. La figure I.10 représente un tel sous-ensemble d'activation du module de planification réflexe. Les agents experts ou comportements réflexifs (*suivre un mur à droite, continuer à avancer, ...*) constituent un lien direct entre perception et commande par l'intermédiaire de capteurs virtuels. Ces capteurs virtuels correspondent à un traitement spécifique des données nécessaires au comportement qui y est connecté. Les agents s'exécutent simultanément et proposent chacun une commande dans un tableau noir. En cas de conflit, la commande effective est déterminée par un arbitre s'appuyant sur une notion de priorité, a priori fonction du degré d'importance de chaque comportement.

Points forts : Les avantages de la méthode résultent dans la relative facilité d'ajouter des comportements nouveaux et sur la possibilité d'effectuer différents traitements sur les données issues des capteurs pour satisfaire la demande des différents niveaux de l'architecture (différents niveaux d'abstraction des données).

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

4.2. L'architecture d'Arkin

Contrairement à l'architecture de Payton, qui a introduit de la réactivité à travers le module de *planification réflexe*, Arkin [1] a lui ajouté à un niveau réactif des niveaux délibératifs. Cette architecture s'appuie sur le concept de *schémas moteurs* et de *schémas perceptuels* assurant le couplage "perception-action". Les schémas sont utilisés en éthologie pour décrire de tels couplages.

Les schémas moteurs

Les schémas moteurs sont des spécifications de comportement générique. L'instanciation de ces schémas génériques fournit des actions pouvant être exécutées par le robot. Chaque schéma moteur propose un déplacement sous la forme d'un vecteur représentant la direction et la vitesse du prochain déplacement devant être effectué par le robot. Les sorties de tous les schémas sont ensuite combinées par sommation vectorielle pour déterminer le déplacement effectif du robot.

Les schémas perceptifs

Les schémas perceptifs sont associés à des schémas moteurs et ont pour tâche de fournir les informations pertinentes à ces derniers pour leur exécution, et ce, à partir des informations délivrées par les capteurs.

L'architecture AuRA

L'architecture hiérarchique AuRA[3] (figure I.11) a été développée par Arkin et s'appuie sur l'utilisation de schémas. A partir de spécifications provenant d'un

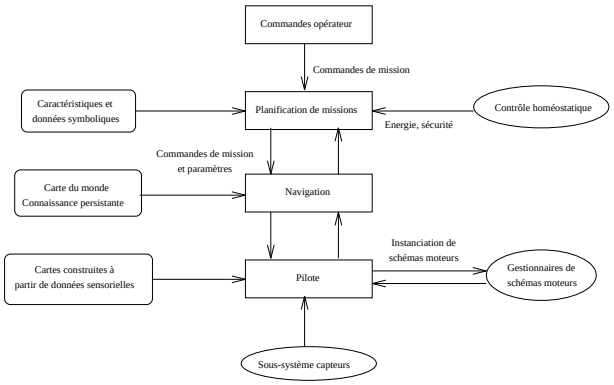


FIG. I.11 –: *L'architecture AuRA*

opérateur et de considérations homéostatiques (prises en compte d'informations proprioceptives du robot), un planificateur de missions délivre un ensemble de

3. Autonomous Robot Architecture.

buts à atteindre. Ceux-ci sont fournis à un module de navigation qui, par utilisation d'une carte, délivre un chemin permettant de les satisfaire. Enfin, un pilote, après instanciation des schémas moteurs et perceptifs, assure le déplacement du robot selon le chemin fourni et en contrôle la réalisation.

Points forts : La distributivité du contrôle (à travers les schémas moteurs) et la démonstration de son activité semblent présenter des avantages conséquents.

Points faibles : Un problème concernant la meilleure instanciation des schémas moteurs peut exister. On rejoint en cela le problème de la détermination des agents experts dans l'approche de Payton précédemment exposée.

5. Contrôleurs d'exécution

Nous venons de voir, dans ce qui précède, que les architectures hiérarchiques et hybrides disposent de modules de haut niveau, comme la génération de missions et la planification. De tels niveaux travaillent sur des états instantanés de l'environnement, pour déterminer les prochains mouvements du robot sur un certain horizon temporel T. Pour cela, une prédiction sur l'évolution de l'environnement considère généralement que l'état de celui-ci va rester invariant sur T. Cependant, lors de l'exécution des mouvements proposés par ces modules, des modifications de l'environnement et/ou du robot peuvent survenir. On peut citer, entre autres, des obstacles imprévus ou dont le comportement diffère de la prédiction, ou des aléas au niveau du robot, comme des phénomènes de glissement ou de patinage des roues. Il est donc nécessaire de disposer d'un niveau de contrôle d'exécution permettant de vérifier, voire d'amender localement, les séquences d'actions prévues par les modules de haut niveau.

De nombreux contrôleurs d'exécution existent dans la littérature. Nous allons présenter quelques-uns d'entre eux.

5.1. Robot planétaire du JPL

Au sein de l'architecture de contrôle du robot planétaire du Jet Propulsion Laboratory [26], un planificateur local délivre un chemin à partir d'informations perceptives. Ensuite, pour chaque capteur du robot, les valeurs attendues tout au long du chemin sont calculées. Ces valeurs permettent d'établir des profils d'exécution correspondant, pour chaque mesure de capteur, aux valeurs limites autorisées autour des valeurs attendues (cf. figure I.12). Ensuite, le contrôle d'exécution proprement dit consiste à vérifier, lors du déplacement effectif du robot, que les valeurs délivrées par les différents capteurs ne dépassent pas les valeurs limites. Si tel est le cas, une action réflexe associée est activée.

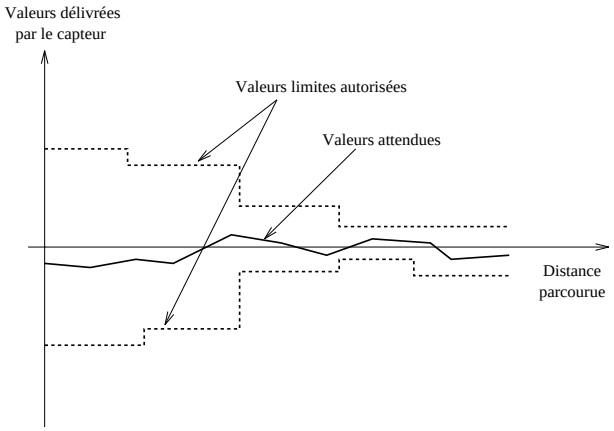


FIG. I.12 –: *Profil d'exécution associé à un capteur dans le contrôleur d'exécution du JPL*

5.2. Robot d'intervention AMR

Le robot d'intervention AMR du LAAS et de Matra-Espace fait appel à une ou plusieurs procédures pour réaliser une tâche. C'est le rôle du contrôleur d'exécution de déterminer, pour une tâche donnée, la procédure associée la plus adaptée et ce, en fonction du contexte. Au sein d'une procédure, les actions permettant de réaliser la tâche sont décrites à l'aide d'un script [46]. Ce script (cf. figure I.13) permet également de décrire les actions réflexes attendues en réponse à des événements asynchrones (mécanismes de surveillance).

5.3. Robot à chenilles RAMI

Dans l'architecture de contrôle du robot à chenilles RAMI, un plan sous la forme de primitives, est fourni au contrôleur d'exécution qui en assure le séquencement [62]. Une primitive est constituée :

- de conditions d'activation de la primitive;
- de conditions sur le domaine de validité de la primitive;
- d'une action associée à la primitive.

Les conditions d'activation et de validité constituent des expressions logiques appelées *primitives de perception*. Durant l'exécution de l'action liée à la primitive courante, un module de surveillance contrôle la primitive de perception associée. En fonction des valeurs courantes des conditions, le contrôleur :

- indique une erreur à l'opérateur et arrête le robot si une des conditions sur le domaine de validité de la primitive est devenue fausse;

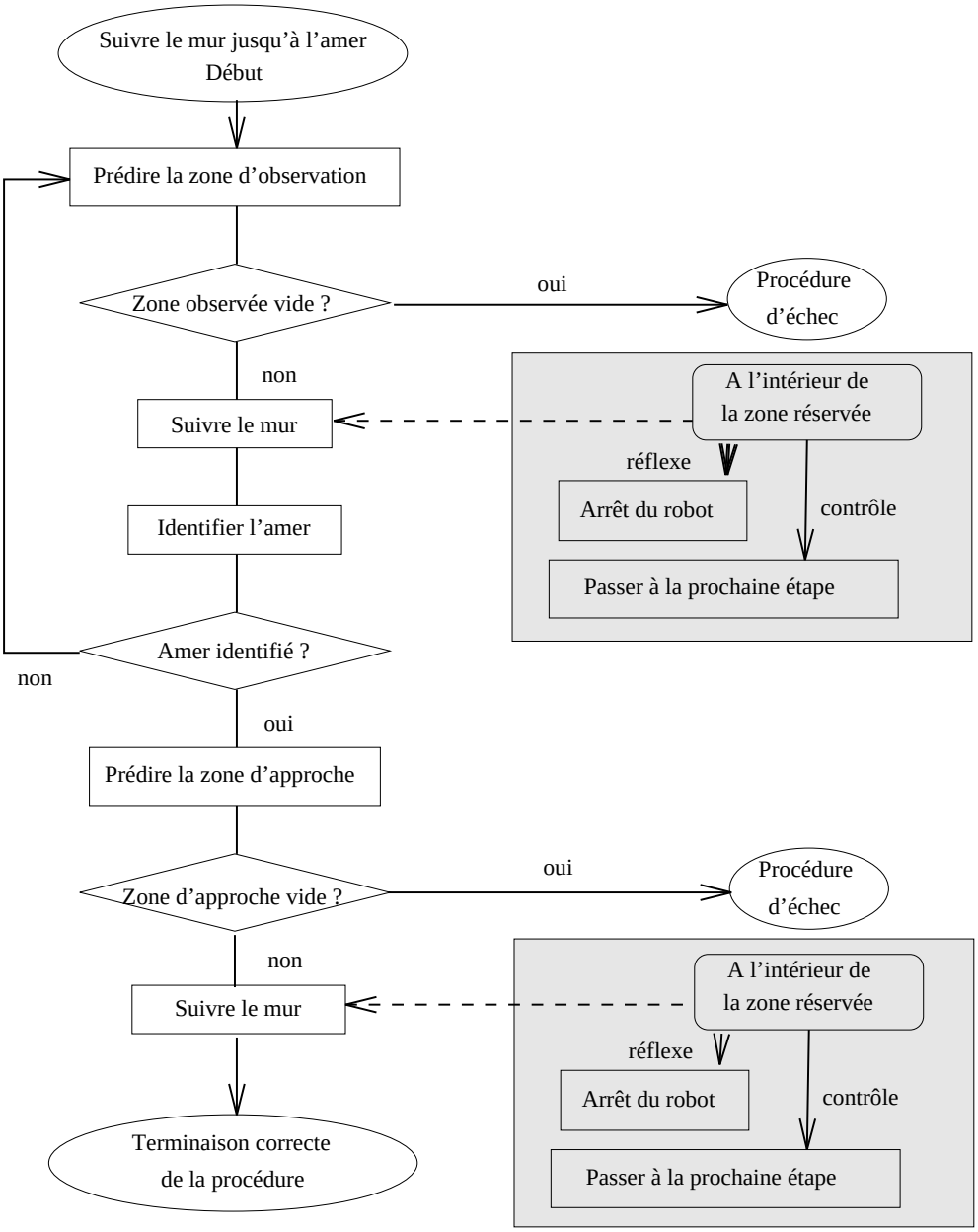


FIG. I.13 –: *Robot AMR : script de la procédure "suivre le mur jusqu'à l'amer"*

– active l'action associée à la primitive dont une des conditions d'activation est devenue vraie. Dans la mesure où plusieurs actions de primitives peuvent être activées simultanément, seule l'action associée à la primitive de plus haute priorité sera effectivement activée.

5.4. Le démonstrateur Prolab2

L'équipe Sharp du Lifia a développé dans le cadre de projet européen Eureka PROMETHEUS un contrôleur d'exécution de mouvements qui a été implanté sur le démonstrateur Prolab2 [32]. Ce contrôleur est constitué de trois modules : le pilote, le manager et l'exécuteur.

Le pilote
Le pilote représente le niveau symbolique du contrôleur. Il est constitué d'un ensemble de règles de comportements répartis en classes permettant d'obtenir des schémas de réactions adaptés à des situations types. Il analyse périodiquement la situation courante pour adapter le plan nominal qui lui a été délivré par un module de planification. La sortie de ce module est une instruction symbolique de type *"ralentir", "s'arrêter", "dépasser", ...* Si l'analyse de la situation courante ne remet pas en question le plan nominal, l'instruction symbolique "suivre le plan" est générée.

L'exécuteur
L'exécuteur représente le niveau numérique du contrôleur. Il est basé sur le concept de champs de potentiels fictifs [40] pour générer des consignes de mouvements appropriées, en fonction de l'instruction symbolique courante fournie par le pilote. En fait, le champ de potentiel fictif U dans lequel évolue le robot est défini comme une fonction linéaire de potentiels fictifs :

$$U = a \times U_s + b \times U_d + c \times U_v$$

Le potentiel U_s permet de suivre le plan nominal délivré par le planificateur, tout en prenant en compte les obstacles statiques. Le potentiel U_d intègre la dynamicité des obstacles mobiles. Enfin le potentiel U_v permet de prendre en compte les contraintes temporelles associées au plan nominal. Les coefficients (a, b, c) sont associés à une instruction symbolique. Ainsi, à chaque instruction symbolique délivrée par le pilote, correspond un triplet (a, b, c). L'idée est de définir les importances relatives des différents champs de potentiels composant U, dans le but d'obtenir le comportement désiré. L'espace des consignes à envoyer au robot est un espace discrétisé défini par :

$$alK = \{-\dot{v}_{max}, -\dot{v}_{max}/2, 0, \dot{v}_{max}/2, \dot{v}_{max}\} \times \{-\dot{\phi}_{max}, -\dot{\phi}_{max}/2, 0, \dot{\phi}_{max}/2, \dot{\phi}_{max}\}$$

où $\dot{v}_{max}$ et $\dot{\phi}_{max}$ représentent respectivement l'accélération linéaire et la vitesse de braquage des roues du véhicule. Les consignes effectivement envoyées au robot seront celles permettant d'aller vers la configuration minimisant le potentiel U.

Le manager
Le rôle du manager est d'effectuer l'interface entre le pilote et l'exécuteur mais également entre le contrôleur lui-même et le module de planification. Ce module reçoit donc le plan nominal délivré par le planificateur pour le fournir au pilote. Il reçoit périodiquement les instructions symboliques fournies par le pilote. S'il s'agit d'une instruction de type *"échec"*, le manager demande un nouveau plan au planificateur; dans le cas contraire, l'instruction symbolique est transmise à l'exécuteur.

6. Notre architecture

L'architecture de contrôle du projet Sharp de l'INRIA Rhône-Alpes est une architecture hiérarchique hybride (figure I.14). Cette architecture, d'un point de vue fonctionnel, a déjà été utilisée dans le cadre du projet PROMETHEUS [32].

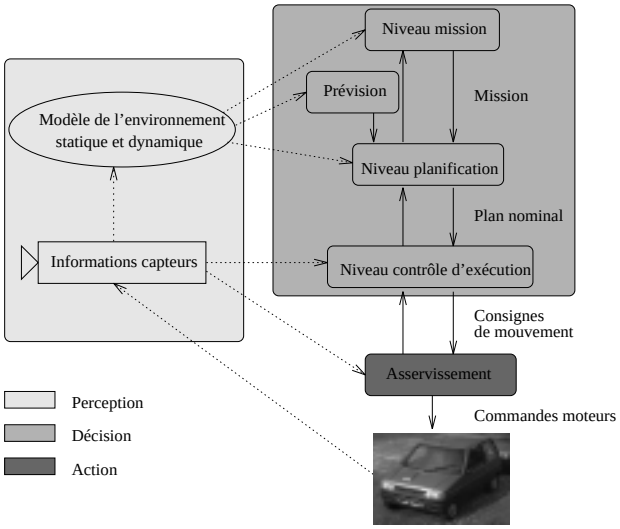


FIG. I.14 –: *L'architecture de contrôle de Sharp (Lifia/INRIA Rhône-Alpes)*

Cette architecture se décompose verticalement en modules de haut niveau (missions, planification) et de bas niveau (contrôle d'exécution, 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.

6.1. Fonctionnement

Le niveau mission détermine des buts que doit rejoindre 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 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.

6.2. 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 de notre équipe [17] est composé de deux modules complémentaires : un "planificateur de chemins" intégrant les contraintes géométriques et cinématiques, et un planificateur de trajectoires permettant la prise en compte de l'aspect dynamique.

Le planificateur de chemins

Ce planificateur permet de trouver un chemin géométrique sans manœuvre dans un environnement statique structuré (i.e composé de voies de circulation rectilignes pouvant s'intersecter). Le chemin, qui est une succession de segments et d'arcs de cercle, assure l'évitement d'obstacles statiques tout en intégrant les contraintes cinématiques du robot (rayon de giration borné par exemple). L'algorithme s'appuie sur la notion de "virage" permettant de passer d'une voie à une autre. Ces virages, de courbure constante[4], sont déterminés à travers la recherche de leur centre dans un espace dérivé de l'espace des *centres de giration* introduit par Laumond. La figure I.15 montre quelques résultats obtenus dans des environnements statiques structurés. En plus des obstacles polygonaux, apparaissent les voies de circulation sélectionnées sous la forme de segments de droites. En gras, est représenté le chemin *lissé* résultant de la prise en compte de la géométrie du

4. Cette courbure est choisie égale à $1/\rho_{min}$, ρ_{min} représentant le rayon de giration minimal du robot.

robot ainsi que de sa cinématique à travers le rayon de giration minimal (cf. sous-fenêtres). Une trace sous la forme de secteurs, illustre la construction des arcs de cercles composant le chemin.

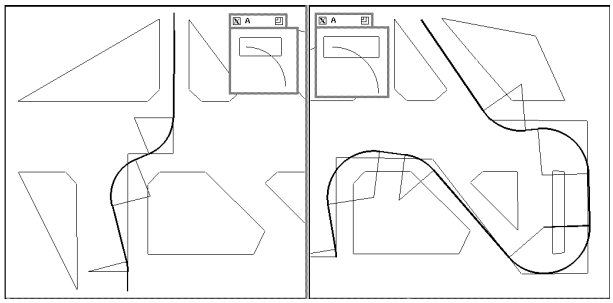


FIG. I.15 –: *Exemple de génération de chemin [17]*

Le planificateur de trajectoires

La seconde composante du planificateur consiste à doter le chemin trouvé précédemment d'une composante temporelle, à travers la détermination d'un profil de vitesse le long de celui-ci. L'espace de travail est maintenant dynamique (présence d'obstacles mobiles) et le robot subit de nouvelles contraintes dynamiques (vitesse et accélération bornées) et temporelle (temps associé à la trajectoire minimal). 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. L'algorithme utilisé travaille sur une discrétisation de *l'espace des états-temps* du robot (cf. figure I.16).

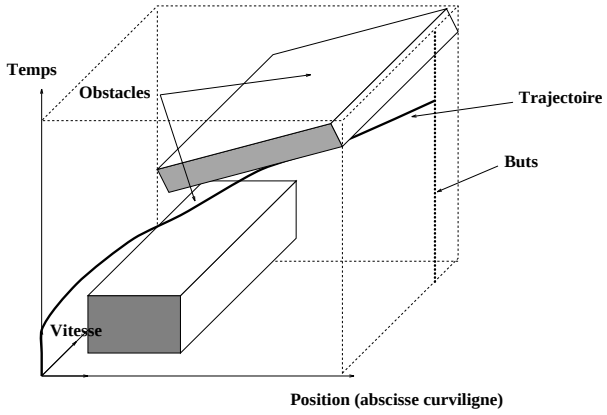


FIG. I.16 –: *Exemple de génération de trajectoire [18]*

Un point $(q, \dot{q}, t)$ de cet espace caractérise le véhicule dans la position q (abscisse curviligne le long du chemin) à la vitesse $\dot{q}$ et à l'instant t. L'algorithme consiste à trouver une trajectoire, sans collision avec les obstacles de cet espace, permettant de relier l'origine de celui-ci à un point état but $(q_{but}, \dot{q_{but}})$. La prise en compte de la composante temporelle transforme ce point en une droite dans *l'espace des états-temps* (cf. figure I.16). La discrétisation de l'*espace des états-temps* est obtenue en définissant un pas de temps et en choisissant les accélérations appliquées au mobile parmi un ensemble fini de valeurs. Le choix du pas de temps temporel détermine le nombre de points dans l'espace des états-temps discrétisé et par conséquent le temps moyen d'exécution de l'algorithme. La planification dans cet espace se ramène ainsi à la recherche du plus court chemin dans un graphe, l'algorithme utilisé étant alors du type A^*.

6.3. 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 doit être capable d'amender localement le plan nominal à travers des manœuvres d'évitement et de rattrapage de la trajectoire nominale.

Le contrôleur sur lequel nous avons travaillé est un contrôleur flou. Celui-ci sera détaillé par la suite (chapitre III). L'avantage de l'approche floue nous permet de coder le comportement attendu de la part du contrôleur d'exécution sous la forme de règles de type *système expert*. Ce point est très important dans la mesure où, comme le montrent de nombreuses applications utilisant la logique floue, le temps nécessaire à l'élaboration du contrôleur est très inférieur à celui que demandent d'autres approches. De plus, ceci nous permet de ne travailler qu'à un niveau symbolique contrairement, par exemple, au contrôleur du démonstrateur Prolab2 [32] qui a une composante fortement numérique à travers le module *exécuteur*. L'utilisation d'un contrôleur flou nous permet également, comme nous le montrerons ultérieurement, d'intégrer de manière élégante les informations imparfaites que l'on a quant à la description du processus à contrôler, tant au niveau des capteurs qu'à celui des actionneurs. Il est à noter que notre contrôleur d'exécution intègre directement, au sein d'un même module, les aspects réactifs à prendre en compte pour le robot (comme l'évitement d'obstacles, par exemple). Il est intéressant de préciser également que le contrôleur peut être facilement porté sur une architecture parallèle : sans entrer dans les détails ici, il est très possible de répartir les différentes règles du contrôleur sur les processus dont on dispose.

Notre approche diffère de celle défendue par [12] qui prône un contrôleur

d'exécution faisant un lien entre planification et composantes réactives du robot. Il nous semble en effet plus pertinent de travailler sur un contrôleur plus générique, effectuant directement le lien entre planification et asservissement, ne serait-ce que pour mieux caractériser le comportement attendu du robot.

7. Conclusion

Nous avons présenté dans ce chapitre les trois grandes catégories d'architectures de contrôle de robots ainsi qu'un certain nombre de contrôleurs d'exécution de mouvements tirés de la littérature. Il semble que les architectures hybrides, de par leur prise en compte tant des aspects de haut niveau (génération de missions, planification de trajectoires) que des aspects de bas niveau (contrôle réactif), soient les plus adaptées pour une architecture de contrôle d'un robot évoluant dans un environnement dynamique partiellement connu. Celles-ci permettent en effet de pallier les lacunes des architectures hiérarchiques peu réactives et des architectures purement réactives souffrant d'un manque de raisonnement (planification). Ces considérations nous ont amené à opter pour une architecture hybride. En ce qui concerne le contrôle d'exécution, nous avons choisi un contrôleur flou pour la relative simplicité d'écriture de celui-ci à travers des règles de type *système expert*, la prise en compte au niveau même du contrôleur des informations imparfaites (imprécisions notamment) et la possibilité de disposer d'un système relativement ouvert en ce qui concerne la mise à jour de la base de règles.

Chapitre II

Logique floue et contrôle

Le concept de logique floue est détaillé dans ce chapitre. En plus de la théorie de base définie par Zadeh, une attention particulière est portée à la notion de contrôle flou à travers la description du contrôleur flou générique de Mamdani que nous avons utilisé dans notre travail.

1. Introduction

Les connaissances de l'univers dans lequel nous évoluons sont généralement imparfaites [7] dans la mesure où elles peuvent souffrir d'*incertitudes* et/ou d'*imprécisions*, ne serait-ce qu'à travers la perception que nous en avons. Or, nous pouvons constater que l'homme intègre naturellement ces imperfections dans la vie de tous les jours, en particulier au niveau du raisonnement et de la décision. L'idée de Zadeh, en 1965, 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 i.e. prendre en compte les états intermédiaires entre le tout et le rien. L'utilité de cette approche peut être illustrée de la manière suivante :

une température de 10^{o}C, pour un humain, est généralement considérée comme froide; une température de 40^{o}C est, elle, qualifiée de chaude. Si chacune de ces valeurs appartient à une catégorie (ensemble) bien définie, qu'en est-il pour des valeurs intermédiaires? Une réponse intuitive consiste à affirmer qu'elles appartiennent à une ou deux des catégories précédentes avec des niveaux (normalisés i.e. définis sur [0,1] [1]) différents. On évite ainsi des transitions rigides entre différentes catégories, comme cela est le cas en logique binaire (cf. figure II.1). Il semble en effet surprenant de considérer qu'une température de 40^{o}C est chaude, alors qu'une température de $39,9^{o}$C ne l'est pas. Cet exemple d'école permet

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

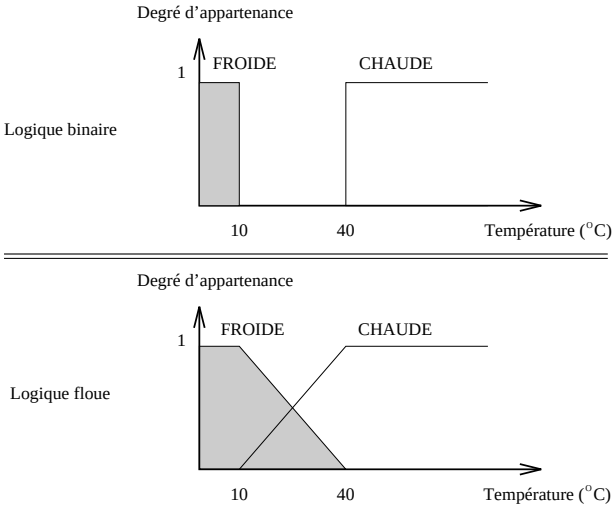


FIG. II.1 –: *Exemple de définition d'ensembles sur un univers de discours en logique binaire et en logique floue*

d'illustrer le fait qu'une logique binaire classique est, dans certains cas, trop restrictive. Il est nécessaire de faire appel à une logique multivaluée qui sera vue comme une extension de la précédente.

En ce qui concerne le contrôle d'un processus quelconque, la logique floue permet une approche novatrice par rapport à l'automatique classique. En automatique, en général, on s'attache à modéliser le processus au travers d'un certain nombre d'équations différentielles. Cette modélisation est rendue difficile à mesure que la complexité des processus à contrôler augmente. D'une manière radicalement opposée, un contrôleur flou va décrire non pas le processus mais la façon de le contrôler, tout comme le ferait un expert humain à travers des règles intégrant naturellement imprécisions et incertitudes. Dans cette approche, nous parlerons donc de systèmes experts flous basés sur des règles de production de la forme "**si** *prémisse* **alors** *conclusion*". Ces systèmes sont des extensions des systèmes experts classiques dans la mesure où ils intègrent des connaissances imparfaites.

Un certain nombre d'applications utilisant la logique floue ont vu le jour ces dernières années. Les plus médiatisées sont certainement les réalisations des chercheurs et industriels japonais qui, depuis les années 80, se sont intéressés notamment au contrôle/commande de processus. Ces applications s'appuient sur les travaux de Mamdani [48, 51, 49, 50], qui fut certainement le premier à voir la potentialité de la théorie des sous-ensembles flous dans ce domaine. Nous pouvons citer des biens de consommation courante[2] dont le terme flou a même constitué un élément de marketing certain, l'automatisation du métro de Sendai en 1988, ... et le fameux hélicoptère de Sugeno dans le milieu scientifique [66]. En France,

2. Machine à laver de Panasonic, caméra vidéo de Sanyo, aspirateur d'Hitachi, télévision de Sony, climatiseur de Mitsubishi, ...

le flou a été utilisé entre autre pour la conduite de hauts fourneaux à Fos-sur-mer et Dunkerque, le contrôle de niveau dans une usine de raffinage du pétrolier Elf. Dans l'industrie du transport automobile, des travaux utilisant la logique floue existent également : on peut citer Siemens Automotive travaillant en collaboration avec le laboratoire LAAS à Toulouse, Renault et son approche hybride entre le flou et les technologies classiques, les programmes PROMETHEUS(Eureka) et Drive au niveau notamment des suspensions actives. Ces exemples illustrent la percée du concept de logique floue dans le domaine industriel.

2. Les concepts

Le concept de la théorie des sous-ensembles flous [3] (et par extension, la logique floue [4]), 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 degré (éventuellement nul). En résumé, pour un sous-ensemble A défini sur un univers de discours U, on peut écrire :

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

Par extension, ce nouveau concept définit une logique multivaluée qui apparaît comme une généralisation de la logique binaire. Nous allons maintenant définir un certain nombre de termes propres au domaine de la logique floue auxquels nous pourrons nous référer, si nécessaire, dans la suite de ce chapitre.

Sous-ensemble flou

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 II.2 à travers sa fonction caractéristique μ_A. Il peut également être décrit par un certain nombre de caractéristiques comme :

- son support :

$$support(A) = \{x \in U / \mu_A(x) \neq 0\}$$

3. 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.

4. Il existe un lien étroit entre le langage ensembliste et la logique.

– sa hauteur :

$$hauteur(A) = \sup_{x \in U} \{\mu_A(x)\}$$

– son noyau :

$$noyau(A) = \{x \in U / \mu_A(x) = 1\}$$

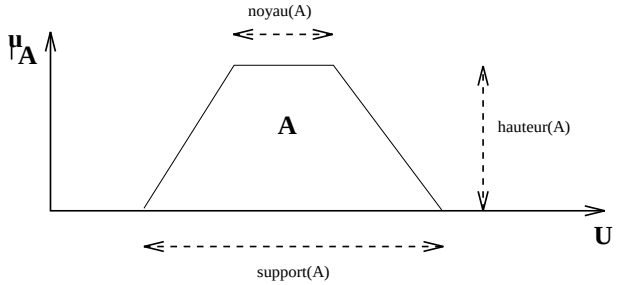


FIG. II.2 –: *Représentation d'un sous-ensemble flou et principales caractéristiques*

Un sous-ensemble flou permet de représenter différentes notions, en particulier la spécificité et la précision [7]. Ces différentes notions apparaissent dans la figure II.3.

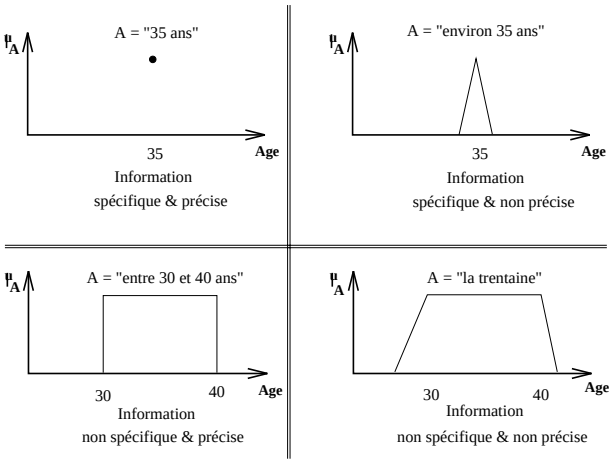


FIG. II.3 –: *Notions de spécificité et de précision représentée à l'aide de sous-ensembles flous*

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 des instanciations possibles de la variable V : il s'agit de sous-ensembles flous repérés par leur label A_i : on écrit ainsi $T_V = \{A_1, A_2, \ldots, A_n\}$. Graphiquement, une variable linguistique peut être représentée comme dans la figure II.4.

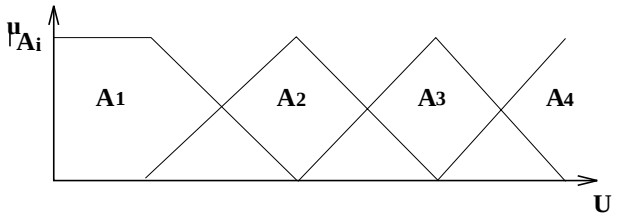


FIG. II.4 –: *Représentation d'une variable linguistique définie comme* $\{V, U, T_V = \{A_1, A_2, A_3, A_4\}\}$

Caractérisation floue

Une **caractérisation floue** d'une variable linguistique (V, U, T_V) est un label A_i de sous-ensemble flou appartenant à T_V. Par la suite, cette dénomination qualifiera indifféremment un sous-ensemble flou ou son label.

Propositions et règle floue

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, "*taille_de_Paul* est MOYENNE" est une proposition élémentaire définie à partir de la variable linguistique (taille_de_Paul, {tailles}, {...,MOYENNE,...}). 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.
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 et (**V_3 est A_3**) *conclusion.*

Activation d'une règle floue
Une règle ne peut être activée (i.e. intervenir dans le processus d'inférence) que lorsque la valeur de vérité de la proposition floue constituant sa prémisse est non nulle.

3. Architecture classique d'un contrôleur flou (FLC)

L'architecture classique d'un contrôleur flou (FLC[5]), proposée par Mamdani [51], est illustrée dans la figure II.5 tirée de [4].

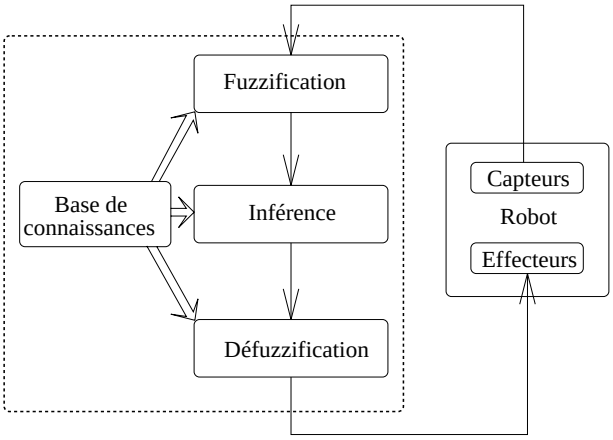


FIG. II.5 –: *Architecture d'un contrôleur flou*

Comme nous pouvons le voir dans la figure II.5, un contrôleur flou est composé de quatre parties :

1. **la fuzzification**[6] : les valeurs délivrées par les capteurs du robot, décrivant l'état du système contrôlé (en l'occurrence le robot), sont traduites en labels

5. Acronyme de Fuzzy Logic Controller.
6. Faute de trouver une traduction correcte, nous garderons cet anglicisme.

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.

4. **la défuzzification** [5] : 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.

3.1. Fuzzification

Dans la littérature, l'étape de fuzzification est qualifiée de transformation permettant de passer :

- d'un niveau discret à un niveau continu [38];
- d'un niveau numérique/quantitatif à un niveau symbolique/qualitatif[7, 4].

Chaque instanciation d'une variable d'entrée réelle du contrôleur va être traduite en un label caractérisant la variable linguistique associée. Par exemple, un âge de 30 ans sera traduit en JEUNE, label de la variable linguistique (âge, {âges}, {JEUNE, ..., AGÉ}). Ainsi une information ne sera plus caractérisée par une valeur numérique, mais par un terme linguistique comme JEUNE, si l'on considère un âge 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 II.6 illustre le mécanisme de fuzzification pour deux éléments appartenant au domaine des âges.

Ainsi deux personnes ayant respectivement 30 et 40 ans appartiennent toutes deux à la catégorie "JEUNE" avec cependant des degrés d'appartenance différents ($\mu_{JEUNE}(30) = 0.66$ et $\mu_{JEUNE}(40) = 0.33$).

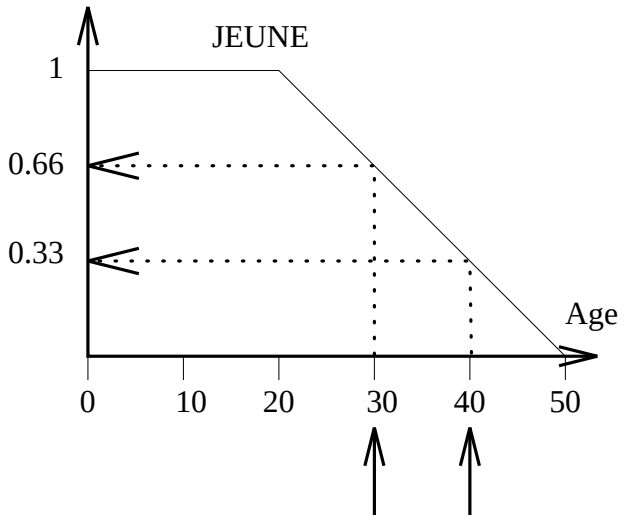


FIG. II.6 –: *Mécanisme de fuzzification*

3.2. 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.
Les conclusions de règles peuvent être de deux sortes [4, 20] :

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

  si x **est** A_1 **et** y **est** B_1 **alors** z **est** C_1

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

  si x **est** A_1 **et** y **est** B_1 **alors** $z = f(x, y)$

3.3. 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.

Les deux méthodes d'inférence les plus couramment utilisées sont le MINIMUM et le PRODUIT (figure II.7). L'inférence MINIMUM [8] consiste à tronquer à la valeur $\mu_{prémisse}$ la caractérisation floue associée à une variable de sortie, alors que l'inférence PRODUIT [9] revient à affecter à celle-ci un facteur d'échelle correspondant à $\mu_{prémisse}$.

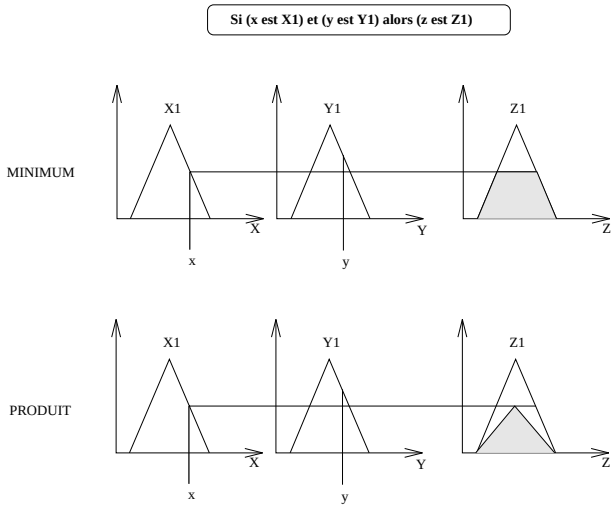


FIG. II.7 –: *Inférence "MINIMUM" et "PRODUIT"*

3.4. 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 ([38]) en particulier les compositions MAXIMUM (en général couplée avec l'inférence MINIMUM) et SOMME (en général couplée avec l'infé-

8. Correspond à l'implication de Mamdani (cf. précédemment).
9. Correspond à l'implication de Larsen (cf. précédemment).

rence PRODUIT)[10]. 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 II.8). On notera que la seconde méthode ne

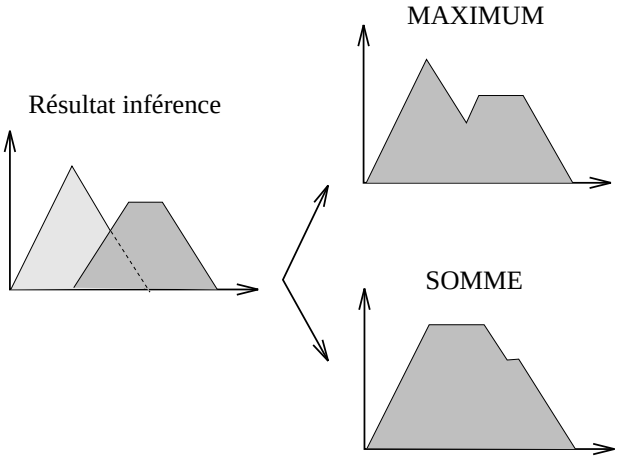


FIG. II.8 –: *Compositions de valeurs floues issues de l'inférence*

garantit pas une normalisation de la caractérisation floue résultant de la composition.

b) Passage symbolique → 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[11]
  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 :

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

10. 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.

11. 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, dans la littérature, l'on recourt souvent à des fonctions trapézoïdales, 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 II.9). Dans ce cas, la valeur numérique sera obtenue de la manière suivante :

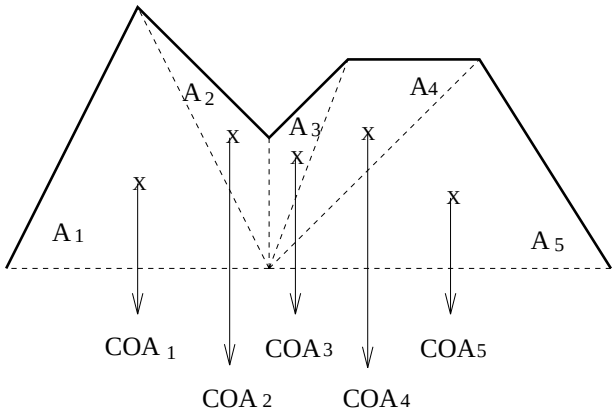


FIG. II.9 –: *Calcul du centre de gravité dans le cas de fonctions d'appartenance simples*

$$valeur = \frac{\sum_{i} COA_i \times A_i}{\sum_{i} A_i}$$

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 II.10).

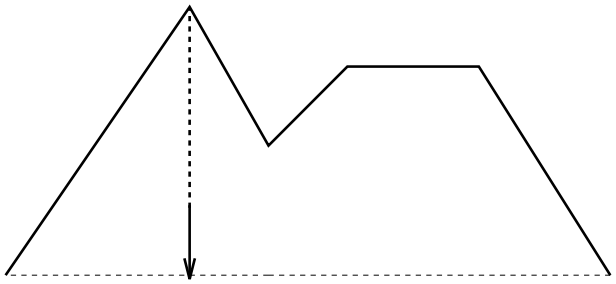


FIG. II.10 –: *Défuzzification par la méthode du maximum*

– la méthode de la moyenne des maxima [12]
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 II.11).

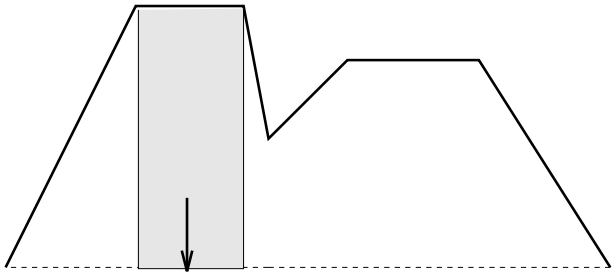


FIG. II.11 –: *Défuzzification par la méthode de la moyenne des maxima*

Une étude comparative de ces différentes approches peut être trouvée dans [47]. 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

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

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

4. 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 modeste aperçu de ces applications.

4.1. La voiture floue de Sugeno

La voiture de Sugeno est capable de se garer automatiquement dans un garage [68, 67]. Les règles du contrôleur flou proviennent d'une modélisation des actions humaines dans une telle situation. Les entrées du contrôleur sont au nombre de trois : la distance au mur de devant (x), la distance au mur latéral (y) et l'angle θ que fait l'axe longitudinal du véhicule, par rapport à un repère associé à la scène. Les sorties sont également au nombre de trois, à savoir : le braquage en marche avant, le braquage en marche arrière et la vitesse du véhicule.

Dix-huit règles permettent de contrôler le braquage dans les mouvements en avant, seize dans les mouvements en arrière. Un exemple de règle pour le contrôle du braquage dans le cas d'une marche avant est :

$$If\ x\ is\ A,\ y\ is\ B,\ \theta\ is\ C\ then\ f = p_0 + p_1 x + p_2 y + p_3 \theta.$$

L'identification des paramètres p_i résulte d'un apprentissage supervisé sur des échantillons collectés lors de manœuvres effectuées par des humains [67, 69].

4.2. Le métro de Sandai

Hitachi Ltd. a développé un système de contrôle du métro de Sandai (Japon). Ce contrôleur consiste à prédire les performances de chaque commande de contrôle candidate et sélectionne la plus appropriée en se basant sur une expérience humaine. Plus précisément, ce système est constitué de deux bases de règles prenant principalement en compte la sécurité, le confort, le suivi d'une vitesse de consigne, la consommation d'énergie et le temps. La première base de règles (CSC [13]) consiste à effectuer un asservissement en vitesse assurant le démarrage du métro et le maintien à une vitesse de croisière. La seconde base de règles

13. Constant Speed Control.

(TASC[14]) régule la vitesse du convoi afin que celui-ci s'arrête à une position donnée dans la station. Chacune des deux bases de règles contient douze règles floues. Le contrôle est effectué tous les 100ms. Le système se comporte d'une manière analogue à un contrôle humain; il s'avère cependant supérieur à un contrôleur de type *PID*[15] en termes de précision d'arrêt, consommation d'énergie et confort de conduite.

4.3. Autres travaux

Takeuchi et al. [70] proposent une méthode de navigation d'un robot mobile où les informations perceptives sont délivrées par une caméra CCD. Une analyse des images permet de déterminer des espaces libres entre des obstacles. La logique floue est utilisée pour déterminer, en fonction des paramètres de position, de largeur et de longueur des passages libres, le changement de direction à effectuer pour rallier un but. Dans [3], les auteurs utilisent des informations ultrasonores pour la navigation d'un robot mobile. L'architecture du contrôleur s'appuie sur un ensemble de modules (*Atteinte d'un point d'arrivée*, *Evitement des obstacles fixes*, *Evitement des obstacles mobiles*, *Demi-tour*, ...) activés par un routeur, en fonction de la carte de proximité associée aux capteurs ultrasons.

5. 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. Cependant, le problème majeur réside dans le fait que l'on ne puisse pas prouver la stabilité d'un système flou , contrairement aux modèles mathématiques classiques (fonctions de Lyapunov). Cela peut présenter des problèmes certains dans des domaines fortement contraints. En conséquence, au lieu de se limiter à des comparaisons stériles entre contrôleurs flous et autres plus formels, il semble intéressant de s'acheminer vers des approches hybrides intégrant différents types de contrôleurs. Au sein d'une même application, celles-ci pourront mettre en exergue les avantages d'élaboration d'un contrôleur flou et ceux de vérifications (stabilité, contraintes logiques, contraintes temporelles, ...) de contrôleurs plus "classiques". On peut ainsi imaginer des contrôleurs flous, chargés de tâches ne

14. Train Automatic Stop Control.
15. Proportionnal,Integral and Derivative.

souffrant pas de contraintes de sécurité, cohabitant avec des contrôleurs dont le comportement doit être prouvé de manière formelle. C'est dans cette optique que semble travailler le constructeur automobile Renault. En ce qui concerne notre travail, nous considérons que notre contrôleur d'exécution de mouvements peut être entièrement codé par un contrôleur flou dont les sorties pourront éventuellement être filtrées dans les situations nécessitant une sécurité absolue. Nous pensons par exemple aux situations d'arrêt d'urgence de notre véhicule contrôlé.

Chapitre III

Contrôleur d'exécution de mouvements

Ce chapitre présente notre contrôleur flou d'exécution de mouvements. Celui-ci est basé sur l'utilisation d'une base unifiée de règles floues pondérées codant un certain nombre de comportements (suivi de trajectoire, prise en compte de l'environnement statique et dynamique, ...). Chaque poids associé à une règle permet de définir l'importance de cette règle par rapport au reste de la base. Les règles floues codant les différents comportements sont données en annexe.

1. Problématique

Comme nous l'avons vu précédemment, l'architecture hybride, hiérarchique en couches, associée à la commande d'un véhicule, sur laquelle nous travaillons, est celle apparaissant dans la figure III.1.

De par son découpage fonctionnel, cette architecture est classique en robotique mobile. Le rôle des différents modules a été brièvement présenté en § I.6., page 23. Dans ce chapitre, nous allons décrire le fonctionnement du niveau correspondant au contrôleur d'exécution de mouvements. Au sein de cette architecture, nous rappelons que le module de planification permet de déterminer une trajectoire dans un environnement dynamique structuré partiellement connu, permettant ainsi de réaliser une mission déterminée par le gestionnaire de missions. Ce planificateur travaille sur un certain horizon temporel T; cela nécessite une prédiction des états futurs des environnements statique et dynamique sur cet horizon et ce, en fonction des informations perceptives caractérisant la situation courante. Une telle prédiction induit des incertitudes et imprécisions qui augmentent bien évidemment avec l'horizon temporel. Deux cas de figures apparaissent alors :

1. on modélise au mieux incertitude et imprécision dons un but de générer des trajectoires les plus sûres possibles. L'inconvénient majeur est que l'on

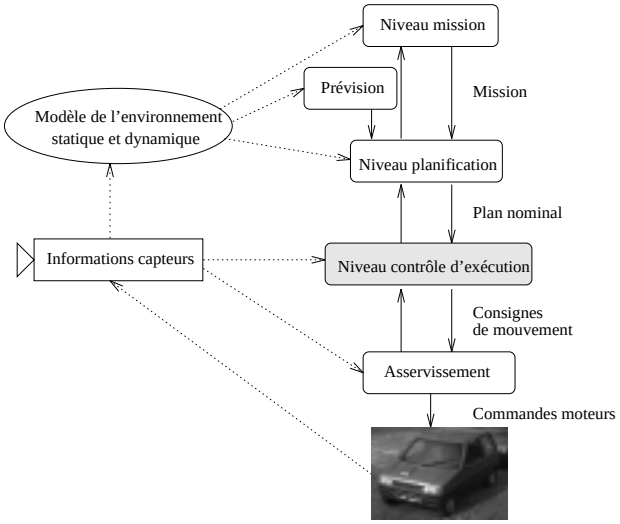


FIG. III.1 –: *Architecture de commande associée à un véhicule*

restreint fortement l'espace des solutions au point de ne plus avoir, dans certains cas, de trajectoire nominale générée.

2. on fait un certain nombre de suppositions simplificatrices sur l'évolution de l'environnement [1] et on confie à un contrôleur d'exécution la tâche d'adapter, si cela s'avère nécessaire, le plan proposé.

Nous avons opté pour la seconde approche : nous considérons que le planificateur délivre une trajectoire nominale (qui sera détaillée par la suite) que le contrôleur d'exécution devra suivre en l'amendant, si cela est nécessaire, de manière réactive pour s'adapter à la situation courante.

2. Notre contrôleur flou

Afin de réaliser notre contrôleur d'exécution de mouvements réactif, nous avons opté pour une approche *logique floue*. Comme dans le cas de l'approche à base de champs de potentiels fictifs que nous avons présentée en § I.5.4., notre contrôleur va s'appuyer sur une base de règles mais, dans notre approche, niveaux symbolique et numérique sont fondus au sein du système flou décrivant le contrôleur. L'élaboration de la base a été faite en quatre phases distinctes qui sont 1) le suivi de trajectoire, 2) la prise en compte de l'environnement statique, 3) la prise en compte de l'environnement dynamique et 4) l'activation du planificateur en cas d'échec du contrôleur. Nous allons préciser ces quatre phases en présentant

1. Par exemple, on suppose qu'un obstacle dynamique détecté conservera une vitesse constante sur l'horizon temporel considéré.

leur codage sous forme de règles floues et les résultats obtenus. Les règles codant les différents comportements sont données en annexe.

2.1. Modifications par rapport à un FLC classique

Nous avons vu dans le chapitre précédent en quoi consiste un contrôleur en logique floue (FLC), à travers la description du contrôleur de Mamdani. Dans le cadre de notre travail, nous avons effectué deux modifications essentielles. La première porte sur la notion de **règles pondérées** et la seconde sur la **méthode de défuzzification**.

Règles pondérées

Lors de la mise au point de notre contrôleur flou, nous avons été confronté au problème suivant : bien que l'écriture des règles floues constituant la base soit relativement aisée, car plus ou moins intuitive, il s'est rapidement avéré difficile de faire cohabiter plusieurs comportements dans la même base. En effet, suivant la situation courante, certains comportements devaient avoir une importance qui ne ressortait pas de par l'écriture des règles. Il s'agissait donc de déterminer un paramètre de la base de connaissance, qui permette d'adapter au mieux le comportement global du contrôleur. Parmi tous les paramètres possibles[2], nous avons opté pour la détermination de poids associés à chaque règle floue de la base. L'idée intuitive est que chaque règle se voit dotée d'un poids représentant son importance relativement aux autres règles de la base. Dans un premier temps, la détermination de ces poids s'est faite de manière empirique. Cependant, comme nous l'introduirons à la fin de ce chapitre et le détaillerons dans le chapitre suivant, nous avons développé une méthode d'apprentissage supervisée automatique de ces poids permettant de s'affranchir de ces réglages fastidieux.

Méthode de défuzzification

Nous avons choisi une méthode de défuzzification différente de celles exposées dans le chapitre précédent. Ceci est en fait lié à trois considérations qui sont respectivement : (1) **intégrer la notion de règles pondérées** dans le processus, (2) **optimiser le contrôleur** en se dispensant de la phase de composition et (3) **mieux appréhender l'importance d'une caractérisation floue** dans le processus de défuzzification, lorsque celle-ci apparaît plusieurs fois à l'issue de la phase d'inférence.

La méthode peut être décrite comme suit :

1. pour chaque règle activée de la base, on détermine les aires des caractérisations floues des variables linguistiques de sortie du contrôleur. Ceci

2. Fonctions d'appartenance des sous-ensembles flous caractérisant les variables linguistiques du contrôleur, nombre de ces mêmes caractérisations, ...

correspond à la phase d'inférence. A ce niveau, chaque caractérisation floue est déterminée par :

- une aire A_i;
- un centre de gravité COA_i;
- un poids w_i qui n'est rien d'autre que le poids de la règle courante.

2. pour une variable de sortie v donnée, la valeur effective délivrée sera donnée par la formule :

$$v = \frac{\sum_{i=1}^{\#R} A_i \times COA_i \times w_i}{\sum_{i=1}^{\#R} A_i}$$

où $\#R$ représente le nombre de règles de la base portant sur la variable linguistique associée à v en partie conclusion[3].

Cette approche permet de s'affranchir de la phase de composition des caractérisations floues associées à une même variable. Par rapport à la composition MAXIMUM que nous avons présentée, et pour des poids w_i unitaires, la différence essentielle est que chaque caractérisation floue issue de la phase d'inférence est prise en compte autant de fois qu'elle apparaît en conclusion des règles activées. En effet, avec la méthode classique, on ne traite que l'enveloppe englobante de l'ensemble des caractérisations floues issue de l'inférence. La figure III.2 illustre la différence entre notre méthode de défuzzification et l'approche de type "composition MAXIMUM et méthode du centre de gravité".

On suppose que trois règles portant sur une même variable floue en conclusion sont activées. Les deux premières portent sur la même caractérisation floue de cette variable. Dans l'approche "composition MAXIMUM et méthode du centre de gravité", la valeur numérique résultant de la défuzzification correspond au centre de gravité de la caractérisation floue englobante. Par contre, avec notre approche, chaque caractérisation est considérée à part entière dans la détermination du centre de gravité.

2.2. Consignes de mouvement

Nous avons choisi comme consignes de mouvements le couple $(\dot{v}_{cons}, \dot{\phi}_{cons})$ où v représente la vitesse du point de référence R du véhicule (milieu de l'essieu arrière) et ϕ l'angle des roues directrices par rapport à l'axe longitudinal du véhicule.

3. En fait, ici, seules les règles activées nous intéressent mais la prise en compte de l'ensemble des règles ne modifie en rien le résultat v dans la mesure où la non-activation de règles provoquera une valeur nulle de la caractérisation floue A_i.

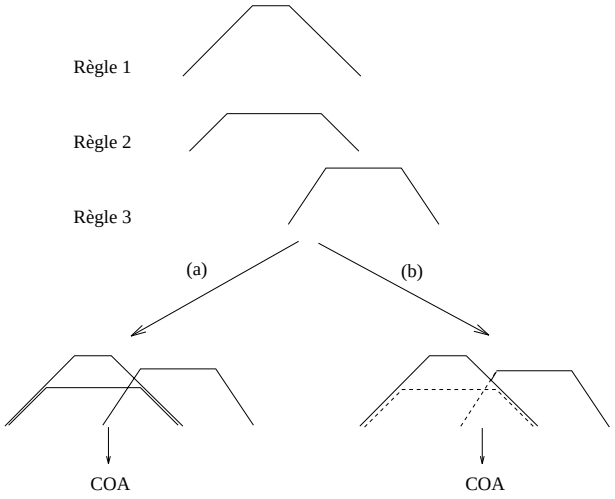


FIG. III.2 –: *Comparaison de notre méthode de défuzzification (a) avec une approche "composition MAXIMUM et méthode du centre de gravité" (b)*

2.3. Suivi de trajectoire

Dans un premier temps, nous avons réalisé un suivi de trajectoire nominale délivrée par le module de planification. Nous disposons pour cela d'une trajectoire de référence discrétisée (comme en [72]) sous la forme d'une séquence de configurations et d'états datés du véhicule, c'est-à-dire, à chaque instant :

- la configuration de référence $q_{ref}(t) : (x_{ref}(t), y_{ref}(t), \theta_{ref}(t))$;
- l'état de référence $\Omega_{ref}(t)$ i.e. vitesse $v_{ref}(t)$.

En complément de cette trajectoire de référence, nous disposons d'informations courantes, correspondant aux caractéristiques réelles du véhicule, comme :

- la configuration courante $q_{cour}(t) : (x_{cour}(t), y_{cour}(t), \theta_{cour}(t))$ délivrée par un module de localisation (odométrie par exemple);
- la vitesse courante $v_{cour}(t)$.

Notion de sous-but évolutif

Notre suivi de trajectoire va consister à essayer de rallier à chaque instant t le sous-but $G_{ref}(t)$ caractérisé par les configurations $q_{ref}(t)$ et état $\Omega_{ref}(t)$ de référence du véhicule au même instant. Nous voyons donc que le sous-but évolue continuellement (figure III.3). Ainsi, le suivi de trajectoire va consister en des minimisations de critères, chacune associée à un comportement, qui sont :

- la minimisation de l'écart entre configuration de référence et configuration courante noté $\|q_{ref}(t) - q_{cour}(t)\|$;

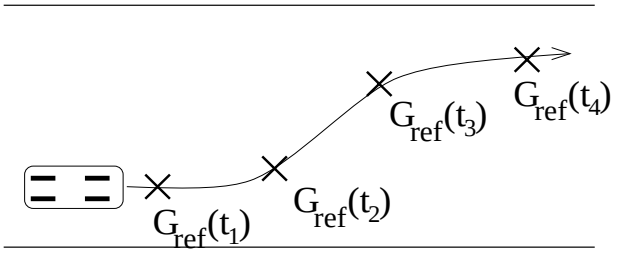


FIG. III.3 –: *Sous-but évolutif*

– la minimisation de la vitesse relative $|v_{ref}(t) - v_{cour}(t)|$.

Il est à préciser que chaque sous-but $G_{ref}(t)$ ne constitue pas forcément un point de passage obligé (hormis bien évidemment le but final). Il s'agit du seul cas où le véhicule ne se trouve pas sur la trajectoire de référence[4]. La figure III.4 illustre la récupération d'une trajectoire de référence, à la suite d'un évitement d'obstacle. Dans cette figure, les configurations courantes et nominales correspondant à un même instant, sont reliées par un segment.

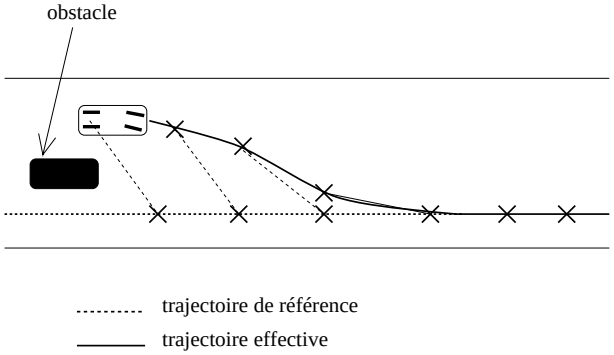


FIG. III.4 –: *Suivi de trajectoire*

Minimisation de la configuration relative $\|q_{ref}(t) - q_{cour}(t)\|$

Cette minimisation est assurée en deux étapes par :

1. une convergence en position
   Pour cela, nous considérons la position de référence $(x_{ref}(t), y_{ref}(t))$ en coordonnées polaires dans le repère du véhicule de configuration $q_{cour}(t)$, la composante radiale représentant l'erreur en distance et la composante angulaire, la direction à suivre (angle de cap) pour minimiser cette erreur. Une consigne de mouvement adaptée en accélération longitudinale et vitesse de braquage permet de réduire l'erreur en position. Cela est illustré

4. Cela pourra être le cas à la suite d'un évitement d'obstacle dont le comportement ne correspond pas à la prévision faite par le module de planification.

par la figure III.5. Le point de référence du véhicule est le milieu de l'essieu arrière.

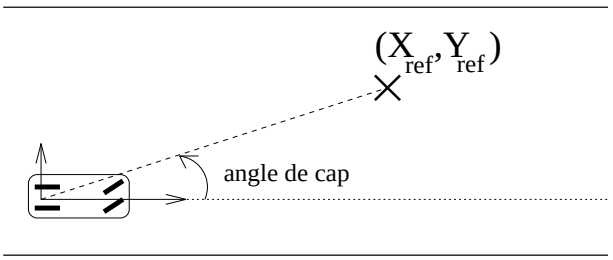


FIG. III.5 –: *Convergence en position*

2. une convergence en orientation
   En complément de la convergence en position, nous devons considérer également celle en orientation. Celle-ci est assurée par action sur la vitesse de braquage.

Minimisation de la vitesse relative $|v_{ref} - v_{cour}|$

A première vue, une minimisation de la configuration relative $\|q_{ref} - q_{cour}\|$ telle que nous venons de la décrire semblerait suffisante pour un suivi de trajectoire. Cependant, il apparaît rapidement nécessaire de prendre en considération la vitesse courante, vis-à-vis de la vitesse de référence, pour assurer une convergence asymptotique du véhicule vers la configuration de référence. En effet, une simple convergence en position provoquerait des oscillations du véhicule autour de la position désirée.

Comparaison avec la méthode de Kanayama

Afin de compléter les résultats obtenus en simulation au niveau du suivi de trajectoires, nous avons comparé notre contrôleur flou avec un contrôleur tiré de [37] que nous allons maintenant présenter.

Tout d'abord, une erreur de configuration q_{err} est calculée dans le système de coordonnées locales lié au mobile :

$$q_{err} = \begin{pmatrix} x_{err} \\ y_{err} \\ \theta_{err} \end{pmatrix} = \begin{pmatrix} \cos\theta_{cour} & \sin\theta_{cour} & 0 \\ -\sin\theta_{cour} & \cos\theta_{cour} & 0 \\ 0 & 0 & 1 \end{pmatrix} (q_{ref} - q_{cour}) \qquad \text{(III.1)}$$

où $q_{ref} = (x_{ref}, y_{ref}, \theta_{ref})^T$, représente la configuration de référence du véhicule et $q_{cour} = (x_{cour}, y_{cour}, \theta_{cour})^T$, sa configuration courante. L'objectif est ensuite de trouver une loi de contrôle qui minimise l'erreur de configuration. Les entrées

du système de contrôle sont $q_{ref}(t)$ et les vitesses de référence v_{ref} et ω_{ref} qui définissent la trajectoire en q_{ref}, toutes deux fonctions du temps.

Le vecteur de commande est donc de la forme :

$$\begin{pmatrix} v_{cons} \\ \omega_{cons} \end{pmatrix} = \begin{pmatrix} v_{cons}(q_{err}, v_{ref}, \omega_{ref}) \\ \omega_{cons}(q_{err}, v_{ref}, \omega_{ref}) \end{pmatrix} \tag{III.2}$$

soit, d'après [37] :

$$\begin{pmatrix} v_{cons} \\ \omega_{cons} \end{pmatrix} = \begin{pmatrix} v_{ref} \cos\theta_{err} + K_x x_{err} \\ \omega_{ref} + v_{ref}(K_y y_{err} + K_\theta \sin\theta_{err}) \end{pmatrix} \tag{III.3}$$

où K_x, K_y et K_θ sont des paramètres constants positifs.

Dans le cas où le véhicule ne s'arrête pas, la stabilité asymptotique du point d'équilibre $q_{err} = 0$ est prouvée au moyen d'une fonction de Lyapunov, et ce, quelles que soient les valeurs des paramètres K_x, K_y et K_θ. Pour une trajectoire rectiligne, le système contrôlé est alors équivalent à un second ordre en y_{err}, c'est à dire que :

$$\ddot{y_{err}} + 2\zeta\xi\dot{y_{err}} + \xi^2 = 0 \tag{III.4}$$

avec $\zeta = \dfrac{K_\theta}{2\sqrt{K_y}}$ le coefficient d'amortissement

$\xi = v_{ref}\sqrt{K_y}$ la pulsation propre (s^{-1})

Cette méthode de réglage des coefficients est d'ailleurs souvent admise pour des trajectoires non rectilignes.

Comme nous l'avons vu précédemment (2.2.), les commandes que nous utilisons sont du type $(\dot{v}_{cons}, \dot{\phi}_{cons})$. Nous allons donc effectuer le passage d'une commande $(v_{cons}, \omega_{cons})$ à la commande $(\dot{v}_{cons}, \dot{\phi}_{cons})$ correspondante.

Passage de v_{cons} à $\dot{v}_{cons}$:

$$\dot{v}_{cons} = \frac{v_{cons} - v_{cour}}{\Delta t} \tag{III.5}$$

Passage ω_{cons} à $\dot{\phi}_{cons}$: A partir des équations du mouvements associées au véhicule (plus précisément l'équation V.6) qui seront détaillées en § V.1.4., nous pouvons écrire :

$$\phi_{cons} = \arctan\left(\frac{l_w\ \omega_{cons}}{v_{ref}}\right) \tag{III.6}$$

ce qui nous amène à :

$$\dot{\phi}_{cons} = \frac{\phi_{cons} - \phi_{cour}}{\Delta t} \tag{III.7}$$

Dans ce qui précède, Δt correspond à une période d'échantillonnage du système. On constate en fait que les consignes attendues à un instant t seront effectivement appliquées à l'instant $t + \Delta t$. Cette approximation n'est pas préjudiciable,

si l'on considère une période d'échantillonnage faible par rapport à la dynamique du véhicule contrôlé.

Les figures III.6, III.7, III.8 et III.9 illustrent les résultats expérimentaux obtenus pour différents suivis de trajectoires[5] avec notre contrôleur flou et le contrôleur de Kanayama. Ces résultats portent sur la variation de la distance euclidienne entre la position réelle q_{cour} et la position de référence q_{ref} du véhicule. Les paramètres K_x, K_y et K_θ ont pour valeurs respectives 0.9, 1 et 1. Le véhicule réel apparaît en foncé et la configuration initiale de référence est en clair. La figure III.10 illustre la convergence en orientation du véhicule avec les deux méthodes.

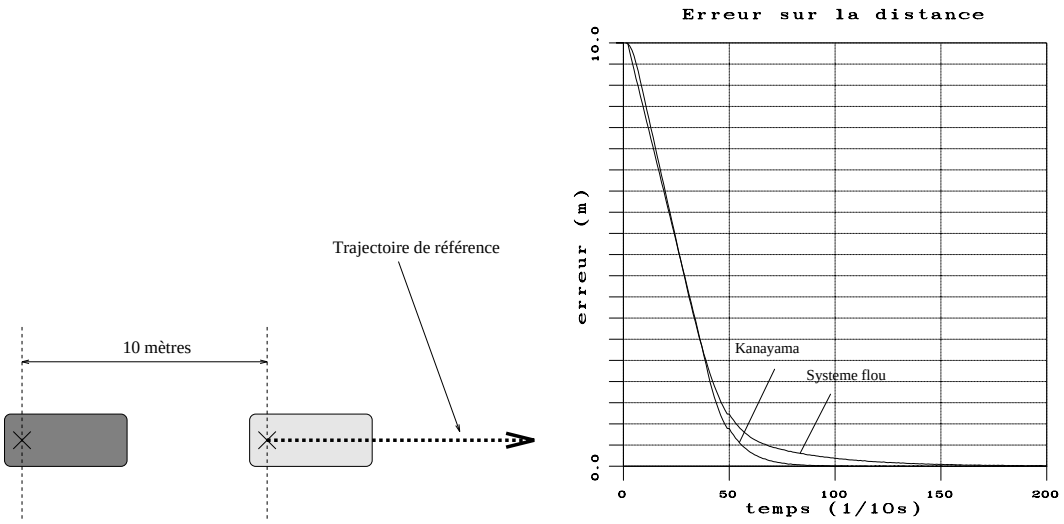


FIG. III.6 –: *Evolution de l'erreur en distance avec le contrôleur de Kanayama et notre contrôleur flou pour un suivi de trajectoire rectiligne avec une erreur initiale* $q_{err}(x_{err}, y_{err}, \theta_{err}) = (10, 0, 0)$

Comme nous pouvons le voir dans les figures III.6, III.7, III.8, III.9, III.10 et à travers les expériences que nous avons faites, notre contrôleur flou et celui de Kanayama ont des comportements relativement identiques. L'avantage du contrôle flou réside cependant dans le fait que nous allons pouvoir compléter notre suivi de trajectoire par de nouveaux comportements, en particulier la prise en compte d'obstacles tant statiques que dynamiques, et ce, au sein d'une même base de règles.

5. Il serait en fait plus pertinent de parler de rattrapage et de suivi de trajectoire dans la mesure où l'erreur de configuration initiale q_{err} peut être relativement importante. Cela peut, par exemple, être dû à un évitement d'obstacle.

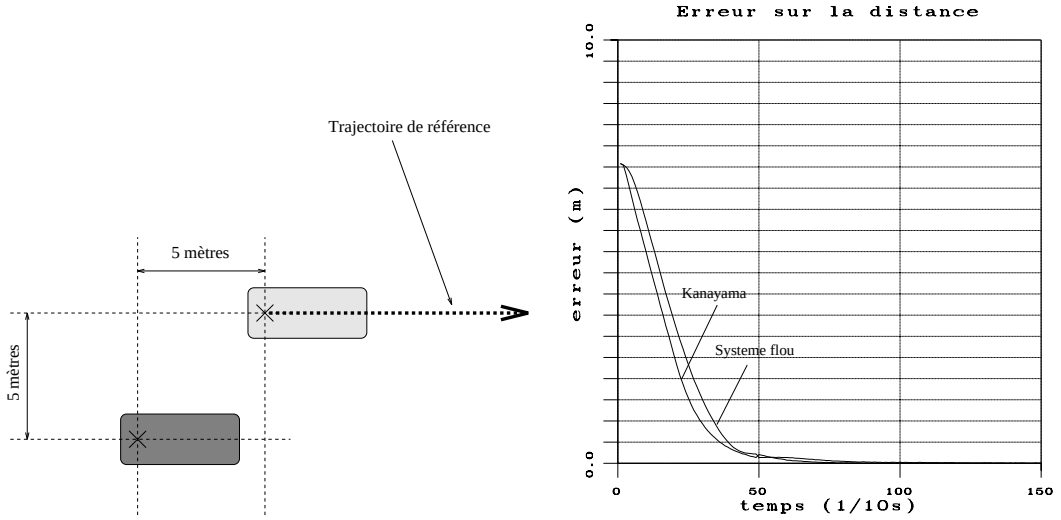


FIG. III.7 –: *Evolution de l'erreur en distance avec le contrôleur de Kanayama et notre contrôleur flou pour un suivi de trajectoire rectiligne avec une erreur initiale* $q_{err}(x_{err}, y_{err}, \theta_{err}) = (5, 5, 0)$

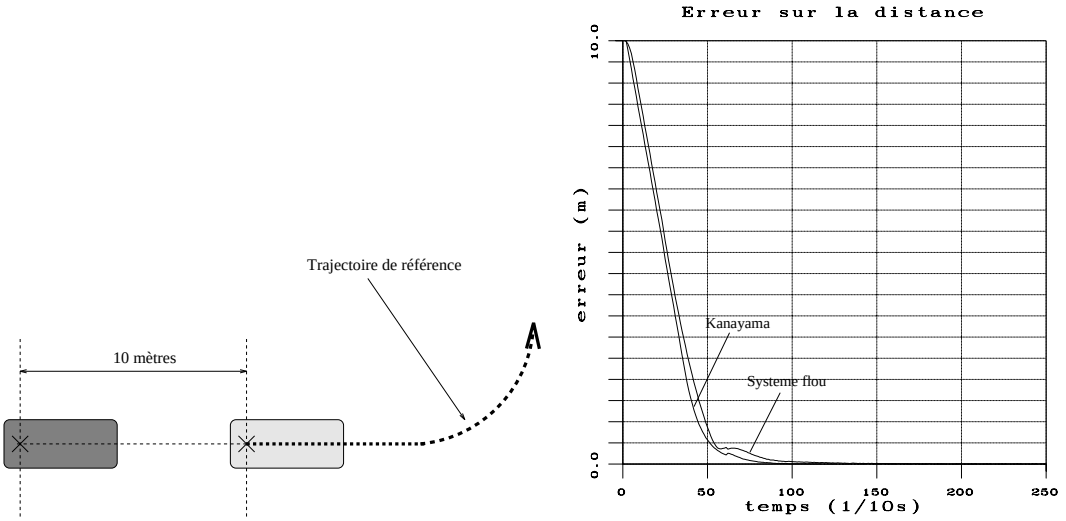


FIG. III.8 –: *Evolution de l'erreur en distance avec le contrôleur de Kanayama et notre contrôleur flou pour un suivi de trajectoire courbe avec une erreur initiale* $q_{err}(x_{err}, y_{err}, \theta_{err}) = (10, 0, 0)$

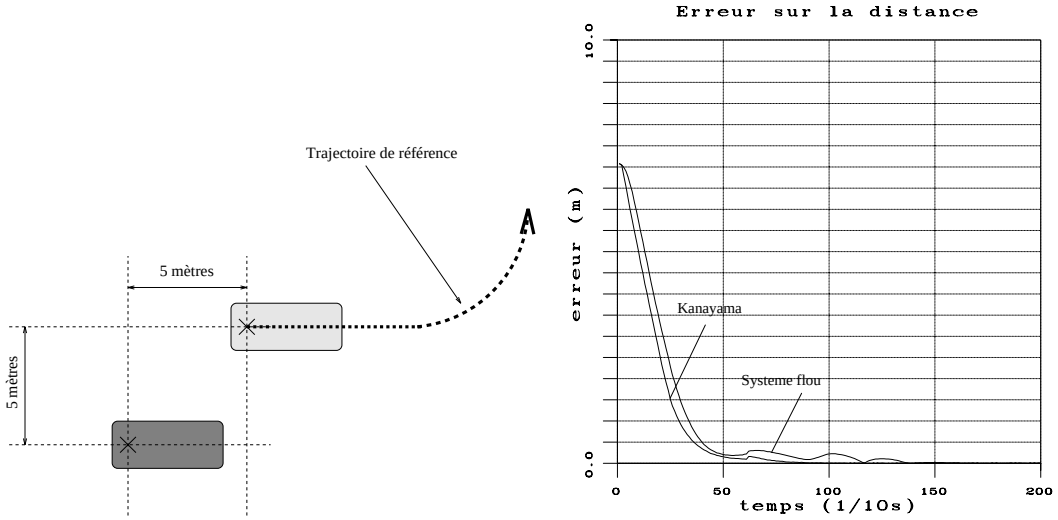


FIG. III.9 –: *Evolution de l'erreur en distance avec le contrôleur de Kanayama et notre contrôleur flou pour un suivi de trajectoire courbe avec une erreur initiale* $q_{err}(x_{err}, y_{err}, \theta_{err}) = (5, 5, 0)$

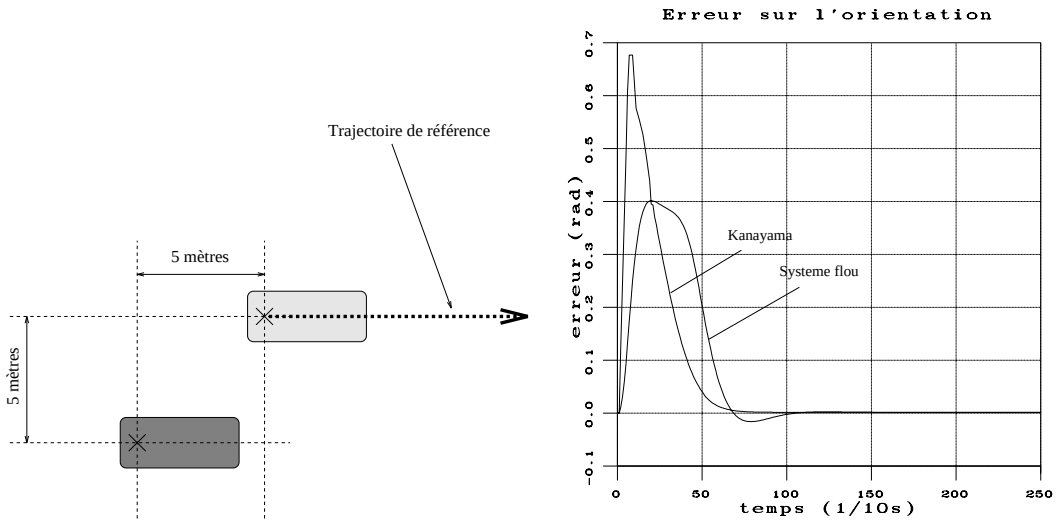


FIG. III.10 –: *Evolution de l'erreur en orientation avec le contrôleur de Kanayama et notre contrôleur flou pour un suivi de trajectoire rectiligne avec une erreur initiale* $q_{err}(x_{err}, y_{err}, \theta_{err}) = (5, 5, 0)$

2.4. Prise en compte de l'environnement local

Nous supposons que seule la connaissance de l'environnement local autour du véhicule est connue à travers un certain nombre de zones, appelées *zones d'intérêt* (figure III.11). Ces zones assurent un découpage relativement sommaire de

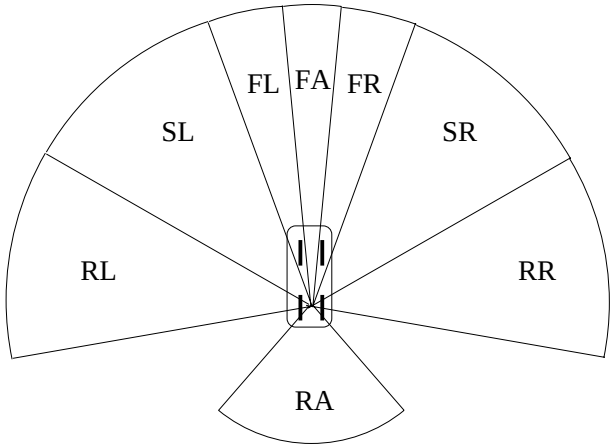


FIG. III.11 –: *Zones d'intérêt associée à un véhicule*

l'environnement correspondant ainsi à ce que pourrait être le champ de perception de capteurs de type *ultrasons* ou *caméras*. Ainsi, environnement statique et dynamique seront traités à travers la présence ou non d'obstacles dans ces zones d'intérêt. De plus, afin de ne pas travailler sur un modèle trop riche de l'environnement, un obstacle détecté dans une zone d'intérêt ne fournira comme information que la profondeur à laquelle il se trouve à l'intérieur de celle-ci. Ces informations seront ainsi plus proches d'informations capteurs de type *ultrasons*, capteurs communément utilisés pour la création de cartes de proximité. Le raisonnement flou sur ces zones d'intérêt se fera à travers une discrétisation de ces dernières comme le montre la figure III.12. Les comportements codés dans les règles floues seront directement liés au degré d'intrusion d'un obstacle à l'intérieur de ces zones. On retrouve ici une modélisation un peu analogue de celle des Zones Virtuelles Déformables utilisées dans [55].

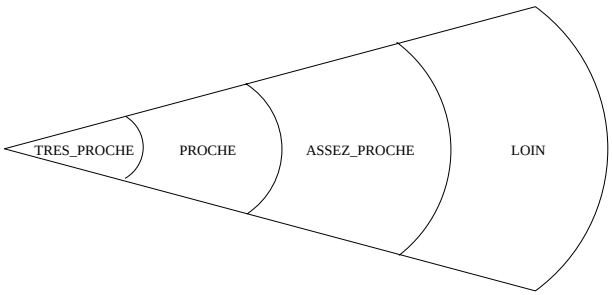


FIG. III.12 –: *Discrétisation d'une zone d'intérêt*

Environnement statique

L'environnement statique pourra être matérialisé non pas seulement par des murs mais, par exemple, par des terre-pleins ou des lignes au sol indétectables par des capteurs autres que des caméras. Il est donc nécessaire de disposer d'une carte locale de l'environnement. Cette carte sera en fait projetée sur les zones d'intérêt associées au robot. Pour cela, nous supposons que nous avons un module de localisation fiable s'appuyant sur de l'odométrie corrigée régulièrement par des opérations de recalage (amers).

En plus de ces obstacles connus a priori, il convient de prendre en considération des obstacles imprévus (comme, par exemple, un véhicule en panne) à travers une projection des informations capteurs (carte de proximité) sur les zones d'intérêt. Comme nous le verrons pour la prise en compte des obstacles dynamiques, nous supposons disposer d'un moyen de différencier obstacles statiques et dynamiques. Le comportement attendu du véhicule en présence d'obstacles statiques consiste à se diriger tout naturellement vers une région libre de l'espace.

Environnement dynamique

On peut penser que le traitement des obstacles dynamiques est le même que celui des obstacles statiques. Pour cette raison, nous retrouvons toutes les règles concernant l'environnement statique avec cependant la modification suivante : compte tenu des vitesses relatives des obstacles dynamiques, il est nécessaire de rallonger les distances de prise en compte des obstacles dynamiques (dans la limite de portée des zones d'intérêt). Pour cette raison, nous avons renommé les variables linguistiques associées et leurs caractérisations floues, bien que ces zones soient strictement identiques. Nous supposons être capable de différencier les obstacles dynamiques des autres, par intégration dans le temps des informations capteurs liées aux zones d'intérêt ainsi que des informations liées au véhicule contrôlé considéré(vitesse, braquage).

Plusieurs types de véhicules ont été pris en compte durant les expérimentations en simulation : des véhicules sans aucun contrôle, se contentant de suivre "en aveugle" des consignes de mouvements, et des véhicules dotés de systèmes de contrôle différents, en particulier notre contrôleur de mouvements flou.

Les expérimentations nous ont permis d'observer que, bien qu'aucune collision ne soit constatée, des situations de blocages pouvaient survenir. Celles-ci se sont avérées être facilement contournables en introduisant **une coopération passive** entre les véhicules ou en travaillant dans un **environnement structuré**. Nous présenterons plus loin ces extensions.

Prise en compte de la vitesse du véhicule

Il va de soi que la vitesse du véhicule doive avoir une influence sur la distance de prise en compte d'un obstacle par les règles concernées. L'idée intuitive est

la suivante : cette distance doit varier avec la vitesse i.e. augmenter quand celle-ci croît et diminuer quand elle décroît. Cette notion est partie intégrante de la construction des Zones Virtuelles Déformables de Zapata [74, 55] présentée en § I. Malheureusement, la discrétisation d'une zone d'intérêt (figure III.12) est figée au sein de notre contrôleur[6]. La seule façon d'intégrer la vitesse du robot est de alors de l'ajouter dans les prémisses de certaines règles portant sur l'évitement d'obstacles. Il s'agit en fait des seules règles portant sur la détection d'obstacles dans les zones d'intérêt à l'avant et à l'arrière du véhicule i.e FL, FA, FR et RA (figure III.11). En effet, la vitesse du véhicule ne doit pas, à nos yeux, avoir d'incidence sur les sous-comportements associés aux zones latérales RL, SL, SR et RR.

2.5. Cas d'échec et appel du planificateur

Le contrôleur d'exécution de mouvements est dans une situation d'échec lorsque l'écart entre la configuration courante q_{cour} et la configuration de référence q_{ref} devient trop important. Il s'agit là d'un écart métrique. Dans ce cas, la trajectoire de référence est trop distante pour avoir une quelconque utilité. Cet écart peut être dû à l'évitement d'obstacles non pris en compte par le planificateur ou dont le comportement a changé depuis la génération de la trajectoire de référence.

Dans ces cas de figure, le véhicule s'immobilise et fait appel au planificateur dans l'attente d'une nouvelle trajectoire de référence.

2.6. Extension 1 : coopération passive entre véhicules

Nous considérons que les véhicules ne communiquent pas entre eux, soit directement, soit par un quelconque médium. En effet, d'un point de vue pratique, de telles communications seraient assez lourdes à mettre en place. Cependant, les véhicules peuvent assurer des mouvements d'évitement qui, du point de vue d'un observateur extérieur, apparaissent comme une coopération. Une tentative d'évitement comme celle de la figure III.13 peut conduire à une situation de blocage, chaque véhicule essayant de passer dans le couloir libre.

Afin d'éviter ce genre de blocage, nous avons opté pour un comportement d'évitement d'obstacle dynamique par la gauche. Ce comportement permet de traiter convenablement la plupart des situations d'évitement pouvant survenir, comme le montre la figure III.14. On obtient ainsi des comportements analogues à ceux présentés par Zeghal ([75]) dans le cadre de la navigation aérienne : ce dernier, à partir du concept de force de glissements injecté dans une méthode à base de potentiels fictifs, assure également une coopération dans l'évitement de collisions entre avions sans aucun échange d'informations.

6. Chaque niveau de discrétisation d'une zone d'intérêt est une instanciation floue de la variable floue associée à cette zone.

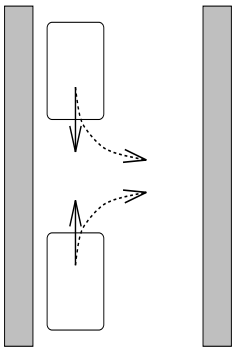


FIG. III.13 –: *Situation d'échec sans coopération entre véhicules*

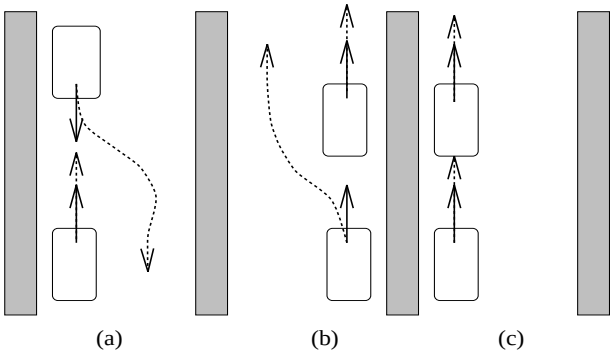


FIG. III.14 –: *Situations d'évitement intégrant une coopération passive*

Il est important de constater que dans la situation représentée par la figure III.15, le véhicule de derrière dépassera le véhicule arrêté par la droite car ce dernier, comme cela a été précisé précédemment, est considéré comme un obstacle statique. On notera ici la différence de comportement par rapport à la situation de la figure III.14 (c), où le véhicule détecté est en mouvement.

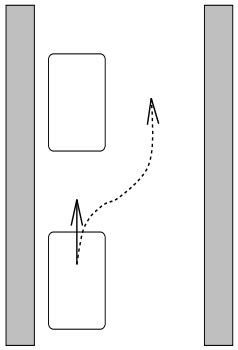


FIG. III.15 –: *Evitement d'un véhicule arrêté*

2.7. Extension 2 : environnement structuré

Si l'on considère un scénario tel que celui apparaissant dans la figure III.16, il est très possible que celui-ci conduise à une situation de blocage au centre du carrefour. En effet, le comportement de coopération passive ne suffira certainement pas dans ce cas particulier, en raison de l'espace relativement contraint du carrefour.

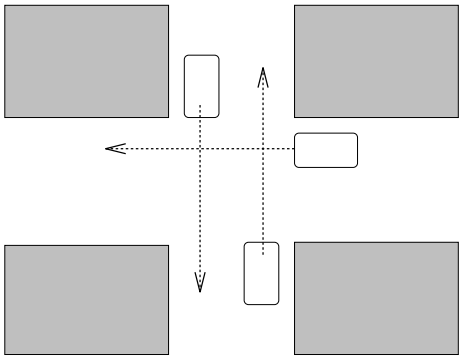


FIG. III.16 –: *Scénario pouvant amener à une situation de blocage en environnement structuré*

Nous avons donc étendu notre base de connaissance à la prise en compte de tels contextes. En fonction du contexte, qui pourra être délivré par une balise[7], on

7. Cette détermination des contextes par balise a été utilisée dans le projet Eureka PROMETHEUS, sur le démonstrateur français Prolab (programme ProArt).

activera les règles associées. Une illustration de ceci peut s'appliquer au contexte *carrefour* exposé dans la figure III.16, dans lequel nous avons introduit un comportement de priorité à droite. Cela nous a conduit à une traversée du carrefour illustrée par la figure III.17.

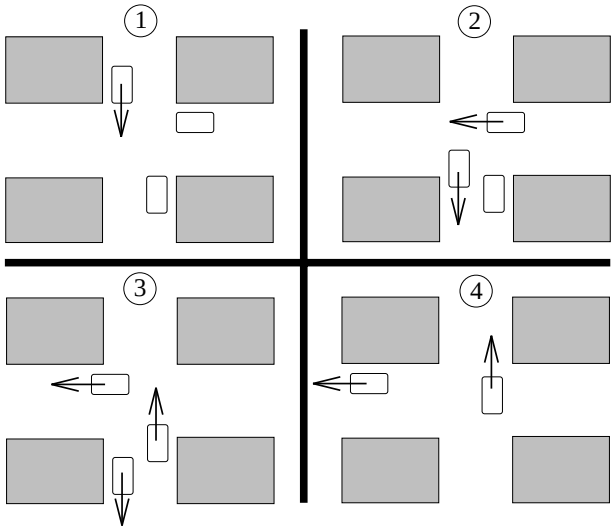


FIG. III.17 –: *Prise en compte du contexte pour la traversée d'un carrefour*

2.8. Problème d'oscillations

Le problème majeur auquel nous avons été confronté concerne les oscillations au niveau des trajectoires effectives des véhicules lorsqu'ils se trouvent dans des environnements relativement contraints. Ceci s'explique essentiellement par le fait qu'il est difficile, avec des actions réflexes, d'avoir un système qui réagisse rapidement aux événements extérieurs et qui ait un comportement global lissé, à partir d'un ensemble de comportements de base. De tels problèmes d'oscillations se produisent par exemple dans une situation comme celle de la figure III.18.

Dans l'exemple correspondant à la figure III.18, le véhicule est commandé par trois comportements activés simultanément, à savoir : le suivi de la trajectoire, l'évitement d'obstacle statique (bord de la route) et l'évitement d'obstacle dynamique (véhicule dépassé). La relative étroitesse du passage libre fait que le véhicule détecte des obstacles très proches, qui ont pour effet d'amorcer une manœuvre d'évitement, avant de "rebondir" sur l'autre obstacle, et ainsi de suite jusqu'à la fin de la manœuvre. Il est à noter également que ce comportement d'oscillations est favorisé par une profondeur des zones d'intérêt réduites (10 mètres pour des vitesses du véhicule de l'ordre de 15 km/h). En effet, nous avons constaté une nette diminution du phénomène en augmentant cette profondeur. Les obstacles, détectés plus tôt, amènent le véhicule à effectuer des trajectoires d'évitement plus lissées. Cependant, cette modification réalisée en simulation n'est pas

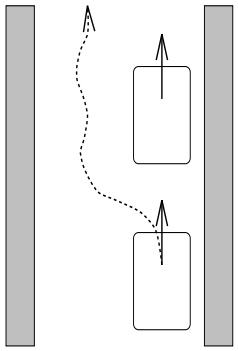


FIG. III.18 –: *Oscillations de la trajectoire effective d'un véhicule en environnement relativement contraint*

acceptable car, sur le véhicule réel, la portée des capteurs utilisés n'excède pas 10 mètres.

Solutions possibles

Plusieurs solutions viennent à l'esprit pour résoudre le problème précédemment exposé. Nous allons les passer en revue en essayant de préciser l'apport qu'elles peuvent fournir ou les lacunes qu'elles peuvent présenter.

a) Raffinement du contrôleur flou

Une première solution intuitive semble résider dans un raffinement du contrôleur flou. Pour cela, on peut diviser le domaine de variation des variables en un plus grand nombre de sous-ensembles flous et ajouter des règles permettant d'introduire les actions intermédiaires à celles du système initial. Nous avons cependant constaté que le véhicule ainsi contrôlé se trouve souffrir d'une "inertie" accrue le rendant beaucoup moins réactif. En d'autres termes, le véhicule réagit moins rapidement aux situations de danger qui demandent des réactions rapides, étant donné la vitesse des véhicules (15 km/h) et des distances de sécurité restreintes (2 à 3 mètres). L'explication est la suivante : pour une même situation, le fait d'introduire de nouvelles règles intermédiaires va atténuer l'effet de celles prônant des réactions d'urgence du robot (braquage et/ou freinage maximum par exemple). Ceci est dû au mécanisme de prise en compte des différentes règles activées (défuzzification) qui tend à moyenner le comportement de l'ensemble des règles. Une telle approche, pour donner de bons résultats, nécessiterait une validation dans un environnement peu dynamique (vitesse des véhicules moindre) et/ou avec des profondeurs de cartes de proximité plus importantes (meilleure détection de l'environnement local) permettant des réactions moins brusques de la part du robot.

b) Ordonnanceur de situation

Une autre solution, issue notamment de [3], consiste à introduire la notion de routeur d'un ensemble de modules de mouvements, chacun étant adapté à un comportement particulier (figure III.19). Le principe de base est le suivant :

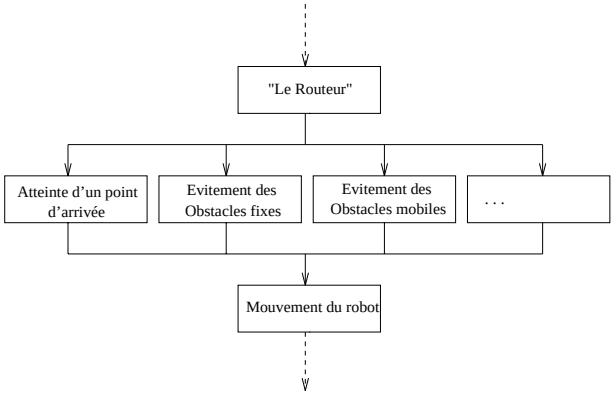


FIG. III.19 –: *Notion de routeur de comportements (Extrait de [3])*

en fonction de la situation dans laquelle se trouve le robot (informations de distance à l'instant t, variation des informations de distance entre les instants t et $t-1$)[8], le routeur décide d'activer un comportement. Cette décision provient de l'interprétation des résultats fournis par une étape de perception et d'analyse de l'espace libre du véhicule. A première vue, cette approche présente l'avantage de travailler sur une base distribuée de règles plus spécialisées qu'une base unifiée, donc plus facile à régler.

Cependant, le travail du routeur ne semble pas aisé dans la détermination d'un comportement à activer lorsque la situation courante nécessite la prise en compte simultanée de plusieurs comportements. Cela est le cas par exemple dans la situation représentée par la figure III.18. En effet, dans un tel cas de figure, quel comportement activer parmi l'évitement d'obstacles statiques, l'évitement d'obstacles dynamiques ou le suivi de trajectoire?

c) Modélisation plus complète de l'environnement local

Comme nous l'avons précisé précédemment, les obstacles statiques et dynamiques sont perçus à travers leur présence dans une zone d'intérêt avec la seule information de profondeur. Des informations plus riches, comme la géométrie et la configuration relative des obstacles, permettrait de raffiner les comportements du contrôleur. Par exemple, dans la situation de la figure III.18, le véhicule, lorsqu'il se trouve entre le bord de la voie et le véhicule dépassé, ne devrait pas osciller.

8. Il s'agit là d'informations issues de capteurs ultrasons.

En effet, il privilégierait une trajectoire rectiligne issue de la prise en compte de la géométrie et de la configuration relative de la voie et du véhicule dépassé.

Le problème de cette approche réside dans la génération des informations perceptives précédentes mettant en œuvre des systèmes de perception embarqués relativement complexes ou des communications inter-véhicules. De plus, la fusion des informations issues des différents capteurs doit considérablement ralentir l'activation du contrôleur pour utiliser ces informations.

d) Navigation locale

Une autre solution consiste à insérer dans notre architecture de contrôle un niveau de navigation locale entre les niveaux de planification et de contrôle d'exécution (figure III.20)[24, 25]. Le rôle de ce niveau est de fournir périodiquement une trajectoire locale au contrôleur d'exécution. Le principe de fonctionnement

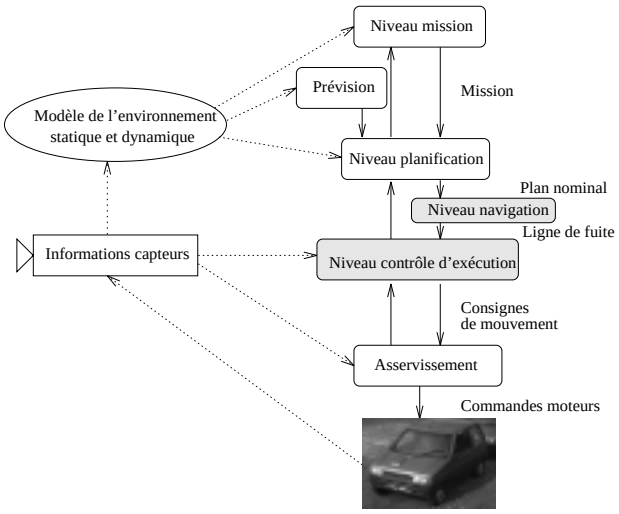


FIG. III.20 –: *Ajout d'un niveau navigation dans notre architecture de contrôle*

du niveau navigation s'appuie sur la notion de lignes de fuite [55]. A chaque période d'activation du contrôleur, une courbe de rattrapage est déterminée dans le but de rallier la position courante du véhicule à celle désirée. Ensuite, en fonction de l'état du véhicule, un certain nombre de trajectoires correspondant chacune à une loi de commande sont générées. Cet ensemble de trajectoires est projeté sur une carte de l'environnement local (figure III.21). La trajectoire de l'espace libre la plus proche de la courbe de rattrapage est envoyée au contrôleur.

L'avantage de cette méthode est de faciliter grandement la tâche du contrôleur d'exécution en lui fournissant régulièrement une trajectoire locale adaptée à la situation courante. Par exemple, dans la situation de la figure III.18, le véhicule ne prendra plus en compte la trajectoire de référence fournie par la planification mais une trajectoire d'évitement fournie par le navigateur local. Ainsi, le

contrôleur d'exécution se ramène essentiellement à un suivi de trajectoire avec des comportements d'arrêt d'urgence indispensables pour un système robuste.

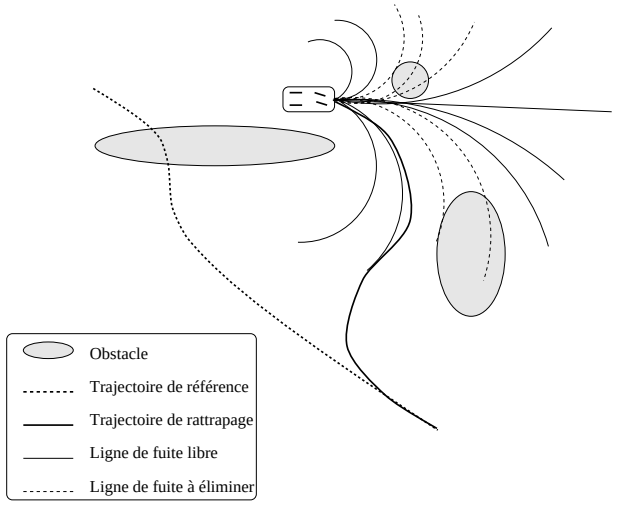


FIG. III.21 –: *Lignes de fuite et courbe de rattrapage (tiré de [55])*

e) Apprentissage supervisé

Une dernière solution peut consister en un apprentissage de la base de règles existante à travers le réglage de paramètres intervenant dans le système flou. Les règles floues peuvent être vues comme une connaissance a priori sur le contrôleur, connaissance affinée par apprentissage. Nous avons travaillé sur un apprentissage paramétrique supervisé des poids associés à chaque règle (nous avons indiqué précédemment ce que nous apportait la notion de poids de règles). Dans un premier temps, nous avons effectué ce réglage de manière empirique mais cela est rapidement apparu fastidieux et restrictif; en effet, des réglages affinés pour une situation donnée peuvent apparaître médiocres pour une autre situation. L'apprentissage supervisé nous permet de présenter un grand nombre d'échantillons issus de différents cas de figure, et la prise en compte de ces derniers permet de régler aisément notre système. Notre méthode d'apprentissage sera développée dans le chapitre § IV.

3. Conclusion

Nous avons présenté, dans ce chapitre, notre contrôleur d'exécution, de mouvements basé sur l'utilisation de la logique floue. Nous avons vu que, de manière expérimentale, celui-ci se comporte d'une manière analogue à une approche classique, en ce qui concerne le suivi de trajectoire. Deux avantages apparaissent

cependant, par rapport à une telle approche classique : la relative facilité d'écriture du contrôleur à travers des règles de type *système expert* et la prise en compte de l'environnement local, de manière unifiée par rapport au suivi de trajectoire. Cependant, un problème inhérent à l'approche floue persiste : celui concernant l'absence de preuve formelle de la stabilité d'un tel système. On ne dispose pas en effet de l'équivalent de fonctions de Lyapunov comme en automatique. De plus, une approche purement réactive, en espace relativement contraint, provoque des oscillations au niveau de la trajectoire effective du robot. Pour remédier à ce problème, nous avons exploré deux approches : la première consiste à insérer un niveau de navigation entre planification et contrôle d'exécution. Cette approche appauvrit sensiblement le contrôleur dans la mesure où celui-ci se ramène pratiquement à un suivi de trajectoire. La seconde revient à déterminer les poids des règles de la base de manière automatique à partir d'un ensemble d'échantillons (apprentissage supervisé). Cette approche permet de considérer les règles comme une connaissance a priori du contrôleur que l'on raffine à travers l'apprentissage.

Chapitre IV

Apprentissage d'une base de règles floues

Nous présentons dans ce chapitre notre méthode d'apprentissage paramétrique supervisé. Celui-ci nous permet de régler automatiquement notre contrôleur flou. L'apprentissage se fait sur la détermination de poids associés à chacune des règles de la base de connaissances, chaque poids représentant l'importance d'une règle par rapport aux autres.

1. Problématique

L'approche *logique floue* doit certainement son succès majeur à l'explication suivante : elle s'appuie sur une formulation naturelle, et donc floue, du mécanisme de contrôle (on décrit le processus d'une manière analogue à ce que ferait un expert humain). En fait, au lieu de modéliser le fonctionnement du processus en intégrant les actions possibles sur celui-ci (commandes), on s'attache uniquement à décrire la manière de contrôler ce même processus. Le processus est asservi par des règles du genre "**si** prémisse **alors** conclusion", définissant le contrôle non pas à un niveau numérique mais symbolique.
Malheureusement, le fait que cette approche présente en théorie un intérêt certain ne doit occulter les difficultés survenant en pratique quant à la détermination, voire le réglage, des règles et/ou des fonctions d'appartenance des sous-ensembles flous considérés [1]. Faute de méthodologie en ce qui concerne cette détermination, on doit avoir recours à un apprentissage structurel [2] pour les règles, et/ou paramétrique [2] pour les sous-ensembles flous. Notre méthode se situe au niveau de l'apprentissage paramétrique. Nous disposons d'une base de règles floues pondé-

1. Cette difficulté est accrue lorsque le nombre de règles composant la base est relativement importante.
2. Cette notion sera décrite dans la section suivante.

rées par des poids que l'on propose de déterminer de manière automatique par mise en œuvre de l'algorithme du simplexe [23, 22]. En effet, une détermination, comme nous l'avons faite dans un premier temps, s'avère relativement fastidieuse.

Tout d'abord, nous présentons les principales méthodes d'apprentissage paramétrique de règles floues. Afin d'essayer de corriger certaines faiblesses exposées, nous présentons notre approche basée sur la notion de programmation linéaire et d'utilisation de l'algorithme du simplexe. Nous montrons ensuite une validation de notre approche, à travers l'approximation d'une fonction analytique. Nous terminerons par la mise en œuvre de notre méthode d'apprentissage, dans le cadre de notre contrôleur d'exécution de mouvements.

2. Méthodes d'apprentissage

En ce qui concerne l'apprentissage paramétrique d'une base de règles floues, on distingue un certain nombre de méthodes qui sont essentiellement neuronales ou issues de techniques d'optimisation[3]. L'approche neuronale semble certes tout indiquée, lorsque l'on parle d'apprentissage. Cependant, l'inconvénient majeur que l'on peut formuler à l'encontre du neuronal réside dans le fait que l'on perd le niveau symbolique, qui fait l'attrait de la logique floue, au profit d'un traitement purement numérique : l'information (ou la connaissance) est répartie sur l'ensemble du réseau. De plus, l'usage maintenant fréquent de la rétropropagation du gradient[4] présente un certain nombre d'inconvénients [15] : la phase d'apprentissage reste coûteuse en temps de calcul, les problèmes inhérents à l'utilisation de gradient resurgissent suivant la forme de la fonction d'énergie à optimiser (minima locaux, "plateaux", "ravins", ...). De plus, même s'il existe des méthodes d'écriture de règles sous une forme neuronale [28], l'optimisation du réseau résultant (neurones, couches, connexions) ne semble pas évidente à déterminer.

L'appel à des techniques d'optimisation provient du fait que la notion d'apprentissage est intimement liée à celle d'optimisation d'un critère (souvent comparé à une énergie). Parmi ces méthodes, on distingue notamment, au niveau des travaux réalisés en logique floue : *la descente de gradient*, *les algorithmes génétiques*[5] [29] [39] et *le recuit simulé* [42]. L'approche *descente de gradient*, de par son aspect local, présente le gros inconvénient d'être trop sensible aux minima locaux. Par contre, la méthode du *recuit simulé*, elle, garantit normalement la détermination de l'optimum global. De plus, celle-ci n'est pas liée à une fonction coût particulière. Cependant, elle n'en est pas moins relativement lente et difficile

3. Cette séparation n'est pas stricte dans la mesure où le modèle de réseau de Hopfield s'appuie sur la notion d'optimisation d'une énergie sur laquelle repose la preuve de convergence de ce dernier.

4. Introduite par Le Cun(85) et Rumelhart(86).

5. Les algorithmes génétiques sont également utilisés pour l'apprentissage structurel à travers les systèmes de classificateurs (Classifier Systems).

à mettre en œuvre (cf. [42]) quant aux paramètres intervenant dans la méthode (température initiale, fonction de décroissance de la température, longueur des paliers, critère d'arrêt).

Afin de préserver les avantages (optimum global, fonction coût quelconque) d'une méthode comme *le recuit simulé*, tout en effaçant les inconvénients que nous venons d'énumérer, nous proposons une méthode basée sur l'algorithme du simplexe que nous allons présenter maintenant.

3. Notre approche : l'algorithme du simplexe

L'algorithme du simplexe de Dantzig [13] est certainement le plus efficace pour résoudre un programme linéaire. Une présentation détaillée de la méthode peut être trouvée dans [64]. Nous allons cependant insister sur les points les plus importants caractérisant la méthode.

3.1. Programme linéaire

Un programme linéaire est un problème d'optimisation dans lequel :

1. les variables du problème doivent satisfaire un ensemble d'équations et/ou d'inéquations constituant ainsi un système linéaire;

2. on doit optimiser (minimiser/maximiser) la valeur z d'une fonction linéaire appelée *fonction objective* ou *fonction coût*.

Il peut être facilement démontré que n'importe quel programme linéaire (P) peut être mis sous la forme suivante (forme standard) :

$$(P)\begin{cases} Ax = b, & x \geq 0 \\ c^T x = z(Max) \end{cases} \qquad \text{(IV.1)}$$

où A représente une matrice $m \times n$, b un m-vecteur, x et c deux n-vecteurs.

Le but est de déterminer x satisfaisant le système linéaire et maximisant[6] la fonction objective. Le simplexe détermine une solution (optimale si elle existe) par une succession de transformations (opérations de pivotage) de la matrice A visant à diminuer à chaque itération les composantes du vecteur c.

6. Dans la suite, nous ramènerons tout problème d'optimisation en un problème de maximisation en posant : Minimiser(x) $\simeq$ Maximiser(-x) et donc :

$$(P)\begin{cases} Ax = b, & x \geq 0 \\ c^T x = z(Min) \end{cases} \simeq (P')\begin{cases} Ax = b, & x \geq 0 \\ -c^T x = -z(Max) \end{cases}$$

3.2. Opérations de pivotage

Nous présentons ici la transformation matricielle, appelée "opération de pivotage". Une telle opération revient à multiplier la matrice A à gauche par une matrice particulière, dite *matrice de pivotage*. L'algorithme est décrit ci-dessous.

Algorithme PIVOTAGE (données : m, n, r, s; données modifiées : A)
/* A est une m×n-matrice ; $1 \leq$ r $\leq$ m; $1 \leq$ s $\leq$ n; $A_r^s \neq 0$ */
pour j = 1, ..., n faire
$A_r^j := A_r^j / A_r^s$
fin pour;
pour i = 1, ..., m; $i \neq r$ faire
pour j = 1, ..., n faire
$A_i^j := A_i^j - A_i^s A_r^j$
fin pour
fin pour
Fin algorithme

Le coefficient A_r^s est qualifié de pivot. Si on appelle $\hat{A}$ le résultat de l'opération de pivotage sur A, on constate que la $s^{ième}$ colonne de A a été transformée en vecteur unité. Autrement dit, $\hat{A}$ est définie de la manière suivante :

$$\hat{A}_i^j = \begin{cases} A_i^j & si \quad i \neq r, j \neq s \\ 0 & si \quad i \neq r, j = s \\ 1 & si \quad i = r, j = s \end{cases} \tag{IV.2}$$

Une telle opération revient donc à multiplier la matrice A par une $m \times m$-matrice D (matrice de pivotage) telle que $\hat{A} = DA$, définie comme :

$$D_i^j = \begin{cases} 1 & si \quad j = i, j \neq r \\ 1/A_r^s & si \quad i = j = r \\ -A_i^s/A_r^s & si \quad j = r, i \neq r \\ 0 & sinon \end{cases} \tag{IV.3}$$

En d'autres termes, la matrice D est la $m \times m$-matrice unité dont la $r^{ième}$ colonne a été remplacée par le vecteur :

$$D^r = \begin{pmatrix} -A_1^s/A_r^s \\ \vdots \\ -A_{r-1}^s/A_r^s \\ 1/A_r^s \\ -A_{r+1}^s/A_r^s \\ \vdots \\ -A_m^s/A_r^s \end{pmatrix} \tag{IV.4}$$

Le théorème IV.1 va nous permettre d'introduire l'algorithme du simplexe, qui consiste à effectuer une série d'opérations de pivotages sur la matrice de coefficients (A, b) de (P).

théorème IV.1 *Si on applique l'opération de pivotage à la matrice de coefficients (A, b) de (P), on obtient une matrice $(\hat{A}, \hat{b})$. Le système*

$$\hat{A}x = \hat{b}$$

est équivalent[7] *à (P).*

3.3. Algorithme du simplexe

Nous allons maintenant voir plus en détail en quoi consiste l'algorithme du simplexe. Nous présentons tout d'abord celui-ci avant de préciser certains points.

1. Initialisation :
   Le programme linéaire (P) est écrit sous forme canonique par rapport à une base réalisable J. On associe à J la solution de base correspondante.

2. Traitement :
   Itérer le processus suivant jusqu'à ce qu'on obtienne soit une base optimale soit un ensemble de solutions pour lesquelles z n'est pas borné :

   (a) Choisir s tel que $c^s > 0$
   si un tel s n'existe pas , la base J est optimale;

   (b) Examiner A^s
   si $A^s \leq 0$, pas de solution optimale i.e. z n'est pas borné;

   (c) Si $I = \{i | A^s_i > 0\} \neq \emptyset$, choisir un r tel que :
   $$b_r/A^s_r = \min_{i \in I}[b_i/A^s_i];$$

   (d) Effectuer une opération de pivotage sur la matrice des coefficients du programme linéaire, avec le pivot A^s_r;

   (e) $J = Jup\{x_s\} \setminus vecteur\_unitaire(r)$
   $vecteur\_unitaire(r) = x_s$.

Dans la phase d'initialisation, on suppose qu'il existe un ensemble de vecteurs colonnes $J \subset \{x_1, \ldots, x_n\}$ tel que A^J soit, à des permutations près, la matrice unité et le vecteur c^J nul. On dit alors que (P) est écrit sous forme canonique par rapport à la base réalisable J et x, la solution de base associée. Cette solution de base est déterminée de la manière suivante : les variables "hors base" sont nulles

7. Deux systèmes linéaires $Ax = b$ et $A'x = b'$ sont dit équivalents s'ils ont les mêmes solutions c'est-à-dire si : $\{x | Ax = b\} = \{x | A'x = b'\}$.

et les variables de la base sont déterminées par la résolution du système carré résultant[8]. Si un tel J n'existe pas, un moyen simple consiste à introduire de nouvelles variables, dites artificielles [64]. Dans la phase de traitement, la fonction *vecteur_unitaire* est définie comme :

$$\begin{cases} \{1,\ldots,m\} & \rightarrow & J \\ i & \mapsto & x \in J \ / \ i^{ième} \text{ composante de } x = 1 \end{cases}$$

L'utilisation de la notion de base réalisable et optimale dans l'algorithme du simplexe s'appuie sur le théorème suivant :

théorème IV.2 *(théorème fondamental de la programmation linéaire) Etant donné un programme linéaire :*

1. *s'il admet une solution réalisable, il admet une solution réalisable de base.*
2. *s'il admet une solution optimale, il admet une solution optimale de base.*
3. *s'il admet une solution réalisable et que la valeur de la fonction objective est bornée. supérieurement, il admet une solution optimale de base.*

Exemple : Considérons le programme linéaire suivant :

$$(P1)\begin{cases} 2x_1 + x_2 + x_3 = 8 & x_1, x_2, \ldots, x_5 \geq 0 \\ x_1 + 2x_2 + x_4 = 7 \\ x_2 + x_5 = 3 \\ 4x_1 + 5x_2 = z(Max) \end{cases}$$

(P1) est écrit sous forme canonique par rapport à la base réalisable $J_0 = \{x_3, x_4, x_5\}$. Nous allons maintenant détailler la résolution de (P1) en utilisant l'algorithme du simplexe. Dans les différentes étapes, le pivot sera encadré.

Etape 1 :

x_1	x_2	x_3	x_4	x_5	b
2	1	1	0	0	8
1	2	0	1	0	7
0	$\boxed{1}$	0	0	1	3
4	5	0	0	0	z

base réalisable : $J_0 = \{x_3, x_4, x_5\}$
solution de base associée : $x = (0, 0, 8, 7, 3)^T$
valeur de la fonction objective : 0
entrée dans la base : x_2
sortie de la base : x_5

8. Ceci implique que l'on ait une solution de base unique.

Etape 2 :

x_1	x_2	x_3	x_4	x_5	b
2	0	1	0	-1	5
$\boxed{1}$	0	0	1	-2	1
0	1	0	0	1	3
4	0	0	0	-5	$z-15$

base réalisable : $J_1 = \{3, 4, 2\}$
solution de base associée : $x = (0, 3, 5, 1, 0)^T$
valeur de la fonction objective : 15
entrée dans la base : x_1
sortie de la base : x_4

Etape 3 :

x_1	x_2	x_3	x_4	x_5	b
0	0	1	-2	$\boxed{3}$	3
1	0	0	1	-2	1
0	1	0	0	1	3
0	0	0	-4	3	$z-19$

base réalisable : $J_2 = \{3, 1, 2\}$
solution de base associée : $x = (1, 3, 3, 0, 0)^T$
valeur de la fonction objective : 19
entrée dans la base : x_5
sortie de la base : x_3

Etape 4 :

x_1	x_2	x_3	x_4	x_5	b
0	0	$1/3$	$-2/3$	1	1
1	0	$2/3$	$-1/3$	0	3
0	1	$-1/3$	$2/3$	0	2
0	0	-1	-2	0	$z-22$

base réalisable : $J_3 = \{5, 1, 2\}$
solution de base associée : $x = (3, 2, 0, 0, 1)^T$
valeur de la fonction objective : 22

Toutes les composantes du vecteur coût c (coefficients de la fonction objective) sont négatives ou nulles : la base réalisable est donc optimale et la solution de base associée est dite *solution de base optimale.*

3.4. Caractéristiques de l'algorithme

Complexité

Grossièrement, avec un programme linéaire comprenant m contraintes et n variables, l'algorithme du simplexe est capable de trouver, en moyenne, une solution en $3m/2$ opérations de pivotage. Pour cela, nous supposons que n varie proportionnellement à m[9]. Pour m fixé, le nombre d'opérations de pivotage est $log(n)$. Comme le dit Sakarovitch [64], il s'agit là d'une **évaluation moyenne**. En effet, un programme linéaire "d'école" comme :

$$\begin{cases} 2\sum_{j=1}^{i-1} 10^{i-j} x_j & \leq & 100^{i-1} \quad i = 1, c, dots, n \\ x_j & \geq & 0 \\ max(\sum_{j=1}^{n} 10^{n-j} x_j) & & \end{cases} \tag{IV.5}$$

nécessite 2^{n-1} itérations. Cependant, un algorithme comme celui proposé par Khachian (méthode des ellipsoïdes) semble moins performant que le simplexe en moyenne.

Complétude

Comme nous l'avons dit précédemment, si on associe la fonction objective à une fonction d'énergie à optimiser, la méthode du simplexe est complète dans la détermination de l'optimum de cette fonction d'énergie. En d'autres termes, on ne tombe jamais dans un minimum local.

3.5. Conventions

Dans la suite, **E** représente l'**ensemble des échantillons** intervenant dans l'apprentissage, **R** l'**ensemble des règles**. Les notations **#E et #R** correspondent aux **cardinalités** respectives de ces ensembles. Nous considérons des règles simples i.e avec seulement une variable en partie *conclusion*. Les échantillons présentés au système auront donc l'aspect suivant :

$$E_j = (e_{j1}, e_{j2}, \ \ldots \ , e_{jm}, s_j)$$

si nous avons m entrées et une seule sortie. Le dernier point (une sortie pour le système) n'est pas une restriction mais rend plus aisée la présentation de la méthode.

Les règles floues que nous utilisons sont du type :

si *prémisse* **alors** *conclusion* ,

où *prémisse* est une proposition floue générale composée de connecteurs (**et**, **ou**

9. Cela sera le cas dans nos programmes linéaires comme nous le verrons ultérieurement.

et **non**) ainsi que de modificateurs linguistiques [73] du genre (**très**, **peu**, ...) et dont l'expression de base est la proposition floue élémentaire (v est A) qualifiant la possibilité que la variable v appartienne à la caractérisation floue A de la variable floue associée. Ces règles sont donc simples i.e. la *conclusion* ne porte que sur une variable et ne sera constituée que d'une proposition floue élémentaire. La méthode de défuzzification est la suivante : pour chaque règle floue R_i de la base, le niveau d'activation $\mu_{prémisse}$ issu de la prémisse détermine une aire A_i et un centre de gravité[10] COA_i pour la caractérisation floue A apparaissant dans la conclusion. La détermination des A_i et COA_i se fait de la manière suivante : A_i est l'intégrale (aire) de la fonction $min(\mu_{prémisse}, \mu_A(v)), \forall v \in U$ où $\mu_A(v)$ représente le degré d'appartenance de la variable de sortie v à A et U l'univers de discours. COA_i est le centre de gravité (en fait la projection de celui-ci dans U) de cette aire[11]. Pour une variable de sortie donnée, la valeur résultant de la défuzzification sera donnée par la formule :

$$COA = \frac{\sum_{i=1}^{\#R} COA_i \times A_i \times w_i}{\sum_{i=1}^{\#R} A_i}, w_i \in I\!R \qquad \text{(IV.6)}$$

où les coefficients w_i correspondent aux poids associés à chaque règle. En ce qui concerne la notion de poids, une approche similaire apparaît dans [61] avec des poids normalisés (i.e. sur l'intervalle [0,1]). Cependant, la différence essentielle réside dans le fait que, dans [61], les poids concernent le niveau d'activation $\mu_{prémisse}$ issu des prémisses de chaque règle, tandis que dans notre approche, les poids n'agissent qu'au niveau de la défuzzification. En d'autres termes, dans la première approche, le poids peut modifier à la fois centre de gravité et aire des sous-ensembles flous des conclusions de règles, ce qui n'est pas le cas dans notre approche. Dans les deux approches, la notion de poids nous permet de disposer d'un paramètre permettant de régler le système flou. Cependant, un avantage de notre méthode semble résider dans un plus grand domaine de variation des poids, à savoir $I\!R$, au lieu du seul intervalle [0,1].

3.6. Algorithme d'apprentissage et programme linéaire résultant

Nous rappelons que le but de notre méthode d'apprentissage est de déterminer les poids w_i associés à chaque règle de la base. Ces poids seront calculés de manière à minimiser, pour chaque échantillon fourni, l'erreur entre les valeurs de sorties désirées du contrôleur et les valeurs effectivement délivrées par celui-ci.

10. La notion de centre de gravité a été définie en § II.3.4., page 37.

11. Il est à noter que si la fonction d'appartenance associée à A est symétrique, la valeur de COA_i est invariante.

L'algorithme d'apprentissage peut être vu comme suit :

1 pour chaque échantillon $E_j = (e_{j1}, e_{j2}, \ \ldots \ , e_{jm}, s_j)$ de E faire
pour chaque règle R_i dans R faire
déterminer COA_{ij} et A_{ij}

2 déterminer $w \in I\!R^{\#R}$ minimisant $\|\delta\|(\delta \in I\!R^{\#E})$ où : $\forall j, 1 \leq j \leq \#E$,[12]

$$\underbrace{s_j}_{\text{valeur désirée}} + \delta_j = \underbrace{\frac{\sum_{i=1}^{\#R} COA_{ij} \times A_{ij} \times w_i}{\sum_{i=1}^{\#R} A_{ij}}}_{\text{valeur calculée}}$$

étape 1 : pour chaque échantillon E_j, on détermine le centre de gravité COA_{ij} et l'aire A_{ij}, pour des poids de règle unitaires ($\forall i, 1 \leq i \leq \#R, w_i = 1$).
étape 2 : l'étape 1 appliquée à l'ensemble des échantillons donne un système linéaire où chaque contrainte est associée à un échantillon. Il s'agit de déterminer alors le vecteur poids $w \in I\!R^{\#R}$ minimisant la norme d'un vecteur δ. Pour chaque échantillon E_j, la composante δ_j représente l'erreur entre la valeur désirée s_j et la valeur calculée (partie droite de l'équation).

Compte tenu de ce que nous venons de voir, le programme linéaire résultant de l'algorithme décrit précédemment s'écrira sous la forme :

pour chaque échantillon E_j	$s_j + (\delta'_j - \delta''_j)$	$= \frac{\sum_{i=1}^{\#R} COA_{ij} \times A_{ij} \times (w'_i - w''_i)}{\sum_{i=1}^{\#R} A_{ij}}$
	δ'_j, δ''_j	$\leq \Delta$
fonction objective	$min\left((\sum_{i=1}^{\#E}(\delta'_i + \delta''_i)) + (\#E) \times \Delta\right)$	

ou

12. Nous rappelons que les notations #R et #E représentent respectivement le nombre de règles composant la base de connaissances de notre contrôleur et le nombre d'échantillons utilisés dans la phase d'apprentissage

pour chaque échantillon E_j	$s_j + (\delta'_j - \delta''_j)$	$= \frac{\sum_{i=1}^{\#R} COA_{ij} \times A_{ij} \times (w'_i - w''_i)}{\sum_{i=1}^{\#R} A_{ij}}$
	δ'_j, δ''_j	$\leq \Delta$
fonction objective	$max\left(-(\sum_{i=1}^{\#E}(\delta'_i + \delta''_i)) - (\#E) \times \Delta\right)$	

si l'on veut se ramener à un problème de maximisation.
La décomposition des w_i et δ_i respectivement en $(w'_i - w''_i)$ et $(\delta'_i - \delta''_i)$ est un artifice permettant d'exprimer toute variable réelle signée à partir de variables positives (les seules prises en compte par le simplexe). La norme retenue est définie comme la somme de deux normes :

$$\|\delta\|_1 = \sum_{i=1}^{\#E} |\delta_i| \qquad et \qquad \|\delta\|_2 = (\#E) \times \underbrace{\max_{i \in E}\{|\delta_i|\}}_{\Delta}$$

La minimisation de la première norme permet de satisfaire complètement le plus grand nombre d'échantillons possibles (avec composante δ_i nulle) au détriment d'autres échantillons, alors que la minimisation de la seconde permet de mieux répartir l'erreur sur l'ensemble des échantillons. La prise en compte des deux normes autorise un compromis entre ces deux comportements.
Le programme linéaire aura donc $2 \times (\#E + \#R)$ variables et $3 \times \#E + 1$ contraintes.
Nous avons choisi $\mathbb{R}$ comme domaine de définition des poids. D'un point de vue pratique, cela nous permet bien évidemment d'accroître le domaine des solutions réalisables pour notre système linéaire, et par conséquent d'améliorer peut-être la solution optimale. D'un point de vue sémantique, la notion de force d'une règle sera représentée par la valeur absolue du poids de cette règle.

3.7. Evolution de la base de règles

D'un point de vue pratique, il est évident que, dans la mesure où nous travaillons sur un système linéaire, les résultats de l'apprentissage seront d'autant meilleurs que le nombre de règles et d'échantillons seront du même ordre de grandeur. Cette considération nous pousse à lancer l'apprentissage sur une base aussi exhaustive que possible. A l'issue de la phase d'apprentissage, toutes les règles avec un poids négligeable seront retirées de la base. Cette méthode a été utilisée notamment par Sugeno au niveau d'un système flou, Halgamuge et Glesner [30] ainsi que Nauck et Kruse [54] dans des réseaux neuro-flous.

3.8. Un exemple : approximation de $f(x, y) = x^2 + y^2$

Afin de tester cette méthode le plus simplement possible, nous avons décidé d'approximer la fonction analytique $f(x, y) = x^2 + y^2$ sur le domaine $[-1, 1] \times [-1, 1]$ (cet exemple est emprunté à [28]). L'avantage de cet exemple réside dans la facilité de génération d'une base d'apprentissage, ce qui n'est pas toujours le cas pour une application quelconque. De plus, il est facile de vérifier visuellement les résultats obtenus.
Nous disposons d'une base de 25 règles avec 5 sous-ensembles flous (NG, NM, ZR, PM, PG)[13] pour les entrées x et y et 3 sous-ensembles flous (P, M, G)[14] pour la sortie.
La figure IV.1 illustre l'évolution de l'apprentissage supervisé pour un nombre croissant d'échantillons. La figure IV.1.1 correspond à la sortie du système flou

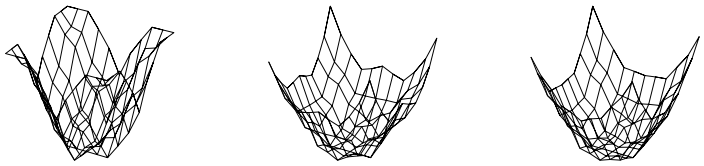


FIG. IV.1 –: *Approximation de $f(x, y) = x^2+y^2$ (1) avant apprentissage, (2) après apprentissage sur 120 échantillons, (3) après apprentissage sur 440 échantillons*

avant l'apprentissage; les figures IV.1.2 et IV.1.3 correspondent à la sortie du système pour des apprentissages réalisés respectivement avec 120 et 440 échantillons.

4. Apprentissage du contrôleur d'exécution de mouvements

En ce qui concerne l'apprentissage de notre contrôleur d'exécution de mouvements, un problème s'est posé concernant la collecte des échantillons constituant la base d'apprentissage supervisé, et ce, en simulation. En effet, pour générer une base d'exemples, il s'avère nécessaire de disposer d'un moyen permettant à un utilisateur de commander le véhicule. Malheureusement, cette fonctionnalité n'est pas disponible sur le simulateur que nous avons utilisé[15] et qui sera présenté

13. Respectivement Négatif Grand, Négatif Moyen, ZéRo, Positif Moyen et Positif Grand.
14. Respectivement Petit, Moyen et Grand.
15. Ce problème n'existera pas dans les expérimentations réelles que nous avons réaliserons ultérieurement, le véhicule utilisé permettant aussi bien une conduite manuelle qu'automatique.

en § V. Pour générer les échantillons nécessaires à la phase d'apprentissage, nous avons donc choisi d'utiliser notre contrôleur d'exécution pour lequel les poids de règles avaient été déterminés de manière empirique.

4.1. Phase d'apprentissage

Cette phase concerne donc la collecte des échantillons nécessaires à l'apprentissage et la détermination hors-ligne des poids de règles en lançant notre algorithme d'apprentissage sur la base d'échantillons. La récupération des échantillons se fait à partir de plusieurs scenarii, au sein d'une même base, aussi exhaustive que possible. Pour l'ensemble de nos expérimentations, nous avons utilisé un ensemble de sous-bases issues de cette base initiale à travers une fonction de tirage aléatoire d'échantillons. Il s'agit de la fonction :

fonction random_select (**entier** n; **fichier** *base_init*; **fichier** *base*)

où n représente le nombre d'échantillons désirés, *base_init* la base exhaustive initiale et *base*, la base résultant du tirage.

L'apprentissage proprement dit est réalisé en lançant notre algorithme d'apprentissage sur la base d'échantillons.

4.2. Phase de généralisation

La phase de généralisation consiste à utiliser le contrôleur d'exécution de mouvements, après apprentissage sur des scenarii n'ayant pas été utilisés dans la phase d'apprentissage[16]. Il s'agit là d'une approche classique permettant de mettre en évidence les capacités de généralisation du contrôleur après apprentissage. En effet, de bons résultats obtenus sur les mêmes scenarii ne pourraient traduire qu'un apprentissage "par cœur", ce qui n'est pas le but recherché.

Il convient avant tout de préciser que notre apprentissage n'a pas pour but de créer des comportements nouveaux, mais de régler l'importance relative de chacun par rapport aux autres. Les comportements sont déjà codés à travers les règles floues constituant le contrôleur et représentant en quelque sorte une connaissance a priori. Cependant, ces comportements sont codés indépendamment les uns des autres, comme s'ils étaient les seuls à constituer la base. Par conséquent, si l'activation d'un seul comportement de base donne rapidement ce qui est attendu de la part du contrôleur (e.g. suivi de trajectoire), il peut en être tout autrement lorsque plusieurs comportements sont activés simultanément (e.g. suivi de trajectoire avec évitement d'obstacles).

Comme nous l'avions fait pour comparer notre contrôleur flou avec une approche classique en ce qui concerne le suivi de trajectoire, nous allons présenter un certain nombre de résultats nous permettant de comparer notre contrôleur

16. Ces scenarii sont relativement voisins de ceux ayant été utilisés dans la phase d'apprentissage. En effet, notre système généralise par interpolation et non par extrapolation.

après apprentissage automatique, avec notre contrôleur réglé de manière empirique mais aussi avec notre contrôleur doté de poids unitaires (i.e. sans aucun réglage des poids). Pour cela nous allons considérer le scénario suivant : un véhicule contrôlé doit suivre une trajectoire rectiligne sur laquelle se trouve un obstacle, amenant ainsi le véhicule à effectuer une manœuvre d'évitement. Pour les différents contrôleurs présentés, nous ferons apparaître la trajectoire suivie par le véhicule contrôlé ainsi que les variations de l'erreur en distance et orientation par rapport à la trajectoire de référence.

Contrôleur avec poids unitaires

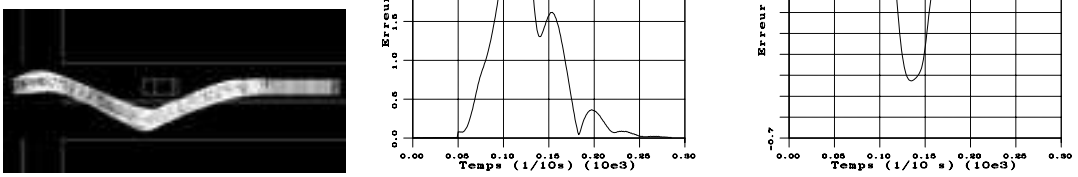


FIG. IV.2 –: *Evitement d'obstacle avec poids de règles unitaires*

Au niveau de la trajectoire effectuée (figure IV.2), on constate que celle-ci est relativement proche de l'obstacle. La manœuvre d'évitement est amorcée un peu trop tardivement. Le véhicule contrôlé n'utilise de ce fait pas au mieux l'espace libre constitué par la voie de dépassement et les distances de sécurité avec les obstacles ne sont pas respectées. Cela s'explique par le fait que le comportement d'évitement d'obstacles n'est pas assez privilégié par rapport à celui assurant le suivi de trajectoire.

Contrôleur avec poids déterminés de manière empirique

Ici, la trajectoire effectuée (figure IV.3) caractérise une meilleure anticipation au niveau de la manœuvre d'évitement. Ainsi, le véhicule utilise au mieux la voie de dépassement (contrairement au cas précédent).

Contrôleur avec poids déterminés par apprentissage

Comme nous l'avons exposé précédemment, notre contrôleur a appris à partir d'exemples fournis par le même contrôleur (même base de connaissances) dont les poids avaient été déterminés de manière empirique. Il s'ensuit que la trajectoire du

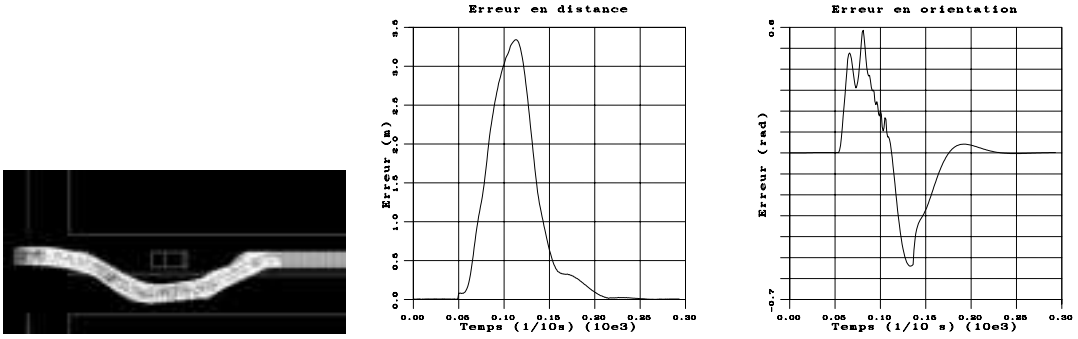


FIG. IV.3 –: *Evitement d'obstacle avec poids de règles déterminés de manière empirique*

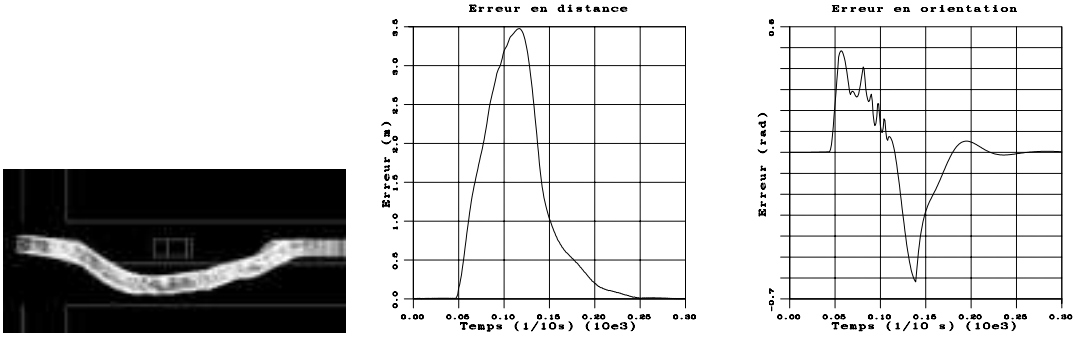


FIG. IV.4 –: *Evitement d'obstacle avec poids de règles déterminés par apprentissage*

véhicule contrôlé est relativement analogue à celle suivie dans le cas précédent. On peut cependant noter une anticipation accrue au niveau de la manœuvre d'évitement (figure IV.4). Celle-ci est due au fait que les poids associés aux règles concernées sont légèrement moindres que ceux déterminés de manière empirique.

4.3. Evolution de l'apprentissage

Nous nous sommes attachés à étudier la convergence de notre apprentissage quant à la détermination des poids de règles. Pour cela, nous avons soumis à notre algorithme d'apprentissage un certain nombre de bases d'exemples croissantes S_i. Le critère que nous avons retenu pour illustrer la convergence est le suivant :

$$c = \sum_j |w_j^{i+1} - w_j^i| \qquad \text{(IV.7)}$$

où w_j^i représente la valeur du poids associé à la règle j après apprentissage sur la base S_i. Ce critère a été choisi pour permettre une représentation plus aisée de l'évolution de l'apprentissage des poids, à travers la prise en compte d'une fonction d'évaluation globale[17]. La figure IV.5 illustre l'évolution du critère IV.7 pour des bases d'apprentissage allant jusqu'à 1700 exemples. La courbe représente l'ensemble des valeurs moyennes résultant de plusieurs expérimentations.

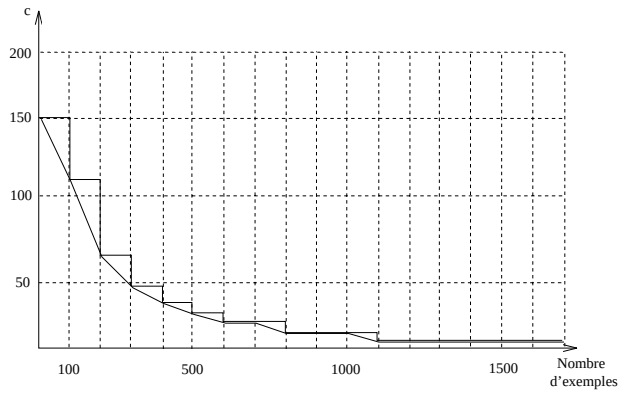


FIG. IV.5 –: *Convergence des poids de règles de la base de connaissances du contrôleur flou*

5. Conclusion

Nous avons décrit une approche permettant de réaliser un apprentissage paramétrique supervisé pour une base de règles floues. Celle-ci est basée sur l'utilisation de l'algorithme du simplexe. Cette méthode se veut générale dans la mesure

17. Le recours à un tel critère nous permet de nous affranchir de l'étude de la convergence propre à chaque poids de règle.

où elle ne dépend pas de l'application considérée à travers un paramétrage spécifique. De plus, elle n'est pas liée à un critère d'optimisation particulier. Contrairement à des méthodes locales (gradient), elle n'est pas sensible aux problèmes de minima locaux et se caractérise par une complexité proportionnelle au nombre d'échantillons présentés dans la phase d'apprentissage.

Cependant, une telle approche implique un apprentissage hors-ligne, ce qui enlève toute dynamicité à la base, lors de la phase de mise en œuvre de celle-ci. Cette considération nous a amené à considérer une méthode incrémentale (s'appuyant également sur le simplexe) permettant une exécution en temps réel. Cette méthode peut très bien s'articuler avec celle présentée dans la mesure où cette dernière constituerait une phase d'initialisation facilitant l'apprentissage de la méthode incrémentale. Les premiers résultats obtenus nous incitent à approfondir ces travaux.

Chapitre V

Simulation et expérimentations

Ce chapitre décrit l'implantation de notre contrôleur d'exécution de mouvements au sein de l'architecture de simulation que nous avons développée, ainsi que sur notre véhicule prototype dans le cadre du projet Praxitèle. Sont décrits notre plate-forme de simulation, le projet Praxitèle avec le concept de TUPI, les caractéristiques du véhicule expérimental et l'outil de spécification de la tâche robotique associée à notre architecture : Orccad. Des résultats issus de la phase de simulation sont également présentés.

1. Simulation

Afin de valider notre contrôleur d'exécution de mouvements, il s'est avéré nécessaire de disposer d'un simulateur d'environnement pour un robot de type *voiture*. Les principales fonctionnalités attendues de ce simulateur sont les suivantes :

- **une interface conviviale pour l'utilisateur**, tant au niveau de la mise en place d'un scénario (environnements statique et dynamique) qu'au niveau de la visualisation du déroulement de celui-ci;

- **la délivrance d'informations perceptives**, tant proprioceptives qu'extéroceptives, sur l'environnement dans lequel évolue le véhicule contrôlé;

- **la prise en compte de commandes** permettant de faire évoluer le véhicule contrôlé au sein de l'environnement;

- **un "encapsulage" du simulateur** permettant un interfaçage aisé avec tout type d'application, le simulateur constituant ainsi une véritable "boîte noire".

Nous allons préciser dans la suite les différentes caractéristiques du simulateur que nous avons développé dans l'environnement du logiciel de CAO-robotique ACT[1].

1.1. Architecture générale

L'architecture générale de notre environnement de simulation apparaît dans la figure V.1. Celle-ci est largement inspirée de PHAROS, simulateur de trafic routier, développé au CMU dans le cadre du projet de robotique mobile NAVLAB [63]. Dans cette architecture, le simulateur (partie droite) échange avec l'appli-

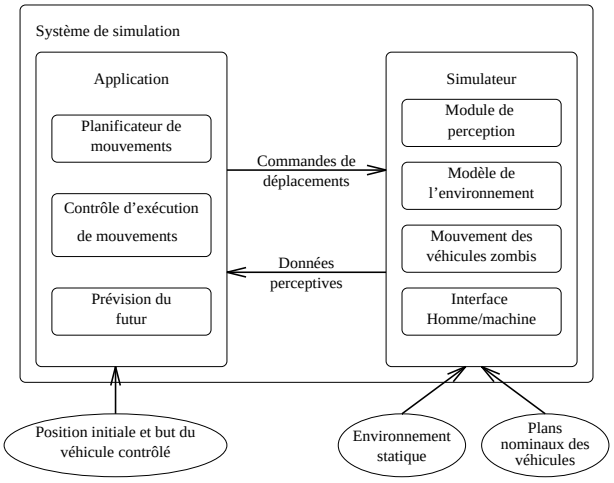


FIG. V.1 –: *Architecture de l'environnement de simulation*

cation (partie gauche) un certain nombre d'informations qui sont des données perceptives (proprioceptives et extéroceptives) sur l'environnement courant et des commandes (accélération et vitesse de braquage) pour le véhicule contrôlé. Il apparaît ainsi que seule la connaissance du protocole de communication (interface) entre les deux modules précédents est nécessaire pour qu'une application quelconque utilise le simulateur. Nous verrons par la suite que l'interface que nous avons retenue (via le réseau Internet) permet que le simulateur communique avec des applications implantées sur différents systèmes (Unix, VxWorks) et/ou sur différentes machines (Sparc Station, carte temps-réel).

1.2. Le logiciel de CAO-robotique ACT

Afin d'alléger notre travail au niveau de la modélisation de l'environnement ainsi qu'au niveau de l'interface homme-machine, notre simulateur utilise l'environnement du logiciel de CAO-robotique ACT [52, 53]. La décomposition d'ACT

1. © ALEPH Technologies.

en librairies écrites en langage de programmation C, fait de ce logiciel un système ouvert, dans lequel est prévu un point d'entrée pour une application utilisateur, notre simulateur en ce qui nous concerne.

Les principales fonctionnalités du logiciel ACT sont :

- **création et gestion de menus, de fenêtres et de boîtes de dialogue** permettant la saisie d'informations de la part de l'utilisateur;
- **modélisation $3D$ d'une cellule robotique** à partir de primitives volumiques élémentaires;
- **gestion de tâches robotiques** à travers leur définition et la génération automatique des transitions entres celles-ci;
- **représentation graphique $3D$ d'une cellule robotique**;
- **système ouvert**, grâce à un interfaçage aisé avec une application utilisateur, par l'intermédiaire d'un point d'entrée utilisateur.

1.3. Interface homme-machine

La définition de l'interface de saisie des données ainsi que la modélisations des environnements statiques et dynamiques sont détaillées dans [41].

La saisie des informations se fait par l'intermédiaire de menus déroulants et de boîtes de dialogue comme le montre la figure V.2. Une représentation graphique $3D$ d'un environnement routier apparaît dans les saisies d'écran de la figure V.3.

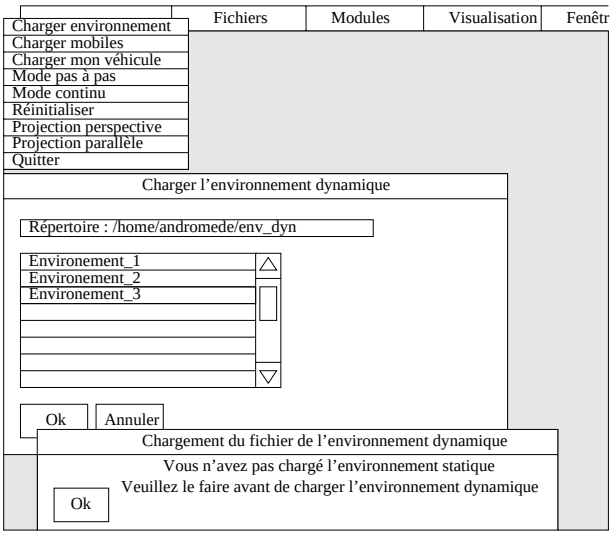


FIG. V.2 –: *Simulateur : saisie d'informations utilisateur*

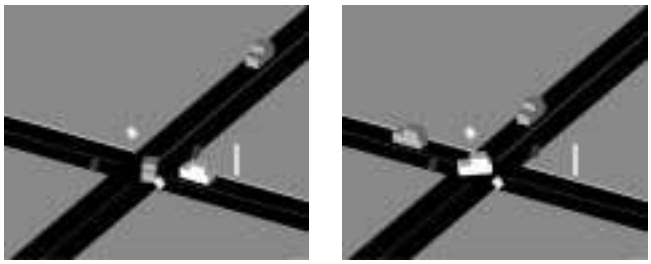


FIG. V.3 –: *Simulateur : visualisation d'un environnement routier*

1.4. Modélisation d'un véhicule de type *voiture*

Nous allons préciser ici le modèle associé à un véhicule et, par extension, à un robot mobile $\mathcal{M}$, de type *voiture*, tant au niveau géométrique qu'à celui des contraintes associées à un tel véhicule.

Modèle géométrique

L'environnement dans lequel nous travaillons peut être assimilé à une projection orthogonale de l'environnement de travail ($I\!R^3$) sur le sol d'évolution que nous considérons comme plan ($I\!R^2$). Ainsi, $\mathcal{M}$ sera modélisé, d'un point de vue géométrique, par un rectangle comme l'illustre la figure V.4 provenant de [32].

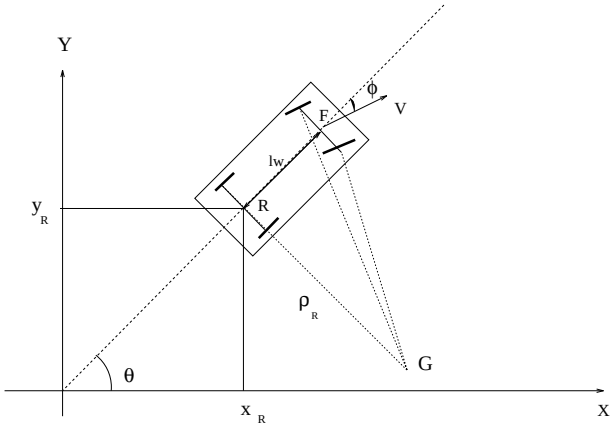


FIG. V.4 –: *Modèle géométrique d'un véhicule de type voiture*

Le point R, centre de l'essieu arrière, a été choisi comme point de référence de $\mathcal{M}$. Ainsi, à tout moment, $\mathcal{M}$ sera caractérisé de manière unique par le triplet de configuration (x_R, y_R, θ), défini sur $I\!R^2 \times S^1$, où (x_R, y_R) représentent les coordonnées de R dans un repère fixe $(0, \vec{\imath}, \vec{\jmath})$ associé à la scène et θ l'orientation de l'axe longitudinal de $\mathcal{M}$ dans ce même repère (cf. figure V.5).

Repère lié au véhicule

Par la suite, nous serons amené à nous référer à un *repère véhicule*. Celui-ci, noté $(R, \vec{u}, \vec{v})$, que nous avons choisi, a pour origine le point de référence R défini précédemment. Le vecteur $\vec{u}$ est porté par l'axe longitudinal de symétrie de $\mathcal{M}$ et dirigé vers l'avant de celui-ci; $\vec{v}$ est tel que le repère $(R, \vec{u}, \vec{v})$ soit direct (figure V.5).

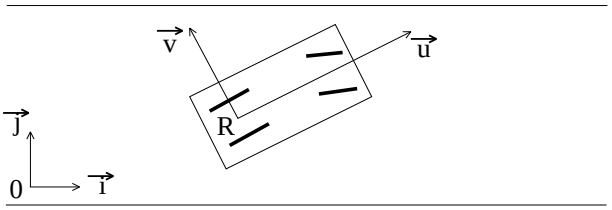


FIG. V.5 –: *Repère lié au véhicule*

Contraintes mécaniques

Les contraintes mécaniques de $\mathcal{M}$ sont exprimées à travers les butées de ϕ, angle matérialisant l'orientation du vecteur vitesse au centre de l'essieu avant, par rapport à l'axe longitudinal de $\mathcal{M}$ (cf. figure V.4).

$$-\phi_{max} \leq \phi \leq \phi_{max} \tag{V.1}$$

Ces butées de l'angle ϕ, appelé angle de braquage, entraînent l'existence d'un rayon de giration minimal ρ_{min_1} du point de référence R. Ce rayon est défini comme :

$$\rho_{min_1} = \frac{l_w}{\tan(\phi_{max})} \tag{V.2}$$

où l_w représente l'empattement de $\mathcal{M}$. Ce rayon de giration ne pourra être atteint que pour des vitesses relativement faibles de $\mathcal{M}$. En effet, à partir d'une certaine vitesse, $\mathcal{M}$ serait soumis à une force centrifuge telle que l'on aurait un glissement des roues par rapport au sol (vitesse des points de contact roues-sol non nulle) ce que nous ne traitons pas. Nous verrons un peu plus loin la contrainte induite sur le rayon de giration minimal par une contrainte de non glissement.

Contraintes cinématiques

– Vitesse bornée
  La première contrainte cinématique concerne le domaine des valeurs admissibles pour le véhicule. On peut écrire :

$$v_R \in [0, v_{Rmax}] \tag{V.3}$$

  où v_R représente la vitesse du point de référence R de $\mathcal{M}$.

– Contraintes de non-holonomie
Cette partie est tirée de [43] que le lecteur pourra consulter pour une analyse plus détaillée du problème soulevé ici.
Si nous écrivons les équations du mouvement associées au point de référence R de $\mathcal{M}$, exprimé dans le repère fixe $(0, \vec{\imath}, \vec{\jmath})$ associé à la scène, nous obtenons :

$$\dot{x_R} = v_R \cos\theta \tag{V.4}$$

$$\dot{y_R} = v_R \sin\theta \tag{V.5}$$

$$\dot{\theta} = \frac{v_R}{\rho} = v_R \frac{\tan(\phi)}{l_w} \tag{V.6}$$

où v_R représente la vitesse instantanée du point de référence R.

A partir des équations V.4 et V.5, nous pouvons écrire l'équation suivante :

$$\dot{x_R}\sin\theta - \dot{y_R}\cos\theta = 0 \tag{V.7}$$

qui constitue une **contrainte non-holonome d'égalité**. Une telle contrainte réduit l'espace des vitesses admissibles de $\mathcal{M}$ à un sous-espace de dimension 2 de l'espace tangent. Elle est due à la contrainte de non glissement des roues par rapport au sol.

Les équations V.2 et V.6 nous permettent d'écrire :

$$\dot{\theta} \leq \frac{v_R}{\rho_{min1}} \tag{V.8}$$

soit :

$$\dot{x_R}^2 + \dot{y_R}^2 - {\rho_{min1}}^2\dot{\theta}^2 \geq 0 \tag{V.9}$$

qui constitue une **contrainte non-holonome d'inégalité**. Cette contrainte exprime le fait que, pour toute configuration (x, y, θ) de $\mathcal{M}$, le vecteur vitesse $(\dot{x}, \dot{y}, \dot{\theta})$ se trouve dans un cône d'angle $2\arctan(1/\rho_{min1})$.

– Conséquences de la non-holonomie
La combinaison des deux contraintes précédentes impose que le vecteur vitesse ne peut se situer qu'à l'intérieur de deux secteurs comme l'illustre la figure V.6. D'une manière moins informelle, de telles contraintes limitent les déplacements instantanés d'un robot. L'exemple couramment donné est que, sous une contrainte de non glissement, un robot de type *voiture* ne peut se déplacer dans une direction parallèle à son essieu arrière. Cependant, ces contraintes ne limitent en rien l'espace des configurations atteignables par le robot [2]. Laumond [45] a démontré que, pour un robot de type *voiture*, tout

2. Malgré la remarque précédente, on sait très bien que l'on est capable de réaliser des créneaux avec une voiture.

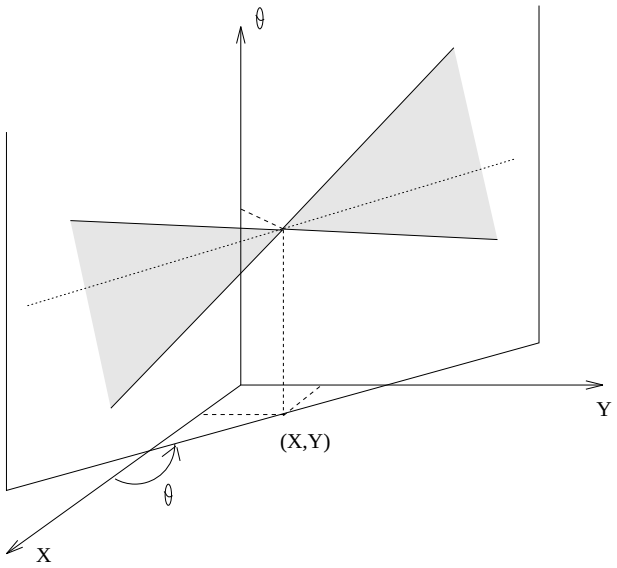


FIG. V.6 –: *Domaine des vitesses possibles résultant de la prise en compte des contraintes de non-holonomie [43]*

chemin libre (éventuellement non réalisable) entre deux configurations peut être transformé en un chemin réalisable au prix d'un nombre fini (mais non borné) de manœuvres. On dit qu'un tel robot est **complètement contrôlable**.

Contraintes dynamiques

Nous travaillons sur une modélisation dynamique très simplifiée de $\mathcal{M}$. Une première contrainte porte sur le rayon de giration minimal à partir d'une certaine vitesse de $\mathcal{M}$. En effet, lors d'une rotation, le véhicule est soumis à une force centrifuge $\gamma_\perp$ définie comme :

$$\gamma_\perp = \frac{{v_R}^2}{\rho} \tag{V.10}$$

où v_R représente la vitesse du point de référence R.
L'équation V.10 nous permet de déterminer une nouvelle valeur du rayon de giration minimal ρ_{min_2} :

$$\rho_{min_2} = \frac{{v_R}^2}{\gamma_{\perp max}} \tag{V.11}$$

avec $\gamma_{\perp max}$ représentant la force centrifuge maximale que peut supporter le véhicule sans glisser (i.e. garantissant une vitesse des points de contact roues-sol nulle).
Ainsi, les équations V.2 et V.11 nous permettent de connaître le rayon de giration minimal résultant de la prise en compte simultanée des contraintes mécaniques

et dynamiques précédemment exposées. Ce rayon ρ_{min} est donné par :

$$\rho_{min} = \max\{\rho_{min_1}, \rho_{min_2}\} \tag{V.12}$$

La seconde contrainte dynamique que nous considérons consiste à borner le domaine des accélérations possibles de $\mathcal{M}$. Nous avons donc :

$$\dot{v_R} \in [\dot{v_{Rmin}}, \dot{v_{Rmax}}] \tag{V.13}$$

où $\dot{v_{Rmin}}$ et $\dot{v_{Rmax}}$ représentent respectivement le freinage et l'accélération maximaux.

1.5. Environnement statique

Dans ce qui suit, on entend par environnement statique, l'ensemble des voies de circulation de l'environnement simulé.

L'entité de base de l'environnement statique, tant au niveau graphique que fonctionnel, est la *zone*. Au niveau graphique, une zone est associée à un parallélépipède[3] de hauteur négligeable et dont les dimensions sont liées au contexte routier considéré. La figure V.7 montre comment est effectué le zonage de l'environnement dans les contextes route et carrefour en croix. La construction géométrique de l'environnement statique peut être réalisée grâce au modeleur de ACT ou à partir d'un fichier d'entrée décrivant la cellule ACT associée à l'environnement statique désiré.

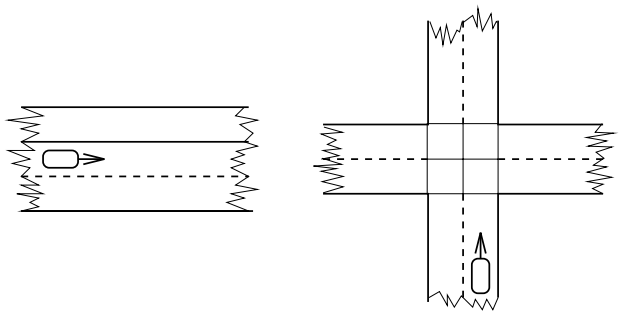


FIG. V.7 –: *Zonage dans les contextes route et carrefour*

1.6. Environnement dynamique

L'environnement dynamique est à la fois constitué de véhicules contrôlés[4] et de véhicules non contrôlés qui se contentent de suivre "en aveugle" la trajectoire

3. Les primitives de base manipulées par ACT sont des primitives volumiques.

4. Le simulateur ne supportait à l'origine qu'un seul véhicule contrôlé. La prise en compte de plusieurs véhicules contrôlés, due à Didier Pallard, permet d'associer aux véhicules, des contrôleurs de mouvements différents.

qui leur a été attribuée. Pour ce dernier type de véhicules, nous avons développé un planificateur simplifié permettant de générer de telles trajectoires et ce, à partir d'*intentions de manœuvre*.

Les intentions sont des instructions de haut niveau permettant de générer en premier lieu une trajectoire (chemin géométrique + profil de vitesse le long de ce chemin). Le modèle du véhicule est un modèle cinématique. Les intentions définies dans le simulateur sont de la forme *aller_tout_droit, tourner, s'immobiliser, ...*

La figure V.8 montre les entrées-sorties du planificateur simplifié. En entrée

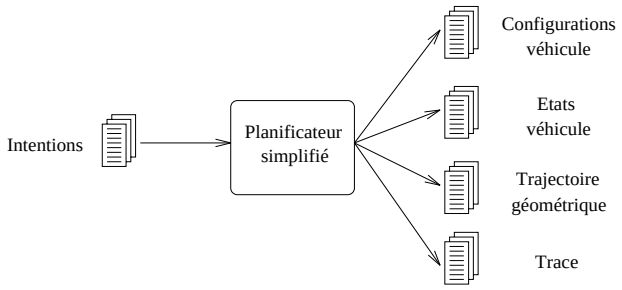


FIG. V.8 –: *Le planificateur simplifié*

du planificateur, nous avons un fichier d'intentions qui va permettre de générer respectivement un fichier de configurations datées (t, x, y, θ), un fichier d'états datés du véhicule $(t, v, \dot{v}, \ldots)$ et un fichier contenant le chemin géométrique résultant, sous forme d'une séquence de segments et d'arcs de cercle. Un dernier fichier permet de garder une trace des différents stades de l'élaboration de la trajectoire résultante.

1.7. Modélisation de la perception

La délivrance d'informations perceptives est l'un des points essentiels que doit fournir un simulateur. Nous avons choisi de fournir le maximum d'informations, quitte à n'utiliser qu'une partie de celles-ci, pour simuler au mieux la perception effective dont on dispose lors d'une application réelle. Nous avons rassemblé ces informations en quatre familles que nous allons détailler par la suite.

Informations proprioceptives

Les informations disponibles sont :

- la vitesse longitudinale;
- les accélérations longitudinale et latérale;
- le braquage;
- l'état des clignotants;

– la distance curviligne parcourue (odométrie).

Informations sur la situation courante

Les informations disponibles sont :

– le numéro de la voie courante;

– la distance au bord de la voie courante;

– l'orientation par rapport à l'axe de la voie.

Le numéro de la voie est donné avec la convention suivante, illustrée dans la figure V.9. La figure V.10 illustre ce que représentent les deux dernières informations.

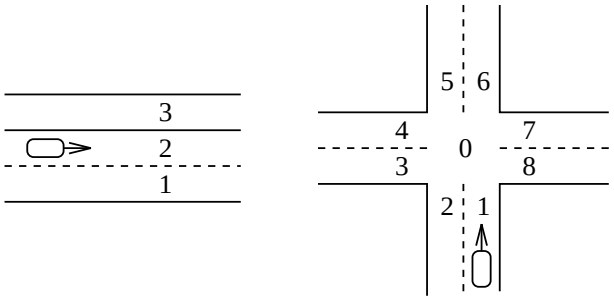


FIG. V.9 –: *Numérotation des voies dans le cas d'une route et d'un carrefour en croix*

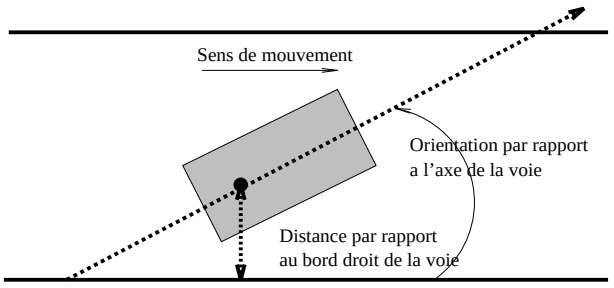


FIG. V.10 –: *Orientation et distance/bord droit de la voie*

Informations sur l'environnement dynamique

Le véhicule contrôlé est doté de zones d'intérêt pour chacune desquelles on dispose des informations suivantes (si un véhicule est détecté) :

– le type du véhicule (voiture, camion, ...);

- le numéro de la voie sur laquelle se trouve le véhicule;
- la configuration $(x_{rel}, y_{rel}, \theta_{rel})$ du véhicule dans le repère du véhicule contrôlé;
- la distance dans le repère associé à la zone d'intérêt;
- la vitesse relative (Vx_{rel}, Vy_{rel}) du véhicule par rapport au véhicule contrôlé;
- l'état des feux du véhicule (clignotants, feux arrière).

Une zone d'intérêt est définie dans le repère véhicule par cinq paramètres apparaissant dans la figure V.11. Il s'agit des coordonnées du repère associé à la zone dans le repère véhicule, de la portée et de l'ouverture de la zone. Les informations issues des différents capteurs du véhicule pourront être fusionnées et projetées dans la zone d'intérêt idoine.

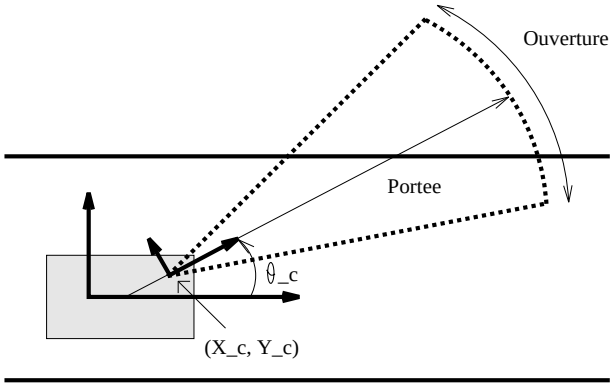


FIG. V.11 –: *Caractérisation d'une zone d'intérêt*

Informations sur l'environnement statique

Les informations disponibles sont :

- le numéro de la voie;
- le sens de circulation de la voie;
- le nombre de voies à droite et à gauche;
- la largeur de la voie;
- la nature des délimiteurs droit et gauche de la voie.

1.8. Communication simulateur – application

Comme cela a été dit précédemment, une attention toute particulière a été donnée à l'interface entre le simulateur et l'application courante, permettant de s'affranchir d'un certain nombre de considérations matérielles comme les systèmes d'exploitation et les machines supportant les deux parties communicantes [21]. Pour cela, nous avons opté pour une communication basée sur l'utilisation de *sockets*[5] avec le protocole TCP/IP correspondant respectivement aux couches transport et réseau de la norme OSI[6]. Cette communication se fait selon le principe d'un échange client-serveur.

Architectures matérielles

Nous avons implanté notre système de simulation sur deux architectures distinctes. Dans ces architectures, la partie simulateur se trouve toujours sur une Silicon Graphics, seule machine supportant le logiciel ACT. Diffère le support de la partie application : dans un premier cas, il s'agit d'une station de travail avec le système d'exploitation Unix, dans le second cas nous avons une carte utilisant le système temps-réel VxWorks. La seule contrainte matérielle réside dans le fait que simulateur et application soient connectés au réseau Internet.

– Architecture Unix-Unix
La figure V.12 représente l'architecture de notre système de simulation dans le cas où l'application se trouve sur une station de travail. Cette architecture

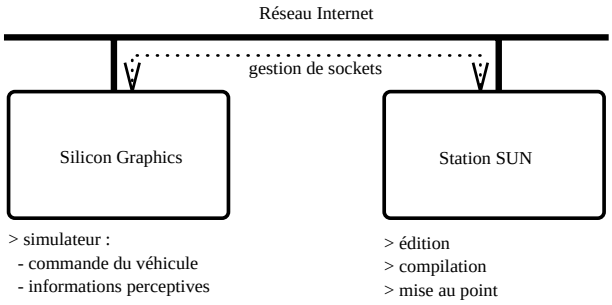


FIG. V.12 –: *Architecture Unix-Unix*

permet de travailler, au niveau de l'application, dans un environnement de développement complet.

– Architecture Unix-VxWorks
Lors de l'intégration du démonstrateur Prolab II[7] dans lequel était impli-

5. Mécanisme de communication UNIX fonctionnant également sous le système temps réel VxWorks.

6. Acronyme de Open Systems Interconnection.

7. Démonstrateur français du programme européen EUREKA-PROMETHEUS, action PRO-ART dont le but était l'assistance à la conduite automobile. Ce démonstrateur (véhicule Peu-

qué le projet Sharp, il a été nécessaire d'implanter l'application sous un système temps-réel. La figure V.13 représente l'architecture de ce système de simulation. L'application se trouve sur une carte MVME 167 (Motorola 68040) sous le système temps-réel VxWorks. Cette carte équipe également le véhicule expérimental sur lequel nous avons travaillé.

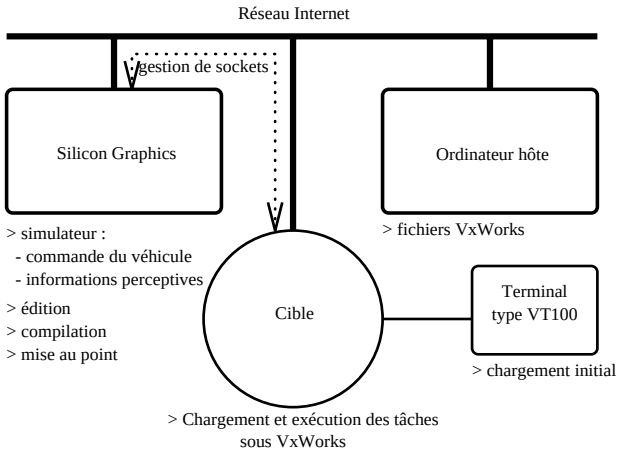


FIG. V.13 –: *Architecture Unix-VxWorks*

Dans ces deux architectures, le protocole de communication, à base de sockets, est identique en tout point.

1.9. Résultats

Nous allons montrer quelques résultats obtenus avec notre simulateur et portant essentiellement sur le comportement "évitement d'obstacles avec coopération passive" de notre contrôleur. Des résultats concernant le suivi de trajectoires ont déjà été présenté en § III. Ces résultats vont être exposés à travers deux exemples. Le premier porte sur un scénario d'évitement dans le cas général i.e. ne se situant pas dans un contexte particulier. Le second correspond à un scénario faisant intervenir le contexte, en l'occurrence le contexte carrefour.

Non prise en compte du contexte

Ce premier exemple illustre le comportement "évitement d'obstacles avec coopération passive" de notre contrôleur dans le cadre général. Dans cet exemple (figures V.14, V.15 et V.16), deux véhicules roulent face à face, au centre d'une voie, centre matérialisé par une ligne continue. Les trajectoires nominales sont rectilignes et opposées. Les deux véhicules ont été volontairement placés au centre

geot 605) a été présenté lors de la manifestation européenne PROMETHEUS Board Members Meeting'94.

de la voie pour mieux illustrer l'aspect de coopération, ce qui aurait été moins visible si ces derniers s'étaient trouvés sur les bords de la voie. En effet, dans ce cas, un des véhicules donnerait l'impression de ne pas changer de comportement en présence de l'autre véhicule.

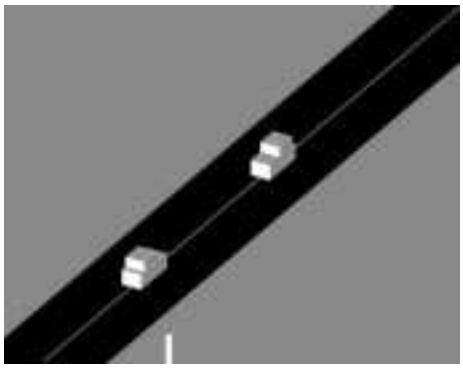


FIG. V.14 –: *Exemple d'évitement d'obstacles avec coopération passive dans le cas général : configuration initiale*

FIG. V.15 –: *Exemple d'évitement d'obstacles avec coopération passive dans le cas général : modification locale de la trajectoire nominale*

La figure V.17 illustre le phénomène d'oscillations décrit dans le chapitre III. Comme nous l'avons vu, celles-ci résultent de l'absence d'une trajectoire locale d'évitement et de profondeur de champs suffisante des capteurs. Ce dernier point occasionne une détection trop tardive d'un obstacle et ne permet pas, de ce fait, de suivre une trajectoire lissée.

Prise en compte du contexte

Ce second exemple illustre la prise en compte du contexte au niveau du comportement "d'évitement d'obstacles avec coopération passive". Les règles cor-

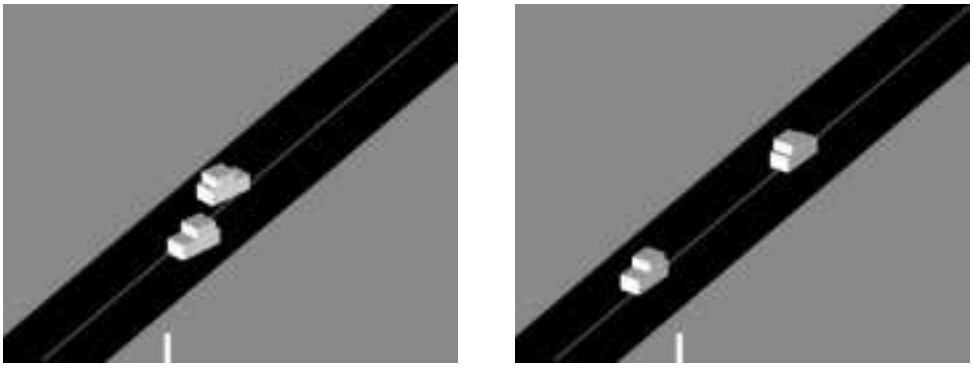


FIG. V.16 –: *Exemple d'évitement d'obstacles avec coopération passive dans le cas général : fin de manœuvre d'évitement et rattrapage de la trajectoire nominale*

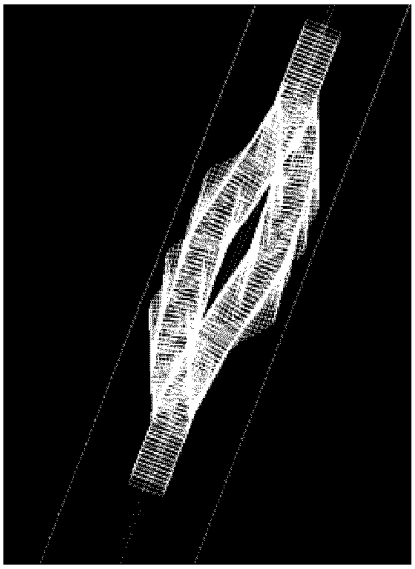


FIG. V.17 –: *Phénomène d'oscillations au niveau de la trajectoire suivie par les véhicules durant la manœuvre d'évitement*

respondant à l'évitement d'obstacles dans le cas général (sans prise en compte du contexte) sont toujours activées, mais constituent en fait une tâche de fond du comportement intégrant le contexte[8]. Dans le cadre du carrefour, le type de coopération passive retenue est la priorité à droite.

Dans l'exemple proposé, trois véhicules abordent un carrefour d'une manière totalement symétrique, avec des trajectoires nominales rectilignes et sans arrêt à l'entrée du carrefour. La non prise en compte du contexte provoquerait une situation de blocage dans le cœur du carrefour (figure V.21). Par contre, la notion de priorité à droite permet à chaque véhicule de franchir convenablement le carrefour (figures V.18, V.19 et V.20). avant de rattraper la trajectoire nominale[9].

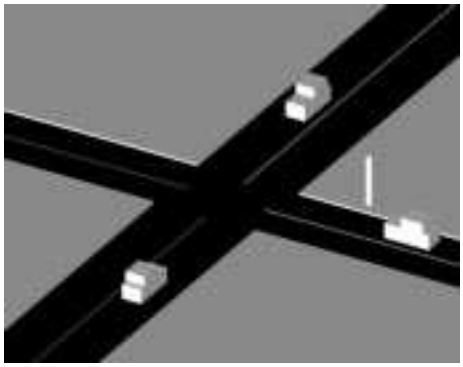


FIG. V.18 –: *Exemple d'évitement d'obstacles avec coopération passive prenant en compte le contexte : configuration initiale*

2. Expérimentations

Les résultats obtenus en simulation nous conduisent actuellement à des expérimentations sur un véhicule réel, au sein du projet Praxitèle de l'INRIA/INRETS, qui constitue le cadre d'application de notre contrôleur d'exécution. Nous allons décrire le concept de TUPI dans lequel se situe le projet, les caractéristiques du véhicule expérimental et l'outil de spécification de la tâche robotique associée à notre architecture de contrôle : Orccad.

8. Cela peut être aisément codé à travers des poids de règles intégrant le contexte, plus importants que ceux associées aux règles d'évitement dans le cas général.

9. Ce rattrapage se traduit par une accélération des véhicules ayant dû stopper à l'entrée du carrefour.

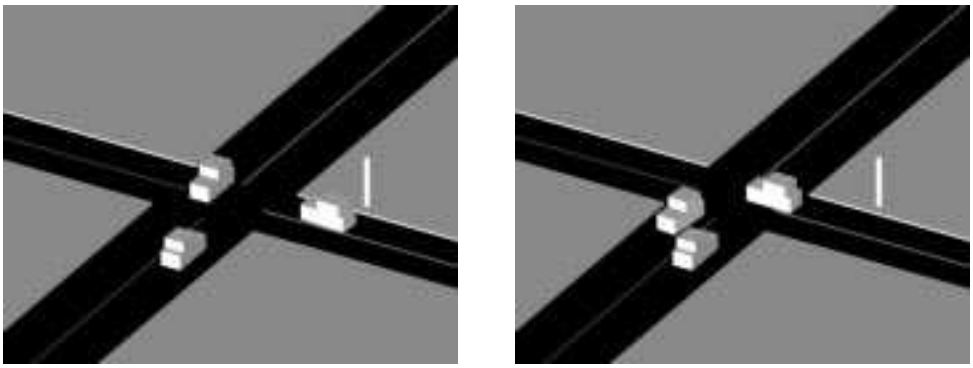


FIG. V.19 –: *Exemple d'évitement d'obstacles avec coopération passive prenant en compte le contexte : notion de priorité à droite*

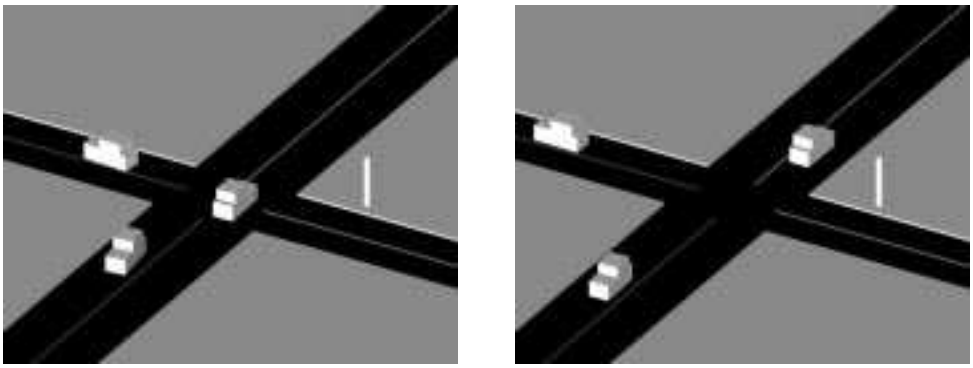


FIG. V.20 –: *Exemple d'évitement d'obstacles avec coopération passive prenant en compte le contexte : rattrapage des trajectoires nominales*

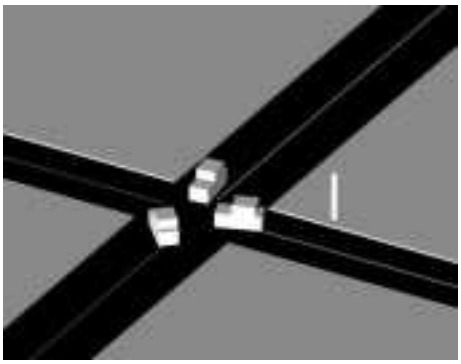


FIG. V.21 –: *Situation de blocage due à la non prise en compte du contexte dans le processus d'évitement d'obstacles*

2.1. Le projet Praxitèle

Le concept de TUPI

Dans la vie de tous les jours, nous sommes confrontés à un certain nombre de problèmes résultant d'un accroissement vertigineux du parc automobile. Sans rechercher une énumération exhaustive de ces problèmes, nous pouvons citer : la congestion des centres urbains, un taux de pollution alarmant et l'augmentation des risques potentiels d'accident. Cependant, des mesures coercitives comme celles prises dans certains pays, tendant à limiter la flotte de véhicules (circulation un jour sur deux par exemple), apparaissent comme des réponses de première urgence. C'est pour cela que les pouvoirs publics s'orientent maintenant vers d'autres solutions plus souples, mettant en œuvre des systèmes informatiques contrôlant aussi bien les infrastructures fixes (carrefours intelligents de l'INRETS [36]) que les véhicules (programme européen Eureka PROMETHEUS [57]) dans le but d'éradiquer les problèmes exposés ci-dessus.

Le programme Praxitèle [59, 58] est mené depuis mai 1993 (après une période de pré-études de 18 mois) avec un consortium formé de l'INRIA, l'INRETS, la CGEA (filiale transports publics de la Compagnie Générale des Eaux), Renault et EDF. Ce programme s'appuie sur la mise en œuvre d'un nouveau mode de transport dénommé TUPI[10], ou TULIP[11] pour ce qui est du programme du constructeur français PSA. Le concept de TUPI réside dans la gestion d'une flotte de petits véhicules électriques accessibles en location. Son but est de concilier les aspirations individuelles et collectives. En effet, il combine les avantages de la voiture individuelle (liberté et souplesse d'utilisation, ...) et ceux des transports publics (pas d'achat, pas d'entretien, ...). Ce nouveau moyen de transport doit être vu comme un complément aux infrastructures de transport classiques.

Description du projet

Le principe de fonctionnement du concept de TUPI, vu par l'INRIA, consiste donc à gérer une flotte de véhicules électriques à partir d'un central chargé de la répartition des véhicules sur le réseau couvert par l'application, l'entretien de ces véhicules et la facturation. Le projet est divisé en plusieurs phases tendant vers une automatisation croissante du véhicule. L'activité d'un véhicule de la flotte pourra, dans un futur à court terme, se résumer de la manière suivante :

- un client prend possession d'un véhicule dans une station dite *de proximité* et l'abandonne après utilisation dans une autre station, voire même en bord de route;
- le véhicule est récupéré par accrochage automatique et immatériel au passage d'un train constitué d'autres véhicules et chargé de la collecte de ces

10. Acronyme de Transport Urbain Public Individuel.
11. Acronyme de Transport Urbain Libre Individuel et Public !

véhicules (opérations de "haut le pied");

- le véhicule peut, à la demande du central, quitter automatiquement le train pour réalimenter une station de proximité lorsque celui-ci passe devant, ou être conduit dans des parkings haute densité de capacité supérieure aux stations de proximité;

- à l'intérieur d'un parking haute densité, le véhicule peut recharger automatiquement ses batteries sur des bornes à induction, être réparé et bien évidemment stocké dans l'attente d'une utilisation ultérieure.

Le programme industriel prévoit les premières expérimentations (avec des véhicules non-automatisés) à la fin de l'année 1995 sur le site de Saint-Quentin-en-Yvelines. Ces expérimentations concerneront une flotte d'une centaine de véhicules électriques (Renault Clio).

2.2. Le véhicule expérimental

Généralités

Le véhicule expérimental est un petit véhicule de type *"voiture sans permis"* (figure V.22) de marque Ligier. La transformation du véhicule originel (équipé

FIG. V.22 –: *Le véhicule expérimental*

d'un moteur thermique) en un véhicule électrique fonctionnant aussi bien en mode conventionnel qu'automatique a été assurée par la société ALEPH Technologies. Ces adaptations concernent principalement le système de direction, de freinage de route et de frein de parking. Pour chacune de ces fonctions, un dispositif électromagnétique pilotable à partir d'un bus VME a été adapté au système conventionnel du véhicule. En ce qui concerne tous les aspects mécaniques et

électromécaniques, ALEPH Technologies a sous traité la société SSP [12], se chargeant quant à elle de fournir le rack VME et les outils logiciels pour le pilotage du véhicule.

Les capteurs

Le véhicule est doté de plusieurs capteurs proprioceptifs et extéroceptifs. Nous ne présentons ici que les deux principales sources d'informations extéroceptives dont nous disposons, à savoir une caméra linéaire et une antenne ultrasonore. Ces dernières sont reliées à un PC dans lequel sont traitées les données qui sont ensuite transmises à un rack VME via une liaison série (cf. figure V.23). Ce rack VME contient notamment une carte VME162 supportant l'architecture de contrôle/commande du véhicule.

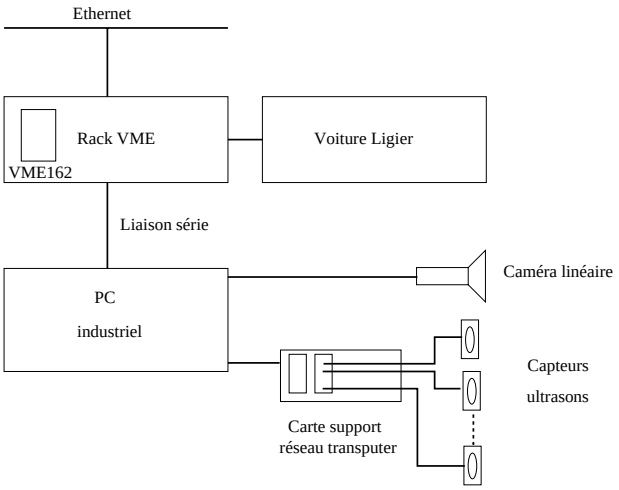


FIG. V.23 –: *Architecture matérielle de perception et de contrôle/commande du véhicule expérimental*

- Caméra linéaire
  Cette caméra linéaire, d'une résolution de 2048 pixels, a été utilisée à l'INRIA de Rocquencourt pour des expérimentations sur un suivi automatique de véhicule : le but est de parvenir à des trains de véhicules sans chauffeur, accrochés de manière immatérielle, et suivant le véhicule leader piloté manuellement [14].

- Antenne ultrasonore
  Une antenne ultrasonore a été développée au sein de notre équipe [56]. Celle-ci s'articule autour d'une architecture à base de transputers. L'organisation matérielle est la suivante : à l'intérieur du PC a été insérée une carte

12. Scholl Sun Power.

support de réseau transputer (carte BO08 d'INMOS). Cette carte peut supporter jusqu'à 9 cartes filles à transputer appelées TRAM. Une carte double TRAM T225 a été mise au point pour gérer 14 capteurs ultrasons de la série 9000 de Polaroïd. Ces capteurs étanches ont une portée d'environ 10 mètres et une demi-ouverture de 20 degrés. L'avantage de cette approche à base de transputers réside essentiellement dans une plus grande puissance de calcul ainsi que dans la possibilité d'évolution de l'architecture de perception; en effet, par la suite, d'autres capteurs pourront être reliés à une carte TRAM de la carte support. Le principe de fonctionnement de l'antenne consiste à synchroniser les capteurs tant au niveau de l'émission que de la réception. Il est possible d'émettre sur un ou plusieurs capteurs et on reçoit les échos sur tous les capteurs. D'après [56], l'intérêt est triple :

1. par triangulation, il est possible de déterminer la position angulaire d'un obstacle;
2. le fait d'utiliser plusieurs capteurs en émission permet de détecter des obstacles fins ou spéculaires mal orientés;
3. l'utilisation synchrone des capteurs permet de s'affranchir de mesures séquentielles, et de son temps prohibitif.

La disposition des capteurs ultrasons apparaît dans la figure V.24. Seules sont visualisées les orientations de ces derniers.

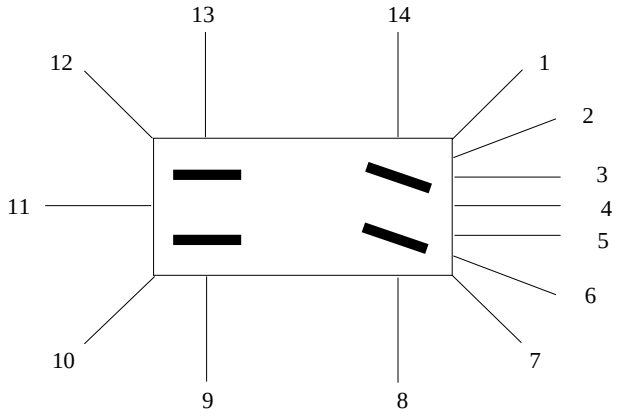


FIG. V.24 –: *Disposition des capteurs ultrasons sur le véhicule expérimental*

Les actionneurs

Le driver du véhicule a été réalisé par la société Aleph Technologies. Les commandes que l'on peut appliquer au véhicule concernent :

– **la locomotion** avec la spécification du sens de rotation du moteur (sortie tout ou rien);

- **la direction**
- **le frein de route**
- **le frein de parking**

Toutes ces commandes moteur sont des signaux analogiques permettant de commander ces derniers en intensité. Une description complète du driver est faite dans [71]. Les asservissements portant sur la locomotion et la direction se font respectivement en accélération ($\dot{v}$) et en angle de braquage (ϕ). Dans la mesure où notre contrôleur d'exécution de mouvements génère des consignes du type ($\dot{v}, \dot{\phi}$), nous devrons donc intégrer la composante "vitesse de braquage" avant d'envoyer les consignes aux asservissements.

2.3. L'architecture de contrôle/commande : ORCCAD

L'architecture de contrôle/commande du véhicule a été réalisée grâce au système ORCCAD [13][65]. ORCCAD est un environnement permettant la conception, la visualisation, la simulation et enfin l'exécution de systèmes génériques de contrôle/commande de robots. Sous ORCCAD, une application robotique est découpée en un ensemble de **tâches robots** correspondant à un comportement local contrôlé par des observateurs et représenté par un automate d'états finis. Une tâche robot est elle-même constituée de tâches élémentaires temps-réel communicantes : les **tâches modules**.

Une tâche robot apparaît sous la forme d'un bloc-diagramme : les tâches modules représentées par des rectangles, communiquent via des ports de communication typés. L'algorithme générique exécuté par les tâches robots est le suivant [65] :

```
Code d'initialisation;
while(1)
{
  Lire les ports d'entrée;
  Traitement des données;
  Ecrire dans les ports de sortie;
}
```

Le corps de la boucle correspond au traitement périodique de la tâche. Une période d'activation est associée par l'utilisateur à chaque tâche.

Les différentes étapes de l'élaboration d'une tâche robot sont les suivantes :

1. tout d'abord, on spécifie les tâches modules avec les communications inter-tâches. Cela peut être réalisé grâce à l'interface graphique d'ORCCAD (cf. figure V.25). Le langage synchrone ESTEREL [6] est utilisé pour définir des automates de gestion des tâches robots. Un certain nombre de preuves sémantiques peuvent être effectuées sur ces automates;

13. Acronyme de Open Robot Controller Computer-Aided Design.

2. un générateur de programme permet de produire un programme en langage C;

3. le programme C est compilé sous le système temps-réel VxWorks;

4. des simulations du comportement d'une tâche robot spécifié peuvent être réalisées grâce au logiciel de simulation SIMPARC [2];

5. le programme VxWorks peut être chargé et exécuté sur la carte cible du robot.

La figure V.25 montre la tâche robot associée à notre application sous Orccad : on peut remarquer en particulier la tâche module FLC (Fuzzy Logic Controller) correspondant à notre contrôleur d'exécution de mouvements.

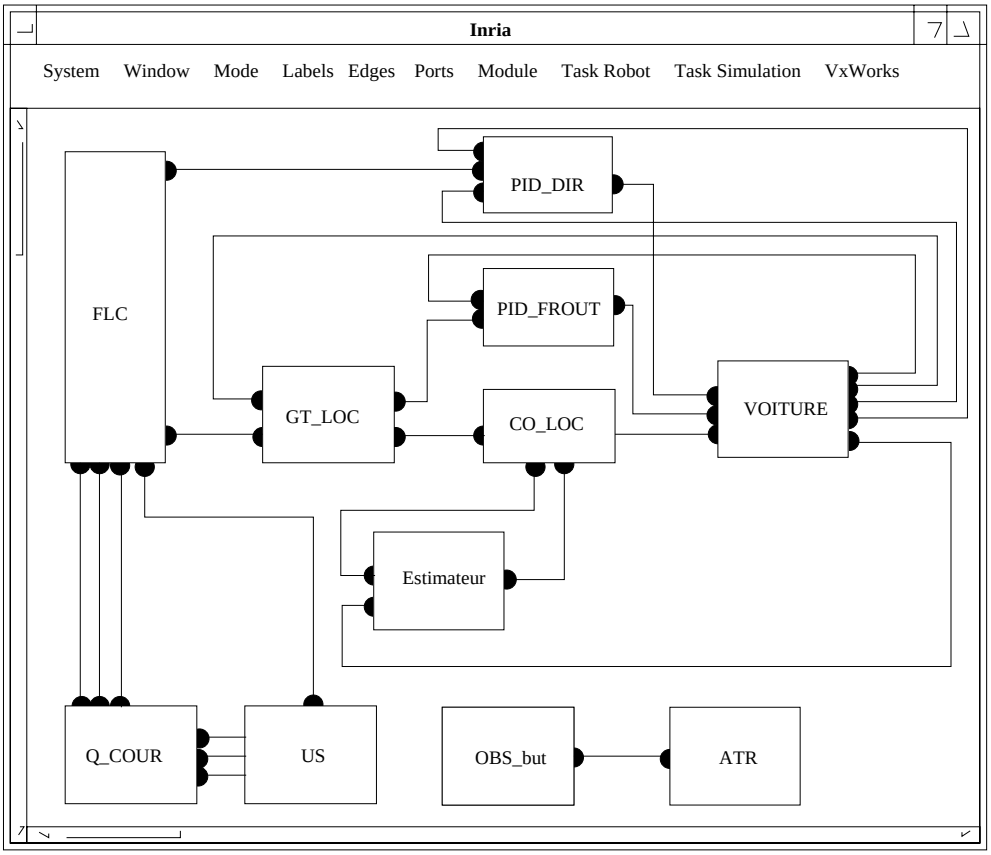


FIG. V.25 –: *Tâche robot associée à notre application sous Orccad*

Conclusion et perspectives

Nous avons présenté, dans ce mémoire, un contrôleur d'exécution réactif de mouvements pour un robot mobile.

Ce contrôleur se situe au sein d'une architecture de contrôle hybride. Une telle architecture comporte des modules de haut niveau (génération de missions, planification de trajectoires), s'appuyant sur un raisonnement approfondi mais nécessitant un certain temps de calcul, et notre contrôleur, garantissant la réactivité du système. Cet aspect réactif est indispensable pour un robot autonome évoluant dans un environnement dynamique. En effet, dans un tel environnement, les modules de haut niveau font appel à une prédiction sur l'évolution de l'environnement, pour délivrer une séquence de mouvements sur un certain horizon temporel. Or, cette prévision peut s'avérer partiellement fausse lors de l'exécution effective de ces mouvements par le robot. Cet état de chose peut résulter, par exemple, du changement de comportement de certains obstacles dynamiques ou d'erreurs de localisation du robot (phénomènes de glissement, de patinage des roues, etc). Le contrôleur d'exécution de mouvements permet de travailler sur une boucle de contrôle rapide, avec la nécessité de réagir promptement aux aléas pouvant survenir. Son rôle est donc de s'assurer de la bonne réalisation des mouvements planifiés, en les adaptant localement si cela est nécessaire.

Notre contrôleur est basé sur l'utilisation de la logique floue. Cette approche permet de coder celui-ci sous la forme d'un ensemble de règles linguistiques décrivant son comportement attendu sur le véhicule. A ce niveau, cette approche constitue un avantage certain par rapport à la plupart des autres méthodes qui s'appuient sur une modélisation du véhicule. De plus, dans le cadre de notre contrôleur d'exécution de mouvements, nous avons pu coder, au sein d'un même formalisme, des comportements aussi divers que, par exemple, le suivi de trajectoire et l'évitement d'obstacles. Il est à noter aussi que le codage sous la forme de règles floues permet d'intégrer, de manière élégante, la notion d'imprécision apparaissant dans tout système réel. Enfin, on retrouve un avantage inhérent aux bases de règles, à savoir une relative facilité de mise à jour de celles-ci.

Le regroupement des comportements associés au contrôleur au sein d'une même base nous a amené à devoir effectuer un "réglage" de celui-ci. Ce réglage, réalisé par apprentissage paramétrique supervisé, porte sur les poids associés à

chaque règle et caractérisant l'importance relative de chacune par rapport aux autres. L'apprentissage consiste à soumettre au contrôleur un ensemble d'échantillons correspondant à des entrées-sorties désirées. Pour réaliser cet apprentissage, nous avons utilisé l'algorithme du simplexe issu de l'optimisation combinatoire. Cela résulte de la constatation qu'un problème d'apprentissage peut être aisément codé sous la forme d'un programme linéaire dont l'algorithme de résolution le plus efficace est celui du simplexe.

Notre véhicule a été testé essentiellement en simulation, mais nous avons cependant commencé des expérimentations sur véhicule réel. Les résultats obtenus sont encourageants et nous ont permis de confirmer les avantages de l'approche floue précédemment exposés. Ces avantages ont aussi été démontrés à travers la mise au point d'un générateur de missions. Ce dernier se situe dans le cadre d'un parking automatisé, mais est généralisable à toute application que l'on peut modéliser sous la forme d'un graphe d'états, où les transitions entre états doivent prendre en compte des aspects multi-critères.

Comme travaux futurs venant en complément de ceux effectués dans le cadre de cette thèse, on peut citer l'ajout de nouveaux comportements intégrant, si cela est possible, davantage d'informations sur la structure de l'environnement statique notamment; en particulier, la notion de voie de circulation apporterait un plus indéniable, permettant de coder des comportements analogues à ceux du code de la route, par exemple. Cependant, ces informations sur la structure de l'environnement statique nécessitent un surcroît de perception (i.e. de capteurs) pas toujours compatible avec le matériel réel dont nous disposons. Pour cette raison, au niveau de notre contrôleur, bien que ces informations fussent disponibles en simulation, nous avons préféré ne pas intégrer cet aspect structurel. Une prise en compte de vitesses plus grandes pour les véhicules serait également à envisager; dans notre travail, nous avons essentiellement considéré que les véhicules évoluaient dans un environnement ne nécessitant pas de vitesses importantes (parking automatisé). Cependant, ce travail ne pourra être réalisé que si les véhicules sont dotés de capteurs d'une portée supérieure à celle disponible actuellement.

Enfin, bien que tous nos tests aient été effectués sans problèmes (en particulier, au niveau des collisions), une telle approche floue souffre d'une absence de preuve quant à la stabilité du système. Cela nécessite, dans des applications devant être fiables, d'accroître les filtrages des sorties du contrôleur flou, essentiellement pour les non-collisions. Nous restons persuadé que cela ne doit pas remettre en question l'utilisation de la logique floue trop souvent ternie par cette absence de preuve. Il est d'ailleurs fort probable que cette lacune soit simplement due à un domaine relativement jeune, non pas au niveau de la théorie de base, mais des applications. Nul doute que des travaux futurs devraient permettre de solutionner ce problème.

Annexe A

Générateur de missions dans un parking automatisé

Les résultats obtenus au niveau du contrôle d'exécution de mouvements nous ont amené à utiliser la logique floue dans le cadre d'une application précise : la génération de missions au sein d'un parking automatisé, également dans le cadre du projet Praxitèle. Dans un tel contexte, la relative facilité d'expression des comportements attendus sous forme de règles linguistiques, ainsi que les possibilités d'agrégation multi-critères que permet la logique floue, nous ont permis de développer ce générateur de missions.

1. Introduction

Une première phase du projet Praxitèle consiste à gérer de manière automatique une flotte de véhicules électriques autonomes, à l'intérieur d'un parking haute densité. Les fonctionnalités attendues de ce parking sont bien évidemment le stockage, mais aussi la recharge des batteries embarquées, voire la réparation des véhicules. Dans un tel parking, la présence humaine se limite à des tâches de télésurveillance, de réparation ou de récupération de véhicules hors-service. La figure A.1 montre l'architecture de commande pour un véhicule de la flotte. Il s'agit là d'une architecture communément utilisée en robotique mobile.

Un calculateur central [44], en fonction de données sur le parking et sur les véhicules qu'il contient, assigne des missions à ces derniers. Sous sa forme minimale, une mission peut se ramener à une consigne du type *"aller en* (x, y, θ)*"*, où (x, y, θ) représente une configuration. Cela nécessite la possibilité pour un véhicule de savoir se localiser et disposer d'une carte de l'environnement associé au parking.

Après avoir reçu une mission, le véhicule considéré pourra activer un module de planification embarqué, afin de générer un plan nominal sous la forme d'une

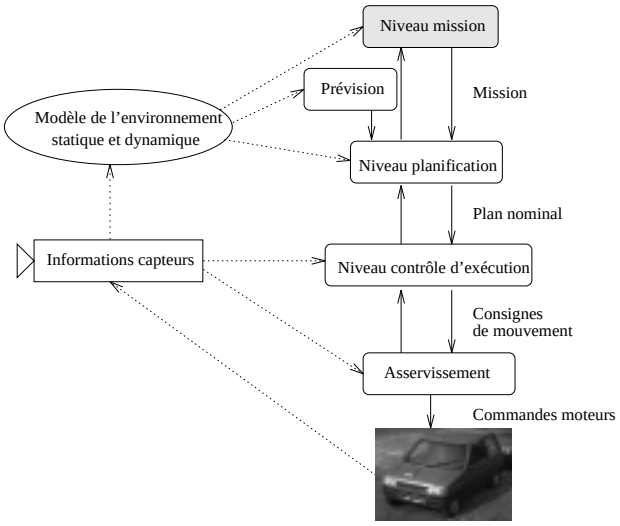


FIG. A.1 –: *Architecture de commande pour un véhicule*

trajectoire (i.e. chemin géométrique + profil de vitesse/accélération) permettant de réaliser la mission [19]. Ce plan pourra être ensuite fourni à un contrôleur d'exécution de mouvements réactif qui, à partir d'informations tant proprioceptives qu'extéroceptives, surveille la validité du plan nominal et l'adapte en temps réel en fournissant des consignes de mouvements au module d'asservissements des moteurs du véhicule (actionneurs de locomotion, freinage et direction).
Dans la suite de ce chapitre, nous allons focaliser notre attention sur le niveau mission.

2. Modélisation du problème

2.1. Environnement structuré en zones fonctionnelles

Sans connaissances a priori sur la topologie du parking, nous avons opté pour un découpage de celui-ci en zones fonctionnelles. Cela se justifie par (1) un découpage implicite du parking en zones de "services"(stockage, réparations, etc.), (2) un souci de généralisation de notre méthode à tout environnement structuré en zones fonctionnelles et (3) une volonté de meilleure gestion du parking. En ce qui concerne le dernier point, nous pensons par exemple à l'idée relativement intuitive consistant à gérer la flotte de véhicules en files; cette approche, bien qu'intéressante au premier abord, dans la mesure où elle ne nécessiterait ni mission ni planification, n'apparaît absolument pas optimale au niveau de la gestion de la flotte. En effet, une telle organisation fait rapidement apparaître des situations de blocages (cas de véhicules défectueux notamment) et une mauvaise répartition des ressources, privilégiant l'ordre d'arrivée des véhicules dans le parking au détriment de leur besoins propres (recharge, réparations, stockage).

Le parking haute densité que nous avons considéré apparaît dans la figure A.2. Celui-ci est constitué d'un silo relié à une station par l'intermédiaire de

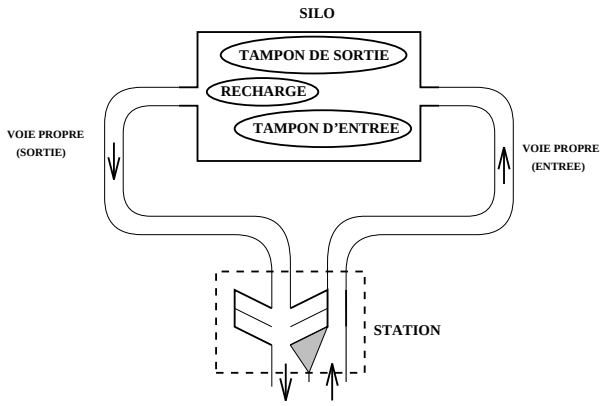


FIG. A.2 –: *Découpage fonctionnel du parking*

voies propres. La station joue le rôle d'interface entre le silo automatisé et le monde extérieur. Le silo est constitué de zones assurant en premier lieu le stockage des véhicules mais aussi la recharge des batteries.

2.2. Graphe d'états

Pour un véhicule donné, il est possible de représenter l'évolution possible de celui-ci dans tout environnement structuré en zones fonctionnelles, en l'occurrence le parking, par un graphe d'états (figure A.3) dans lequel un nœud représente

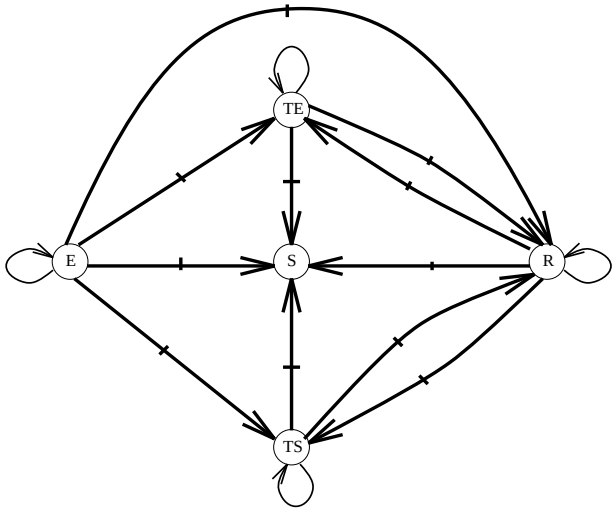


FIG. A.3 –: *Graphe d'états associé au parking pour un véhicule*

une zone, et un arc, une transition que peut effectuer un véhicule. On reconnaît facilement les zones d'entrée (E) et de sortie (S). Il s'agit maintenant de trouver un mécanisme permettant de déterminer les transitions que va effectuer chaque véhicule à l'intérieur du parking.

2.3. Forces de transition

Nous définissons la notion de *force de transition*. Chaque transition du graphe d'états précédent va être pondérée par une force qui détermine l'aptitude du véhicule à la franchir. Ainsi, pour chaque véhicule, le gestionnaire de missions, en fonction d'un certain nombre d'informations sur le véhicule (niveau de charge, distance parcourue par rapport à la moyenne de la flotte, zone courante, ...) et d'informations plus générales (moment de la journée caractérisant l'offre et la demande, taux de remplissage des diverses zones du parking, ...) va déterminer ces "forces" de transition dans le graphe d'états. L'idée est que, pour un véhicule se trouvant dans une zone déterminée, on va privilégier tout naturellement la transition dont la force associée est la plus importante.

2.4. Agrégation multi-critères

Comme cela vient d'être dit en 2.3., un certain nombre de critères interviennent dans la détermination des forces de transition. Chaque critère pris indépendamment va agir sur les forces de quelques transitions du graphe. Afin de déterminer les valeurs résultant de la prise en compte de tous les critères, il faut utiliser un mécanisme de fusion. On parle alors d'*agrégation multi-critères*. Ce mécanisme a pour but d'intégrer, dans un même processus de décision, des considérations de natures différentes.

2.5. Algorithme de gestion du parking

L'algorithme du gestionnaire de missions est relativement simple; il consiste à répéter le cycle suivant :

1. pour chaque véhicule sans mission, **calculer les forces de transition du graphe d'états**.

2. **satisfaire la demande en sortie de parking** en déterminant les véhicules du parking les plus aptes à sortir (i.e. les véhicules ayant les plus grandes forces de transition vers la zone sortie).

3. **mise à jour de la zone recharge** en remplaçant les véhicules rechargés.

4. **stockage des véhicules arrivant en entrée de parking**.

3. Utilisation de la logique floue

L'approche que nous avons utilisée pour la détermination des forces de transition (première phase de l'algorithme présenté en 2.5.) est la logique floue. En effet, le recours à cette approche est essentiellement motivé par deux considérations :

- les règles linguistiques caractérisant un système flou se prêtent bien à la description du processus permettant de déterminer les forces de transition, comme nous le verrons à travers la présentation de la base de règles (3.2.);

- la logique floue est particulièrement bien adaptée (à travers la phase de défuzzification) à la résolution de problèmes nécessitant une fusion multi-critères;

- facilité de mise à jour de la connaissance codée sous forme de règles.

3.1. Entrées/sorties du système flou

Les entrées du système sont :

- la zone courante
  il s'agit en fait d'une information non floue, qui peut cependant être intégrée très facilement, l'aspect flou n'étant qu'une généralisation de l'aspect non flou;

- niveau de charge
  on considère le niveau de charge électrique des batteries embarquées;

- distance parcourue par rapport à la moyenne de la flotte
  cette donnée est utilisée dans le but d'avoir une iso-utilisation de la flotte et, de ce fait, éviter le phénomène de "famine" pour certains véhicules;

- demande et offre potentielle
  on désire à travers ces informations caractériser les flux de véhicules en entrée et en sortie du parking. L'idée est que ceux-ci peuvent varier au cours de la journée. Prenons le cas d'un parking situé dans un centre ville : le matin, les personnes venant de la périphérie et se rendant à leur lieu de travail, provoqueront un afflux de véhicules en entrée du parking. Comportement dual : en fin de journée, le parking sera soumis à une forte demande de la part des utilisateurs de véhicules. Ces aspects peuvent avoir une incidence non négligeable sur la gestion du parking.

Les sorties du système se limitent aux valeurs des forces de transition du graphe d'états présenté en 2.2..

3.2. Base de règles floues

Nous présentons ici un extrait de la base de règles utilisées dans la détermination des forces de transition du graphe de la figure A.3.

```
/* niveau de charge */
...
si ((zone   est ENTREE) et
    (niveau est NIVEAU_FAIBLE))
         alors (force_E-->R est TRES_IMPORTANTE);

si ((zone   est ENTREE) et
    (niveau est NIVEAU_MOYEN))
         alors (force_E-->R est MOYENNE);
...

/* distance parcourue/moyenne */
...
si ((zone     est ENTREE) et
    (distance est INFERIEURE))
         alors (force_E->TS est IMPORTANTE);

si ((zone     est ENTREE) et
    (distance est INFERIEURE))
         alors (force_E-->R est FAIBLE);
...
```

Nous avons fait apparaître deux comportements : le premier illustre la prise en compte du niveau de charge des batteries dans le calcul de la force de transition de la zone entrée vers la zone de recharge; le second illustre la tendance à privilégier la sortie d'un véhicule, lorsque celui-ci a moins roulé que la moyenne.

4. Expérimentations

4.1. Architecture de validation

Dans le but de valider notre gestionnaire de missions, nous avons développé un simulateur graphique communiquant avec un système flou par un mécanisme de boîte aux lettres (files de messages). Les messages échangés concernent les entrées-sorties du système flou. La figure A.4 illustre l'architecture utilisée. Les fonctionnalités proposées par le simulateur sont les suivantes :

- simulation d'arrivée de véhicules en entrée de parking, avec des caractéristiques générées de manière aléatoire;

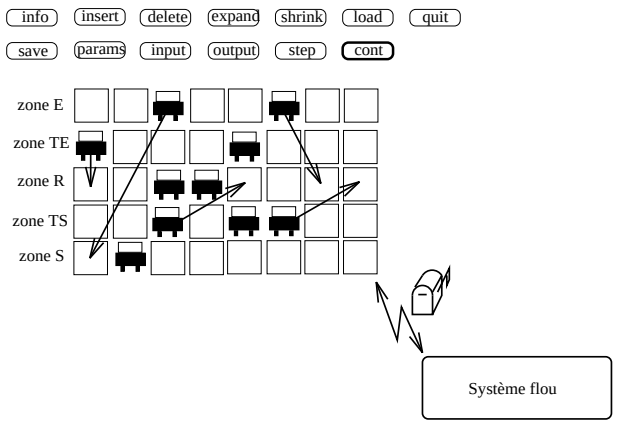


FIG. A.4 –: *Architecture de validation du générateur de missions*

- simulation de départ de véhicules en sortie de parking;
- visualisation des transitions effectuées ainsi que leur historique;
- affichage des informations relatives à un véhicule;
- insertion ou suppression d'un véhicule;
- agrandissement et réduction d'une zone;
- exécution en mode pas à pas ou continu.

Celle-ci nous a permis de voir l'évolution globale du parking, pour différents contextes donnés, et s'est avérée constituer un outil de mise au point intéressant.

4.2. Résultats

Nous allons étudier un scénario à l'aide de notre simulateur et décrire quelques étapes.

Un certain nombre de véhicules se trouvent en entrée du parking (A.5).

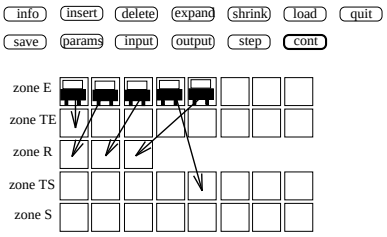


FIG. A.5 –: *Simulation du générateur de missions : première étape*

Pour chaque véhicule, on détermine les forces de transition du graphe d'états associé, par activation du système flou. La zone de recharge se remplit et le reste

des véhicules se répartit dans les zones de stockage. Un véhicule est directement acheminé en tampon de sortie : en cas de demande de véhicule, c'est ce véhicule (prêt à être utilisé) qui sera sélectionné (A.6).

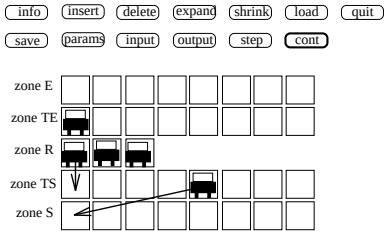


FIG. A.6 –: *Simulation du générateur de missions : deuxième étape*

Les véhicules rechargés sont placés en zone tampon de sortie, remplacés par les véhicules qui étaient en attente dans la zone tampon d'entrée (A.7).

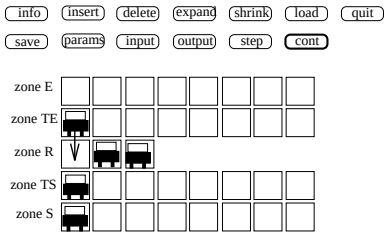


FIG. A.7 –: *Simulation du générateur de missions : troisième étape*

Enfin, après un certain temps, tous les véhicules sont passés en recharge et sont prêts à être utilisés (A.8).

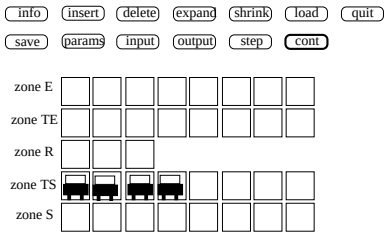


FIG. A.8 –: *Simulation du générateur de missions : quatrième étape*

5. Conclusion

Nous avons utilisé la logique floue afin de générer des missions pour des véhicules dans un cadre particulier, à savoir un parking. Tout comme cela avait été le cas en ce qui concerne l'élaboration de notre contrôleur d'exécution de mouvements flou, la logique floue a permis de coder de manière simple les différents

comportements attendus de la part du générateur de missions. Un tel générateur doit prendre en considération un certain nombre de critères dans son mécanisme de décision, ce qu'un système flou sait réaliser tout naturellement, grâce à un codage uniforme de ces critères.

Des travaux sur le même problème ont été réalisés récemment au laboratoire avec des approches neuronale et neuro-floue [27]. Des résultats similaires ont été obtenus, mais au prix d'un apprentissage conséquent, surtout en ce qui concerne l'approche neuronale pure. Par contre, le système flou a tout de suite donné des résultats satisfaisants, sans phase d'apprentissage.

Une amélioration du générateur de missions est envisageable au niveau de la génération de sous-buts intermédiaires pour les véhicules, permettant de ce fait d'alléger la tâche qui incombe au planificateur de trajectoires.

Annexe B

Base de règles du contrôleur flou

Nous donnons dans cette annexe un extrait de la base de règles utilisées par notre contrôleur flou présenté dans le chapitre § III. Cette base a été décomposée en un certain nombre de comportements, en précisant pour chacun les entrées et sorties considérées du contrôleur. Un effort a été fait pour respecter le séquencement du chapitre § III.

1. Suivi de trajectoire

1.1. Minimisation de la configuration relative $\|q_{ref}(t) - q_{cour}(t)\|$

Convergence en position

1. Entrées concernées du contrôleur flou
   Il s'agit des coordonnées polaires de la position de référence du véhicule dans le repère de celui-ci à sa position courante, à savoir :
   - angle_cap;
   - erreur_distance.
2. Sorties concernées du contrôleur flou
   - v_point :
     accélération longitudinale du robot;
   - phi_point :
     vitesse de braquage des roues directrices du robot.

3. Extrait de la base de règles

```
/* ======================================== */
/*          Convergence en position         */
/* ======================================== */

/* ======== Action sur le braquage ========== */
si (angle_cap est ANGLE_CAP_NG)
        alors (phi_point est PHI_POINT_NG);
...

si (angle_cap est ANGLE_CAP_PG)
        alors (phi_point est PHI_POINT_PG);

/* ========= Action sur la vitesse ========== */
si (erreur_distance est ERREUR_DISTANCE_NG)
        alors (v_point est V_POINT_NG);
...

si (erreur_distance est ERREUR_DISTANCE_PG)
        alors (v_point est V_POINT_PG);
```

remarque :

Dans les règles, les suffixes de labels de sous-ensembles flous PG, PP, ZR, NP, NG correspondent respectivement à Positif Grand, Positif Petit, ZéRo, Négatif Petit, Négatif Grand.

Convergence en orientation

1. Entrée concernée du contrôleur flou

   - delta_theta :
     différence entre θ_{ref} et θ_{cour}.

2. Sortie concernée du contrôleur flou

   - phi_point :
     vitesse de braquage des roues directrices du robot.

3. Extrait de la base de règles

```
/* ========================================== */
/*         Convergence en orientation         */
/* ========================================== */
si (delta_theta est DELTA_THETA_NG)
        alors (phi_point est PHI_POINT_NG);
...

si (delta_theta est DELTA_THETA_PG)
        alors (phi_point est PHI_POINT_PG);
```

1.2. Minimisation de la vitesse relative $|v_{ref} - v_{cour}|$

1. Entrée concernée du contrôleur flou

   – delta_v :
   différence entre v_{ref} et v_{cour}.

2. Sortie concernée du contrôleur flou

   – v_point :
   accélération longitudinale du robot.

3. Extrait de la base de règles

```
/* ========================================== */
/*           Convergence en vitesse           */
/* ========================================== */
si (delta_v est DELTA_V_NG)
        alors (v_point est V_POINT_NG);
...

si (delta_v est DELTA_V_PG)
        alors (v_point est V_POINT_PG);
```

2. Prise en compte de l'environnement local

2.1. Environnement statique

1. Entrées concernées du contrôleur flou
   Les entrées du système flou pour la prise en compte de l'environnement statique sont les zones d'intérêt. Bien que ces zones soient les mêmes pour le traitement des obstacles statiques et dynamiques, on les nomme de manière différente selon que l'on s'intéresse à une catégorie d'obstacle ou à une autre.

Les suffixes des noms de zones correspondent aux noms donnés dans la figure III.11 du chapitre § III, page 56.

- zone_stat_FA :
  zone d'intérêt avant;
- zone_stat_FR, zone_stat_SR, zone_stat_RR :
  zones d'intérêt latérales droites;
- zone_stat_FL, zone_stat_SL, zone_stat_RL :
  zones d'intérêt latérales gauches;
- zone_stat_RA :
  zone d'intérêt arrière.

2. Sorties concernées du contrôleur flou

   - v_point :
     accélération longitudinale du robot;
   - phi_point :
     vitesse de braquage des roues directrices du robot.

3. Extrait de la base de règles

```
/* ========================================== */
/*     Prise en compte de l'environnement     */
/*                  statique                  */
/* ========================================== */
si ((zone_stat_SR est OBST_STAT_TRES_PROCHE) et
    (zone_stat_RR est OBST_STAT_TRES_PROCHE))
          alors (phi_point est PHI_POINT_PG);

si ((zone_stat_SL est OBST_STAT_TRES_PROCHE) et
    (zone_stat_RL est OBST_STAT_TRES_PROCHE))
          alors (phi_point est PHI_POINT_NG);

si ((zone_stat_SL, OBST_STAT_PROCHE) et
    (zone_stat_RL, OBST_STAT_PROCHE))
          alors (phi_point, PHI_POINT_NP);

si (((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FR est OBST_STAT_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FL est OBST_STAT_PROCHE)))
          alors (v_point est V_POINT_NG);

...
```

2.2. Environnement dynamique

On peut penser que le traitement des obstacles dynamiques est le même que celui des obstacles statiques. Pour cette raison, nous retrouvons toutes les règles concernant l'environnement statique avec cependant la modification suivante: compte tenu des vitesses relatives des obstacles dynamiques, il est nécessaire de rallonger les distances de prise en compte des obstacles dynamiques (dans la limite de la portée des zones d'intérêt). Pour cette raison, nous avons renommé les variables linguistiques associées et leurs caractérisations floues, bien que ces zones soient strictement identiques.

2.3. Prise en compte de la vitesse du véhicule

La vitesse du véhicule est ajoutée en prémisse de certaines règles traitant de l'évitement d'obstacles. Il s'agit en fait des seules règles portant sur la détection d'obstacles dans les zones d'intérêt à l'avant et à l'arrière du véhicule i.e FL, FA, FR et RA (figure III.11 du chapitre § III, page 56). En effet, la vitesse du véhicule ne doit pas, à nos yeux, avoir d'incidence sur les sous-comportements associés aux zones latérales RL, SL, SR et RR. Ainsi la dernière règle concernant la prise en compte de l'environnement statique présentée précédemment s'écrit en fait comme suit :

```
si (((vitesse est FAIBLE) et
   (((zone_stat_FA est OBST_STAT_TRES_PROCHE) et
     (zone_stat_FR est OBST_STAT_TRES_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_TRES_PROCHE) et
     (zone_stat_FL est OBST_STAT_TRES_PROCHE))))
   ou
   ((vitesse est MOYENNE) et
   (((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FR est OBST_STAT_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FL est OBST_STAT_PROCHE))))
   ou
   ((vitesse est GRANDE) et
   (((zone_stat_FA est OBST_STAT_ASSEZ_PROCHE) et
     (zone_stat_FR est OBST_STAT_ASSEZ_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_ASSEZ_PROCHE) et
     (zone_stat_FL est OBST_STAT_ASSEZ_PROCHE)))))
          alors (v_point est V_POINT_NG);
```

Une autre solution consiste à réécrire la règle initiale sous la forme de trois règles :

```
si ((vitesse est FAIBLE) et
   (((zone_stat_FA est OBST_STAT_TRES_PROCHE) et
     (zone_stat_FR est OBST_STAT_TRES_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_TRES_PROCHE) et
     (zone_stat_FL est OBST_STAT_TRES_PROCHE))))
          alors (v_point est V_POINT_NG);

si ((vitesse est MOYENNE) et
   (((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FR est OBST_STAT_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_PROCHE) et
     (zone_stat_FL est OBST_STAT_PROCHE))))
          alors (v_point est V_POINT_NG);

si ((vitesse est GRANDE) et
   (((zone_stat_FA est OBST_STAT_ASSEZ_PROCHE) et
     (zone_stat_FR est OBST_STAT_ASSEZ_PROCHE))
    ou
    ((zone_stat_FA est OBST_STAT_ASSEZ_PROCHE) et
     (zone_stat_FL est OBST_STAT_ASSEZ_PROCHE))))
          alors (v_point est V_POINT_NG);
```

3. Cas d'échec et appel du planificateur

1. Entrée concernée du contrôleur flou

   - erreur_distance :
     distance relative du point de référence du robot dans la configuration désirée (exprimée dans le repère du robot).

2. Sorties concernées du contrôleur flou

   - appel_planif :
     indicateur de rappel du planificateur;
   - v_point :
     accélération longitudinale du robot.

3. Extrait de la base de règles

```
/* ======================================== */
/*            Appel au planificateur        */
/* ======================================== */
assertion(appel_planif, OFF);

si (erreur_distance est ERREUR_DISTANCE_TROP_GRANDE)
          alors (appel_planif est ON);

si (erreur_distance est ERREUR_DISTANCE_TROP_GRANDE)
          alors (v_point est V_POINT_NG);
```

remarque : Dans la base de connaissances, la première règle signifie que, par défaut, on n'aura pas d'appel au planificateur. Le poids associé à cette règle est très faible mais suffit pour envoyer en sortie la valeur désirée ("OFF"). La seconde règle est dotée d'un poids beaucoup plus fort : de ce fait, lorsqu'elle est activée, la valeur associée à l'appel du planificateur sera "ON". La dernière règle permet au véhicule de s'immobiliser.

4. Extension 1 : coopération passive entre véhicules

1. Entrées concernées du contrôleur flou
   Comme nous l'avons dit précédemment pour la prise en compte des obstacles statiques, les entrées seront les zones d'intérêt. Leurs noms différents sont justifiés par le fait que l'on ne s'intéresse ici qu'aux obstacles dynamiques.

   - zone_dyn_FA :
     zone d'intérêt avant;
   - zone_dyn_FR, zone_dyn_SR, zone_dyn_RR :
     zones d'intérêt latérales droites;
   - zone_dyn_FL, zone_dyn_SL, zone_dyn_RL :
     zones d'intérêt latérales gauches;
   - zone_dyn_RA :
     zone d'intérêt arrière.

2. Sorties concernées du contrôleur flou

   - v_point :
     accélération longitudinale du robot;

- phi_point :
  vitesse de braquage des roues directrices du robot.

3. Extrait de la base de règles

```
/* ========================================== */
/*     Prise en compte de l'environnement      */
/*                dynamique                    */
/* ========================================== */

si (((zone_dyn_FA est OBST_DYN_PROCHE) et
     (zone_dyn_FR est OBST_DYN_PROCHE))
     ou
     ((zone_dyn_FA est OBST_DYN_PROCHE) et
      (zone_dyn_FL est OBST_DYN_PROCHE)))
          alors (v_point est V_POINT_NG);

si (zone_dyn_FA est OBST_DYN_ASSEZ_PROCHE)
          alors (v_point est V_POINT_NP);

si ((zone_dyn_SR est OBST_DYN_TRES_PROCHE) ou
    (zone_dyn_FR est OBST_DYN_TRES_PROCHE))
          alors (phi_point est PHI_POINT_PG);

si (((zone_dyn_FA est OBST_DYN_LOIN) ou
     (zone_dyn_FA est OBST_DYN_ASSEZ_PROCHE) ou
     (zone_dyn_FR est OBST_DYN_LOIN) ou
     (zone_dyn_FR est OBST_DYN_ASSEZ_PROCHE))
    et
    ((zone_dyn_SL est OBST_DYN_LOIN) ou
     (zone_dyn_SL est OBST_DYN_NON_DETECTE)))
          alors (phi_point est PHI_POINT_PP);
...
```

5. Extension 2 : environnement structuré

1. Entrées concernées du contrôleur flou

   - contexte :
     indicateur de contexte;
   - zone_dyn_FA, zone_dyn_FR, zone_dyn_SR :
     zones d'intérêt à l'avant et à droite.

2. Sortie concernée du contrôleur flou

   – v_point :
   accélération longitudinale du robot.

3. Extrait de la base de règles

```
/* ========================================== */
/*        Prise en compte du contexte         */
/* ========================================== */
si ((contexte est CONTEXTE_CARREFOUR)
    et
    ((zone_dyn_SR est OBST_DYN_LOIN) ou
     (zone_dyn_SR est OBST_DYN_PROCHE)))
          alors (v_point est V_POINT_NG);

si ((contexte est CONTEXTE_CARREFOUR)
    et
    ((zone_dyn_FR est OBST_DYN_LOIN) ou
     (zone_dyn_FR est OBST_DYN_PROCHE)))
          alors (v_point est V_POINT_NG);

si ((contexte est CONTEXTE_CARREFOUR)
    et
    ((zone_dyn_FA est OBST_DYN_LOIN) ou
     (zone_dyn_FA est OBST_DYN_PROCHE)))
          alors (v_point est V_POINT_NG);
```

Références bibliographiques

[1] JR. C. Arkin. Motor Schema Based Navigation for a Mobile Robot. In *IEEE International Conference on Robotics and Automation*, pages 264–271, Raleigh, North Carolina, April 1987.

[2] C. Astraudo and J. J. Borelly. Simulation of Multiprocessor Robot Controllers. In *IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[3] B. Beaufrère and S. Zeghloul. Utilisation de la logique floue dans la navigation d'un robot mobile en milieu inconnu. In *Les Applications des Ensembles Flous - Quatrièmes Journées Nationales*, Lille, Décembre 1994.

[4] H. R. Berenji. Fuzzy Logic Controller. *An Introduction to Fuzzy Logic Applications in Intelligents Systems Kluwer academic publishers*, 1990.

[5] G. Berry. Real Time Programming: Special Purpose or General Purpose Languages. *IFIP Congress*, 1989.

[6] G. Berry and G. Gonthier. The Synchronous Esterel Programming Language: Design, Semantics, Implementation. 19(2):87–152, November 1992.

[7] B. Bouchon-Meunier. *La logique floue*. Collection "Que sais-je?", Presses universitaires de France, 1994.

[8] R. A. Brooks. Achieving Artificial Intelligence Trough Building Robots. Technical report, May 1986.

[9] R. A. Brooks. Elephants Don't Play Chess. *Journal of robotics and Autonomous Systems, Spring'90, North-Holland*, 1990.

[10] R. A. Brooks. Lunar Base Construction Robots. In *IEEE International Workshop on Intelligent Robots and Systems IROS '90*, pages 389–392, Tsuchiura, Ibaraki, Japan, July 1990.

[11] J. H. Connell. A behavior-based arm controller. *MIT AI Memo 1025*, June 1988.

[12] L. Cordewener. *Contrôle d'exécution de missions de robots mobiles à partir de tâches robot.* PhD thesis, Université de Technologie de Compiègne, Janvier 1995.

[13] G. B. Dantzig. *Linear programming and extensions.* Princetown University Press, 1963.

[14] P. Daviet and M. Parent. Platooning for Small Public Urban Vehicles. In *Fourth International Symposium on Experimental Robotics*, pages 213–218, Stanford, California, June 1995.

[15] E. Decamp and B. Amy. *Neurocalcul et réseaux d'automates.* EC2, 1988.

[16] A. M. Flynn, R. A. Brooks, and L. S. Tavrow. Twilight Zones and Cornerstones. Technical report, July 1989.

[17] Th. Fraichard. *Planification de mouvements pour mobile non-holonome en espace de travail dynamique.* PhD thesis, Institut National Polytechnique de Grenoble, Avril 1992.

[18] Th. Fraichard. Dynamic Trajectory Planning with Dynamic Constraints: a 'State-Time Space' Approach. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1393–1400, Yokohama, Japan, July 1993.

[19] Th. Fraichard and A. Scheuer. Car-Like Robots and Moving Obstacles. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 64–69, San Diego, CA (USA), May 1994.

[20] M-C. Fritsch and E. Wendling. Commande de processus : les atouts de la logique floue. *Technologies internationales*, Avril 1994.

[21] Ph. Garnier. Présentation de l'architecture matérielle utilisée par Sharp dans le cadre de Prolab2. Technical report, 1993.

[22] Ph. Garnier. Apprentissage d'une base de règles floues par la méthode du simplexe. In *Les Applications des Ensembles Flous - Quatrièmes Journées Nationales*, Lille, Décembre 1994.

[23] Ph. Garnier. Apprentissage d'une base de règles floues par la méthode du simplexe : application à un gestionnaire de missions. In *Secondes Rencontres Nationales des Jeunes Chercheurs en Intelligence Artificielle*, Marseille, Septembre 1994.

[24] Ph. Garnier, C. Novales, and C. Laugier. An Hybrid Motion Controller for a Real Car-Like Robot Evolving in a Multi-Vehicle Environment. In *Intelligent Vehicles'95*, Detroit, USA, September 1995.

[25] Ph. Garnier, C. Novales, D. Pallard, and G. Baille. Autonomy for Electric Cars. In *EVT'95*, Paris, November 1995.

[26] E. Gat, M. G. Slack, D. P. Miller, and R. J. Firby. Path Planning and Execution Monitoring for a Planetary Rover. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 20–25, Cincinnati, Ohio, May 1990.

[27] E. Gauthier. Gestion d'une flotte de véhicules autonomes à l'intérieur d'un parking haute-densité. Dea informatique, Institut National Polytechnique de Grenoble, France, juin 1995.

[28] P-Y. Glorennec. Un réseau "neuro-flou" évolutif. *Neuro-Nimes'91 Fourth International Workshop Neural Networks and their Applications*, 1990.

[29] P-Y. Glorennec. Application des algorithmes génétiques pour l'optimisation des fonctions d'appartenance d'un réseau neuro-flou. *Neuro-Nimes'92*, pages 219–226, 1992.

[30] S. K. Halgamuge and M. Glesner. Neurals Networks in Designing Fuzzy Systems for Real World Applications. In *Fuzzy Sets and Systems, 65(1):1-12*, 1994.

[31] D. Harel and A. Pnueli. On the Development of Reactive Systems. *Logic and Models of Concurrent Systems, NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, Springer Verlag*, 1985.

[32] 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, Décembre 1994.

[33] M. Hassoun and C. Laugier. Towards a Real-Time Architecture To Control An Autonomous Vehicle In Multi-Vehicle Environment. In *Proc. of the Int. Workshop on Intelligent Robots and Systems*, Yokohama, Japan, July 26-30 1993. IEEE.

[34] M. Hassoun, C. Laugier, N. Lefort, and D. Meizel. An Assistance System for Diagnosis and Monitoring of Driving Manoeuvres. *IMACS International Symposium on Signal Processing, Robotics And Neural Networks, Lille, France*, April 1994.

[35] M. Hassoun, C. Laugier, D. Ramamonjisoa, and N. Lefort. Towards Safe Driving in Traffic Situation by Using an Electronic Co-Pilot. In *Proc. of the IEEE Symposium on Intelligent Vehicles*, Tokyo, Japan, July 1993. IEEE.

[36] INRETS. Du carrefour intelligent à la ville intelligente. *Sphère du mécénat*, 1991.

[37] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A Stable Tracking Control Method for a Non-Holonomic Mobile Robot. *IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS'91, Osaka, Japan*, pages 1236–1241, November 1991.

[38] M. Kantrowitz, E. Horstkotte, and C. Joslyn. Frequently Asked Questions about Fuzzy Logic and Fuzzy Extert Systems. *Newsgroup comp.ai.fuzzy on Internet*, March 1995.

[39] C. L. Karr. Design of an adaptive fuzzy logic controler using a genetic algorithm. *Proceedings of the fourth international conference on genetic algorithms*, pages 450–457, 1991.

[40] O. Khatib. Real-time Obstacle avoidance for Manipulators and Mobile robots. *IEEE International Conference on Robotics and Automation*, pages 500–505, May 1985.

[41] N. Kull. *Simulation d'un véhicule routier autonome dans un environnement dynamique.* Mémoire cnam, CNAM, Lifia, Grenoble, Mars 1993.

[42] R. La, F. Guély, and P. Siarry. Apprentissage d'une base de règles floues par la méthode du recuit simulé. *Troisièmes Journées Nationales Les applications des ensembles flous Nimes*, 1993.

[43] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[44] C. Laugier, G. Baille, P. Garnier, M. Hassoun, and A. Scheuer. Praxitèle : Automatisation des Parkings Haute Densité et conduite automatique sur voie propre. Technical report, Février 1994.

[45] J. P. Laumond. *Feasible Trajectories for Mobile Robots with Kinematic and Environment Constraints.* Preprints of the International Conference on Intelligent Autonomous Systems, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1986.

[46] J. P. Laumond, A. de Saint Vincent, R. Alami, R. Chatila, and V. Pérébaskine. Supervision and Control of the AMR Intervention Robot. *Fifth International Conference on Advanced Robotics*, 2:1057–1062, June 1991.

[47] C. C. Lee. Fuzzy Logic in Control Systems : Fuzzy Logic Controller. 20(2):404–435, April 1990.

[48] E. H. Mamdani. Applications of fuzzy algorithms for simple dynamic plants. *Proc. IEEE*, 121(12):1585–1588, 1974.

[49] E. H. Mamdani. Advances in the Linguistic Synthesis of Fuzzy controllers. *International Journal of Man-Machine Studies*, 8(6):669–678, 1976.

[50] E. H. Mamdani. Applications of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis. *IEEE Trans. Computer*, C-26(12):1182–1191, 1977.

[51] E. H. Mamdani and S. Assilian. An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[52] E. Mazer and al. Act: A robot programming environment. In *IEEE International Conference on Robotics and Automation*, Sacramento, CA (USA), April 1991.

[53] E. Mazer and al. Act: un système de simulation robotique de nouvelle génération. In *MICAD*, Paris, Février 1992.

[54] D. Nauck and R. Kruse. A Neural Fuzzy Controller Learning by Fuzzy Error Propagation. In *NAFIPS'92*, Puerto Vallarta, Mexico, December 1992.

[55] C. Novales. *Pilotage par actions réflexes et navigation locale de robots mobiles rapides.* PhD thesis, Montpellier, Octobre 1994.

[56] C. Novales. La perception extéroceptive du PRAXICAR. Technical report, 1995.

[57] Prometheus Office. "Functions" or How to Achieve PROMETHEUS "Objectives". Technical report, Stuttgart (FRG), july 1989.

[58] M. Parent, F. Dumontet, P-Y. Texier, and F. Leurent. Design and Implementation of a Public Transportation System Based on Self-Service Electric Cars. In *IFAC/IFORS Congress*, Tianjin, China, August 1994.

[59] M. Parent, P-Y. Texier, and F. Leurent. Self-Service Electric Cars : A New Public Transport? In *OECD Seminar*, Omiya, Japan, June 1994.

[60] D. Payton. An Architecture for Reflexive Autonomous Vehicle Control. *IEEE Conference on Robotics and Automation, San Francisco, California*, pages 1838–1845, April 1986.

[61] F. G. Pin, H. Watanabe, J. Symon, and R. S. Pattay. Autonomous Navigation of a Mobile Robot Using Custom-Designed Qualitative Reasoning VLSI Chips and Boards. *Proceeding of the 1992 IEEE International Conference on Robotics and Automation Nice, France*, 1992.

[62] F. X. Pôtel, F. Richard, and P. Tournassoud. Design and Execution of Locomotions Plans. *Fifth International Conference on Advanced Robotics*, 2:1029–1032, June 1991.

[63] D. A. Reece and S. Shafer. An Overview of the PHAROS Traffic Simulator. *Proceeding of the 2nd International Conference on Road Safety Groningen, Netherlands*, September 1987.

[64] M. Sakarowitch. *Linear programming*. Springer Verlag, 1983.

[65] D. Simon, B. Espiau, E. Castillo, and K. Kapellos. Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues. In *IEEE Transactions on Control Systems Technology*, pages 213–229, December 1993.

[66] M. Sugeno, M. F. Griffin, and A. Bastian. Fuzzy Hierarchical Control of An Unmanned Helicopter. In *Fifth International Fuzzy Systems Association World Congress IFSA'93*, 1993.

[67] M. Sugeno and K. Murakami. An Experimental Study on Fuzzy Parking Control Using a Model Car. In *Industrial Applications of Fuzzy Control, Ed. Amsterdam*, pages 125–138, North-Holland, 1985.

[68] M. Sugeno and M. Nishida. Fuzzy Control of Model Car. In *Fuzzy Sets Systems*, volume 16, pages 103–113, 1985.

[69] T. Takagi and M. Sugeno. Derivation of Fuzzy Control Rules from Human Operator's Control Actions. In *IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 55–60, Marseille, France, July 1983.

[70] T. Takeuchi, Y. Nagai, and N. Enomoto. Fuzzy Control of a Mobile Robot for Obstacle Avoidance. *Information Sciences*, 45(2):231–248, 1988.

[71] Aleph Technologies. PRAXITELE : Spécification du logiciel ALEPH. Document Aleph Technologies, ref. 921F004/9308/1, Septembre 1993.

[72] T. Tsumura, N. Fujiwara, T. Shirakawa, and M. Hashimoto. An Experimental System for Automatic Guidance of Robot Vehicle, following the route stored in Memory. *Proceeding 11th International Symposium on Industrial Robots*, pages 187–193, October 1981.

[73] L. A. Zadeh. Outline of a New Approach to the Analysis of Complex Systems and Decision-Making Approach. *IEEE Transactions on Systems, Man, and Cybernetics SME-3(1)*, pages 28–45, January 1973.

[74] René Zapata. *"Quelques aspects topologiques de la Planification de Mouvements et des Actions Réflexes en Robotique Mobile"*. PhD thesis, Université Montpellier II, Juillet 1991.

[75] K. Zeghal and J. Ferber. CRAASH : A Coordinated Collision Avoidance System. *European Simulation Multiconference, Lyon, France*, June 1993.

Table des figures