Apprentissage et adaptation pour des ensembles de robots réactifs coopérants

Philippe Bruno Lucidarme

ACADEMIE DE MONTPELLIER

UNIVERSITE MONTPELLIER II

- SCIENCES ET TECHNIQUES DU LANGUEDOC -

THESE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE MONTPELLIER II

Discipline : **Génie Informatique, Automatique et Traitement du Signal**

Formation Doctorale : **Systèmes Automatiques et Microélectroniques**

Ecole Doctorale : **Information, Structures et Systèmes**

Présentée et soutenue publiquement

par

Philippe LUCIDARME

Le 7 novembre 2003

Titre :

Apprentissage et adaptation pour des ensembles de robots réactifs coopérants

JURY

Rachid ALAMI	Directeur de recherche LASS-CNRS	Rapporteur
Dominique DUHAUT	Professeur à l'Université de Lorient	Rapporteur
Jacques FERBER	Professeur à l'Université Montpellier II	Examinateur
Jean-Louis VERCHER	Directeur de recherche CNRS Aix-Marseille	Examinateur
René ZAPATA	Maître de conférence à l'Université Montpellier II	Directeur de thèse

à Alain …

Remerciements

Je voudrais, avant toute chose, exprimer ma plus grande reconnaissance à Alain Liégeois, qui était l'encadrant initial de cette thèse et qui nous a malheureusement quitté en mars 2003. Il a su diriger cette thèse avec diplomatie efficacité et rigueur tout en me laissant libre dans mes choix.

Les travaux présentés dans ce mémoire ont été effectués au sein du département de robotique du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM). Aussi, je tiens à remercier tout d'abord Messieurs Gaston Cambon et Michel Habib, directeurs successifs du LIRMM, pour m'avoir accueilli au sein du laboratoire durant ma thèse et mon DEA.

Je tiens aussi à remercier Jacques Ferber pour avoir accepté de présider mon jury de thèse, mais également pour avoir suivi mes travaux depuis leurs débuts. Les discussions que nous avons eu m'ont permis d'améliorer l'ensemble de mon travail et surtout de diriger mes recherches vers de meilleures voies.

Je tiens également à exprimer ma gratitude à Dominique Duhaut pour avoir accepter d'être rapporteur de ce travail. L'intérêt qu'il a porté à la lecture du manuscrit me touche particulièrement. Et même si les circonstances ne s'y prêtent pas actuellement, je ne perd pas espoir de pouvoir un jour collaborer avec lui.

Je remercie Rachid Alami d'avoir accepté de m'accorder une partie de son temps précieux pour être rapporteur de cette thèse. Ses questions et remarques ont été pour moi une grande source de réflexion. J'ai grandement apprécié sa gentillesse au cours de nos différentes rencontres, mais aussi le jour de ma soutenance.

Je voudrais aussi remercier Jean-Louis Vercher pour la collaboration que nous avons menée depuis quelques années. Il a su répondre avec pédagogie à mes nombreuses questions sur le vivant. J'espère que l'absence d'Alain n'entravera en rien la collaboration entre neuro-scientifiques et roboticiens, et qu'il restera toujours aussi généreux en explications et anecdotes sur ses connaissances du vivant.

Je voudrais remercier René Zapata pour avoir accepté de prendre la direction de cette thèse après la mort d'Alain. Il a su me conseiller et me guider avec sérieux, efficacité et humour.

Que Etienne Dombre reçoive toute ma reconnaissance pour ses remarques pertinentes et constructives, ses conseils avisés et son soutien. Il m'a fallu du temps pour découvrir que derrière ces moustaches et ce visage rigoureux, parfois sévère, il y a un homme au grand cœur.

J'adresse toute ma sympathie aux permanents du département robotique, avec une mention particulière à certain d'entre eux : François Pierrot (qui a réussit à nous convaincre de partir de l'autre côté de la terre), André Crosnier (pour sa bonne humeur contagieuse), Philippe Poignet (pour la balade en voilier …) et Oliver Strauss (pour les bonnes adresses de petits restaurants romantiques).

Je remercie Philippe Rongier pour m'avoir initié au monde de la recherche et à celui des systèmes multi-agents. Egalement Geovani Borges, auteur du célèbre "Pourquoi manger, pourquoi dormir".

C'est parce que la coutume m'y oblige sinon je m'en serais passé : je remercie l'ensemble des thésards que j'ai du supporter pendant ces quatre longues années. Je commencerais par le trio des cinq mousquetaires de l'amour : Sébastien, Benoit, Micaël et Fabien pour avoir partagé tant de moments intenses, qu'ils aient été festifs, drôles, culinaires, romantiques ou … surtout festifs ! Mais aussi tous les autres, dans le désordre : Gilles (pour ta bonne humeur et ta gentillesse), Pierre (… qui roule n'amasse pas mousse, moi aussi j'ai mangé un clown à midi ☺), Mezziane (bijour jy sui li nouvo pilote di l'avion), les Wany-roboticiens (pour le matériel et les conseils) et bien sûr tout ceux que j'oublie, si vous n'êtes pas dans cette liste, mettez votre nom ici avec un commentaire :

Mon nom : _____

Le commentaire : _____

Bien sur, il me reste à remercier toute la bande de l'ancr et des Porteurs-Libres. Tout cela est de votre faute : si un beau jour d'hiver 1999, vous ne m'aviez pas suivi dans cette folle aventure qu'est la coupe de France de robotique, je n'en serai pas là aujourd'hui. Même si nous ne sommes jamais rentrés vainqueur de la Ferté-Bernard, notre victoire est ailleurs.

Comment pourrais-je oublier celle qui a accompagné mon cœur ces huit derniers mois, celle qui a supporté la rédaction, la soutenance, le départ au Japon, qui sait me réconforter quand rien ne va plus. Celle qui me fait dire que les neurosciences sont passionnantes et qui m'accorde des cours particuliers de biologie … Mille baisés, Laëtitia, pour tous ces merveilleux moments que tu m'as offerts pendant cette période parfois difficile.

Lasts but not least … Pour terminer ces remerciements, il reste deux personnes que je souhaite remercier avec une très grande sincérité : mes parents.

Maman : même si la vie n'a pas toujours été facile dans le foyer familial, tu as su parfois te sacrifier pour nous offrir une vie meilleure. Et grâce à toi, je profite aujourd'hui du meilleur de la vie. J'aimerais pouvoir te rendre tout l'amour et la dévotion que tu nous as offert, mais une vie entière n'y suffirait pas. J'espère au moins que cette thèse y contribuera en partie.

Papa : pour avoir écrit une thèse traitant de l'apprentissage, je suis en mesure de vous expliquer une de mes convictions : chacun de nous naît avec un patrimoine génétique qui va partiellement régir notre apparence physique mais aussi une partie de notre caractère et de notre comportement. Mais cette structure de base est plastique, c'est à dire qu'elle va évoluer au fils du temps. Cette évolution va dépendre de stimulations extérieures et de leurs associations à des récompenses ou à des expériences traumatisantes. Pour résumer, rien n'est joué à la naissance et chacun dispose de sa vie. Un vrai père doit offrir à la fois le patrimoine génétique et l'éducation qui va avec. Mais vous, Guy, vous avez fait encore mieux : vous avez accepté le rôle ingrat de l'éducateur sans pour autant pouvoir revendiquer une quelconque paternité, d'autant plus que le patrimoine de base n'était pas merveilleux. Alors sachez Guy, que je vous ai toujours considéré comme mon père et qu'il vous revient tous les honneurs et la fierté qu'un père peut avoir quand il voit sont fils devenir docteur.

Table des matières

Table des figures, tableaux et algorithmes

CHAPITRE

1

1. Introduction générale

Quand j'avais une dizaine d'années et que cette perpétuelle question m'était adressée, un dialogue similaire s'instaurait à chaque fois :

- "Qu'est ce que tu veux faire plus tard ?
- Des robots.
- Ca c'est l'avenir …"

Mais ces robots, qui semblait présager tant de bonnes choses pour l'avenir, font-ils partie de notre quotidien ? Nous soulagent-ils des tâches fastidieuses et répétitives ? Sont-ils nos fidèles compagnons prêts à se donner corps et âmes (si tant est qu'ils en aient une) afin d'achever la mission pour laquelle ils ont été programmés ?

Même si je ne croise pas encore de robots humanoïdes dans les rues quand je fais mes courses, je dirais que la robotique a réussi une partie du défi qui lui était lancé au milieu du siècle passé. En premier lieu dans l'industrie : les machines automatisées, omniprésentes sur les chaînes de fabrication, assistent (selon le patron), remplacent (selon les délégués syndicaux) les ouvriers dans des tâches répétitives et fastidieuses. Plus important encore, le personnel a été partiellement écarté de certaines tâches pouvant nuire à la santé, comme par exemple la manipulation de produits toxiques.

Les robots se sont également immiscés dans la vie de tous les jours: ils n'ont pas la forme que leur avaient prêté les auteurs de sciences-fiction quelques années auparavant, mais ils remplissent fidèlement la tâche pour laquelle ils ont été conçus. Les lave-vaisselles, les lave-linge et d'autre machines nous assistent fidèlement dans les tâches répétitives de notre quotidien.

Même s'ils se sont discrètement intégrés dans nos vie, une grande partie du défi n'est pas encore relevée. Ces machines, qui existent depuis plusieurs années maintenant, ne disposent que d'un très faible pouvoir décisionnel. Elles sont programmées pour réaliser la même tâche successivement sans pouvoir s'adapter aux événements pour lesquels elles n'ont pas été programmées. Des problèmes, insoupçonnés à l'époque, sont apparus au cours des années et des avancées technologiques. Par exemple, localiser un robot dans un environnement inconnu, reconnaître des objets caractéristiques dans une image, piloter des processus régis par des systèmes d'équations complexes restent des problèmes ouverts.

J'ai le sentiment que le changement de millénaire a été une période cruciale pour la robotique. Plusieurs événements spectaculaires et médiatisés nous donnent un avant-goût de l'avenir :

- Le 4 juillet 1997, le robot Pathfinder de la NASA, pose ses roues là où aucun homme n'a jamais mis le pied. Ce robot réalisera avec brio une mission de trois mois sur Mars. Pathfinder reste aujourd'hui encore un des robots mobiles les plus célèbres au monde.
- Après l'effroyable catastrophe du 11 septembre 2001, des robots téléopérés se sont engouffrés dans les décombres du World Trade Center avec l'espoir de retrouver des survivants.
- Le 7 septembre 2001, une opération chirurgicale consistant à enlever une vésicule biliaire a été réalisée. Le chirurgien se trouvait à New-York et le patient à Paris. Cette opération a été possible grâce à un robot qui a pu être téléopéré à travers l'atlantique. Voilà précisément un magnifique exemple où la machine assiste le geste humain, mais où en aucun cas elle ne remplace les décisions du chirurgien.
- En mai 1997, l'ordinateur Deep-Blue remporte un match contre l'un des meilleurs joueurs d'échecs que l'humanité ait connu : Garri Kasparov. Cette expérience montre les capacités de la machine à dépasser les facultés de ses créateurs et de l'homme sur un problème qui est souvent considéré comme l'incarnation même du jeu intelligent.
- Le robot chien AIBO a été vendu à plus de 100 000 exemplaires à travers le monde par la société Sony. L'engouement mondial et commercial pour la robotique ne cesse de croître: de plus en plus de robots sont disponibles dans le commerce comme par exemple pour tondre la pelouse ou encore nettoyer le sol.
- D'un aspect plus ludique, depuis quelques années, les concours de robotique se multiplient avec comme principal événement la coupe du monde de robotique RoboCup qui déchaîne les passions.
- Enfin, un privilège qui était jusqu'alors réservé aux dieux est en passe de devenir accessible : créer une créature à son image. Les sociétés Sony, Honda et l'AIST ont présenté leurs robots humanoïdes respectivement appelés SDR, ASIMO et HRP-2.

Tous ces événements, pris séparément, restent des faits divers, qui feront au mieux la première page du journal. Mais pris dans leur globalité, ils montrent bien que la robotique traverse une période charnière pour entamer une nouvelle ère.

La robotique est une matière pluridisciplinaire au carrefour de la mécanique, de l'électronique, de l'informatique, des mathématiques mais aussi d'autres disciplines comme les neurosciences ou la psychologie, sans parler du domaine d'application qui s'étend de la microchirurgie à l'exploration spatiale. Les avancées technologiques dans toutes ces disciplines ouvrent aujourd'hui aux chercheurs de nouvelles perspectives et des centaines de voies de recherche à explorer.

Parmi toutes les applications de la robotique, chacun s'accorde à dire que l'accent doit être mis sur le respect de la vie humaine et la préservation de l'environnement. Par exemple, si des machines avaient remplacé les hommes lors de la catastrophe de Tchernobyl, bon nombre de vies auraient été épargnées. Suite à un attentat ou un tremblement de terre, les recherches de survivants seraient accélérées si des flottes de robots pouvaient être envoyées dans les décombres. L'exploration des coques de pétroliers et le pompage sous-marin des cuves permettrait d'éviter les marées noires, nocives pour la santé et l'environnement. Ces quelques exemples n'ont pas été pris au hasard, ils se situent tous en milieu hostile : c'est-à-dire des zones difficilement accessibles voire dangereuses pour l'homme. C'est pourquoi y envoyer des robots afin d'y réaliser une mission permettrait d'épargner des vies. Seulement, il n'est pas toujours possible de téléopérer ces machines. Par exemple pour l'exploration martienne, les transmissions mettent environ 30 minutes pour atteindre la planète rouge. De plus, le retour d'information, souvent incomplet, ne permet pas à l'opérateur de disposer de toutes les informations nécessaires à la prise de décision. C'est pourquoi ces machines doivent disposer d'un maximum d'autonomie. Elles doivent être capables de s'adapter à des environnements inconnus, des situations imprévues et des pannes.

Les techniques de programmation des robots les plus répandues sont basées sur le tout-programmé. C'est-à-dire que l'ensemble des situations envisageables a été programmé, soit de manière discrète, soit de manière continue, soit de manière mixte. Mais il est évident que ces méthodes ne permettent pas de résoudre tous les problèmes. Pour des missions complexes d'exploration ou de sauvetage, l'ensemble des cas ne peut être envisagé et programmé. De nouvelles méthodes qui se présentent comme complémentaires au tout-programmé sont apparues depuis une vingtaine d'années. Nos travaux vont s'intéresser principalement à deux d'entre elles : les systèmes multi-agents et l'apprentissage.

Dans ce manuscrit, nous allons nous intéresser à l'auto-apprentissage. Au lieu de programmer un robot pour qu'il effectue une mission, nous allons le laisser apprendre seul sa propre stratégie. Dès lors qu'il apparaît concevable qu'une machine soit capable de modifier son propre comportement, son propre programme, il ne semble plus y a voir de limite à son développement. Pourtant, voici les deux principaux freins :

- Les données mémorisées sont trop importantes. En effet, toute méthode d'apprentissage nécessite d'enregistrer des données. Malgré l'accroissement des mémoires informatiques, certains problèmes ne sont pas solvables, faute de place. Il existe des méthodes, notamment le Q-learning, qui permettent de trouver la solution optimale pour un problème déterministe. Mais ces méthodes demandent d'enregistrer des données statistiques importantes et ne peuvent pas toujours résoudre ce type de problème.

- Les temps d'apprentissage reste le frein principal à l'émancipation des machines. La relation liant les temps d'apprentissage à la complexité du problème sont exponentiels. La convergence du Q-learning a été prouvée pour un temps d'apprentissage infini. C'est pourquoi il est inconcevable de faire apprendre à une machine en 10 ans ce que l'humanité à appris durant des millions d'années.

Ces deux limites nous montrent qu'il n'est pas envisageable de tenter de procéder directement à la résolution de problèmes complexes en appliquant les méthodes d'apprentissage tel quel. C'est pour cette raison que l'apprentissage doit être découpé et progressif. L'approche envisagée dans ces travaux est basée sur l'approche dite " subsomption " présentée dans le deuxième chapitre. Cette méthode consiste à décomposer

une mission complexe en plusieurs missions ou tâches de plus bas niveaux s'exécutant en parallèle. Ces travaux s'intéressent à l'apprentissage d'actions réflexes ou encore de comportements réactifs où les réactions sont les effets immédiats d'une perception. Ces comportements ne sont jamais le résultat d'un raisonnement de haut niveau mettant en jeu des données complexes du type carte de l'environnement ou représentation interne des agents.

La structure de ce manuscrit est la suivante :

Le deuxième chapitre se concentre sur la présentation des systèmes multi-agents. Les travaux antérieurs les plus pertinents sont décrits, justifiant l'utilisation d'architectures distribuées dans des situations où les systèmes supervisés serait inefficaces.

Le troisième chapitre s'intéresse aux méthodes d'apprentissage. Les principales techniques sont présentées et détaillées : réseaux de neurones, apprentissage par renforcement et algorithmes évolutionnistes.

Le quatrième chapitre est consacré à la description de notre plate-forme expérimentale. Afin de valider les méthodes proposées dans des circonstances réelles, nous avons construit une plate-forme composée de quatre robots mobiles, un manipulateur mobile et un système de vision stéréoscopique.

Le cinquième chapitre est dédié à l'étude de l'apprentissage de comportements réactifs sur des populations de robots identiques. Cette étude s'oriente vers l'application des algorithmes évolutionnistes aux systèmes multi-agents. Des travaux antérieurs sont présentés et nous décrirons une évolution distribuée de ces méthodes.

Le sixième chapitre traite de l'apprentissage de systèmes hétérogènes, c'est-à-dire dans le cas de robots disposant de capacités différentes. Ce chapitre est composé de deux parties : une méthode d'apprentissage basée sur le recuit simulé et une seconde partie basée sur l'apprentissage par renforcement.

Enfin, nous conclurons sur l'ensemble de ces travaux dans le dernier chapitre.

CHAPITRE

2

2. Les systèmes multi-agents

2.1 Introduction

Dans ce document, nous allons nous préoccuper d'un type de comportements particuliers: la coopération. Un exemple classique est celui d'une population de fourmis. L'une de ces dernières, prise seule, dispose d'une " intelligence réduite ". Mais l'ensemble de la population est capable de collaborer afin de réaliser des tâches complexes. Il semble toutefois important de préciser que notre but n'est pas de copier le plus fidèlement possible le milieu animal, mais d'en tirer une source d'inspiration.

Cette inspiration animale pourrait bien être appliquée à un grand nombre de tâches, en milieu hostile par exemple. Les principales recherches sur les systèmes multi-agents s'orientent vers des travaux dans l'aérospatiale, le milieu sous-marin, le nucléaire ou encore pour des tâches de sauvetage en milieu hostile. L'approche multi-agents présente plusieurs avantages pour ce type de tâches. Si une panne survient sur un robot ne disposant que d'un seul système de commande, de vision et de préhension, c'est toute la mission qui échoue. Au

contraire, avec une flotte de robots, il est envisageable d'envoyer plusieurs agents, plus simples, plus légers et redondants. La flotte peut en effet disposer de plusieurs systèmes de vision et d'action sur l'environnement. En cas de panne totale ou partielle d'un agent, le système peut alors continuer à fonctionner en mode dégradé. Il faut bien sûr pour cela que le système ait été conçu de façon à pouvoir s'adapter aux pannes. De plus, pour certaines tâches, notamment dans la recherche de survivants, les zones à explorer sont parfois tellement vastes que l'emploi d'une flotte de robots est imposé par la nature même de la mission.

2.2 Définition

" Celui qui agit, produit un effet " est la définition de l'agent selon le dictionnaire Larousse. Cette définition, même si elle correspond partiellement au sens des multi-agents reste incomplète. Elle a été reformulée par J. Ferber [Ferber 95].

On appelle agent, une entité physique ou virtuelle

- qui est capable d'agir dans un environnement,
- qui peut communiquer directement avec d'autres agents,
- qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- qui possède des ressources propres,
- qui est capable de percevoir son environnement,
- qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,
- dont le comportement tend à satisfaire ses propres objectifs, en tenant compte des ressources et des compétences dont elle dispose et en fonction de sa perception de ses représentations et des communications qu'elle reçoit.

Pour illustrer cette définition, donnons ici quelques exemples. Peut-être considéré comme un agent : un robot, une partie d'un robot (le robot pourrait alors être vu comme un système multi-agents), un programme informatique ou une partie d'un programme etc. Dans la suite du document, nous considèrerons notre robot comme un agent et nous étudierons donc l'évolution et la coopération au sein d'un système multi-robots.

On distingue les systèmes **homogènes** des systèmes **hétérogènes**. La différence réside dans la structure des agents : dans un système homogène, les agents sont tous identiques et disposent, par conséquent, des mêmes moyens de perception et d'action sur l'environnement. Dans un système hétérogène, les agents sont différents et parfois spécialisés dans certaines tâches. La réalisation de certains objectifs peut nécessiter la collaboration d'agents aux compétences différentes et spécifiques. Par exemple, pour une tâche d'exploration de l'environnement, une grande population d'agents explorateurs identiques permettrait de diminuer le temps nécessaire à la visite complète de l'environnement. Il s'agit là d'un système homogène. Pour une tâche de collecte d'échantillons en milieu inconnu, une population composée d'agents explorateurs, d'agents manipulateurs et d'agents transporteurs serait bien approprié. Il suffit de regarder un chantier moderne pour s'apercevoir que tout est régi comme un système multi-agents hétérogène. Chaque engin a sa propre tâche et ses propres spécifications.

2.3 Les tâches génériques

Les premières recherches sur la coopération multi-agents portaient sur l'étude du milieu animal, notamment celui des insectes. Ces études ont rapidement montré que chaque insecte n'avait pas ou peu conscience de son rôle dans la tâche globale réalisée par l'ensemble de l'essaim. Ces actions étaient uniquement guidées par des changements de comportement. Ce phénomène a rapidement intéressé les roboticiens qui se sont inspirés de ce milieu afin de réaliser des flottes de robots coopérants. Les premiers articles concernant les systèmes multi-agents remontent au début des années 80 pour la robotique. Les premiers objectifs furent de définir des méthodes simples et fiables afin de réaliser des tâches courantes [Drogoul 92] et [Arkin 92]. Ces tâches, généralement inspirées du milieu animal (principalement les fourmis), peuvent se décomposer en trois grandes familles de tâches génériques qui sont la mine, la consommation et l'exploration (Sur la Figure 1 les gros ronds noirs représentent les obstacles. Les attracteurs sont symbolisés par les petits. Les traits gras ou pointillés représentent le parcours des robots (deux robots et sept attracteurs)).

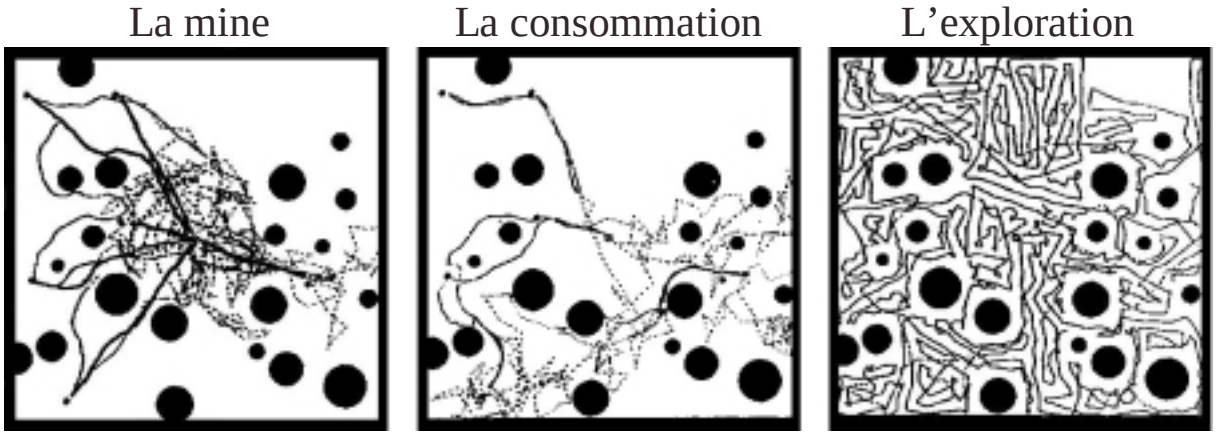


Figure 1 : Simulation pour chacune des tâches (D'après [Arkin 92])

La mine consiste à trouver une ou plusieurs mine(s) et à rapporter le minerai à la base. C'est l'une des applications les plus courantes. Les principaux intérêts de cette tâche résident dans le fait que l'environnement n'est pas connu à l'avance, que les robots sont obligés de faire des aller-retours entre la base et la source. Ces points en particulier font que la coopération prend parfois toute son ampleur.

La consommation similaire à la précédente, la consommation est une tâche qui se différencie de la mine dans le fait que l'action est effectuée sur place. Dans la mine, le minerai est ramassé, puis ramené à la base alors que pour la consommation, la tâche est réalisée sur place comme par exemple de la peinture ou de l'assemblage. C'est parfois cette action qui va nécessiter la coopération entre les robots.

L'exploration consiste à parcourir la plus grande surface possible de l'environnement. Dans bien des cas, plus les agents sont nombreux, plus la tâche est réalisée rapidement. C'est justement sur ce point que la coopération devient importante, le système global doit être bien pensé pour éviter de voir un agent explorer une zone qui l'a déjà été. C'est également pour cela que la communication est l'un des éléments essentiels des systèmes multi-agents.

2.4 L'approche réactive

2.4.1 Introduction

Les systèmes réactifs et cognitifs sont deux familles opposées de systèmes multi-agents. Dans un système réactif, l'agent perçoit localement son environnement (et éventuellement son état interne) et déduit immédiatement les actions à réaliser en se basant uniquement sur cette source d'information. Ce principe se base sur l'action réflexe [Zapata 92]. Les trois exemples qui suivent (Silly Robot, Tom Thumb et Chain-Making Robot) sont tous basés sur des architectures réactives. Ces trois exemples ont été volontairement décrits et explicités afin de montrer qu'un système multi-agents réactif peut avoir des performances très honorables et que des modifications mineures (ajout de communications par exemple) peuvent augmenter de façon significative les performances du système.

2.4.2 Silly Robot

Dans un article relatif à la coopération [Steels 89], Steels décrit des robots ramassant des échantillons en milieu inconnu. La structure décrite ci-après a été conçue afin de réaliser une tâche de type " la mine ". Les robots peuvent détecter la base grâce à un signal émis par celle-ci. A condition d'être suffisamment près, il est possible de sentir les mines (les attracteurs) grâce aux stimuli. Les robots peuvent également ramasser les échantillons et les déposer.

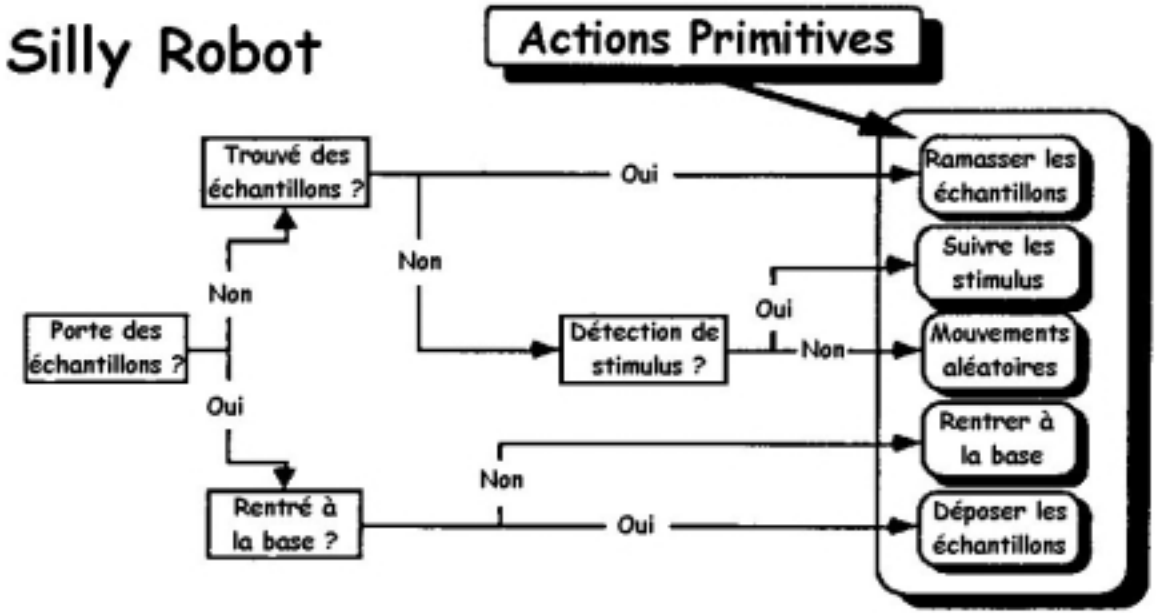


Figure 2 : Diagramme des " Silly Robot " (D'après [Drogoul 92])

Le plus simple des comportements est décrit sur le diagramme de la Figure 2. Le robot se déplace de façon aléatoire jusqu'à ce qu'il sente une mine. Il se dirige vers celle-ci afin d'y ramasser un échantillon, puis il retourne à la base pour le déposer avant de revenir dans une phase de déplacement aléatoire. Malgré leur manque d'organisation sociale, ces robots à l'efficacité réduite vont servir de base aux prochaines générations d'agents. Et nous allons voir que des modifications mineures à l'échelle du comportement individuel peuvent bouleverser l'organisation globale.

2.4.3 Tom Thumb

La première modification apportée à ce " Silly Robot " a été inspirée par la nouvelle de Charles Perrault : Le Petit Poucet (Tom Thumb). En effet, l'idée est de déposer des signes sur le chemin du retour à la base lorsqu'une mine a été découverte. Cette stratégie permet aux autres robots de suivre ces marques afin de se rendre plus rapidement aux attracteurs. De plus, lorsqu'un agent vient de déposer un échantillon, il retourne directement en prélever un autre sans passer dans une phase de déplacement aléatoire.

Le principal inconvénient de ce principe est qu'une mine vide continue d'attirer les robots puisque les marques ne sont pas ramassées. Pour éviter ce phénomène, nous allons permettre aux robots d'effacer les marques. Lorsqu'il a détecté une mine, il laisse une marque sur son chemin qu'il ramassera lors de son retour à la base. Les résultats de simulation ne sont pas des plus satisfaisants. En effet, supprimer complètement la marque cache un grand nombre d'informations aux autres robots. D'un autre côté, lorsqu'une mine est vide, ils ne sont plus attirés par celle-ci.

Un compromis des deux méthodes précédentes a été proposé par Steels [Steels 91]. Le concept est de pouvoir effacer les marques, mais lentement. Les robots déposent donc 2 marques sur le chemin de la base et en ramassent une lorsqu'ils se dirigent vers la mine. Les résultats de la simulation réalisée par A. Drogoul et J. Ferber montrent une efficacité et une stabilité bien meilleure. Seulement, lorsque le nombre de robots est supérieur à 85, les performances diminuent. Par exemple, pour une population de 100 robots, le nombre de cycles nécessaires pour ramasser la totalité des échantillons est équivalent à celui d'une population de 25. Ce phénomène d'embouteillage est constaté lorsque le nombre d'agents est élevé et est dû à une forte concentration autour de la base et des sources de minerai.

2.4.4 Chain-Making Robot

Après avoir présenté et comparé les deux méthodes précédentes A. Drogoul et J. Ferber proposent un nouveau type de comportement nommé Chain-Making. Celui-ci est inspiré des dockers, qui plutôt que de s'amasser autour de la porte du bateau, forment une chaîne allant du bateau au point de déchargement. Pour réaliser ce comportement il est indispensable d'ajouter de nouvelles fonctionnalités à nos agents : la possibilité de détecter un échantillon transporté par un autre agent et de lui prendre. Par exemple, une lampe disposée au-dessus des robots, lorsqu'elle est allumée, signifie qu'il est en train de regagner la base tout en transportant un échantillon. Cette émission de l'état interne pourra être détectée par un robot se trouvant dans une phase de déplacement aléatoire et lui permettra d'adopter le même comportement que s'il venait de découvrir une mine. Il prélèvera l'échantillon transporté avant de se diriger vers la base. De même, s'il rencontre un robot vide sur son chemin, il abandonnera sa cargaison à son successeur pour revenir dans un état d'errance. Le diagramme de ce type de comportement est présenté figure 3.

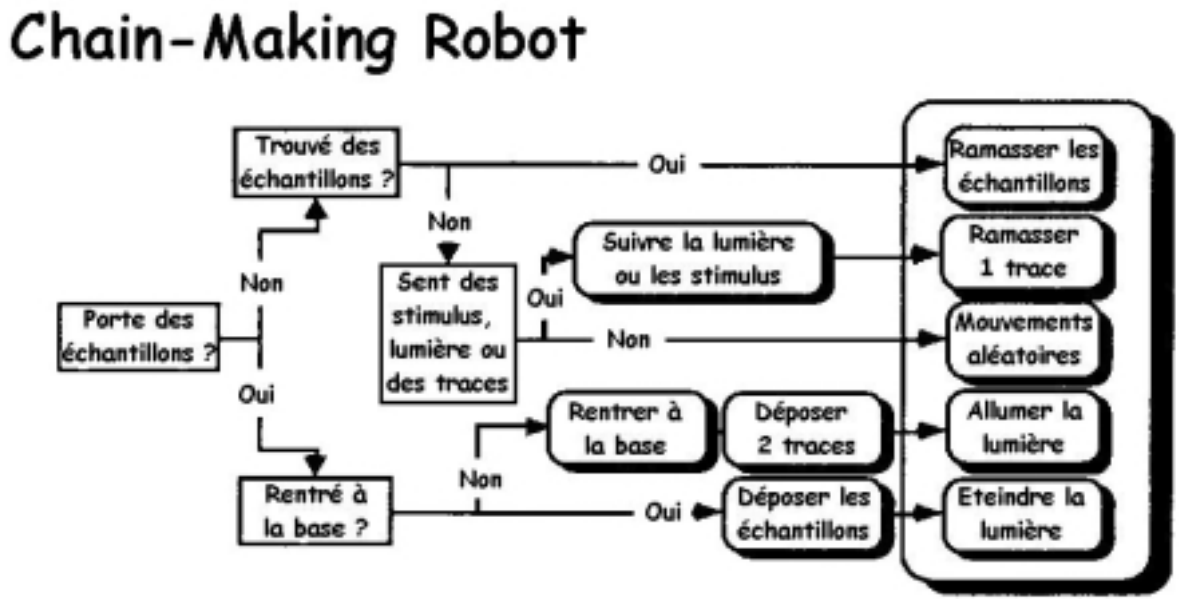


Figure 3 : Diagramme des Chain-Making (D'après [Drogoul 92])

La méthode des chain-making donne de meilleurs résultats que les deux précédentes quel que soit le nombre de robots. De plus, dans la simulation réalisée par A. Drogoul et J. Ferber, même pour un nombre élevé de robots, le phénomène d'embouteillage n'apparaît plus et la stabilité est atteinte. De manière générale, un minimum de modifications au niveau comportemental permet un accroissement des performances globales.

2.4.5 Systèmes réactifs et cognitifs

Depuis le début de la robotique et ce jusqu'aux années 90, la majeure partie des systèmes robotisés étaient basée sur la même architecture (Figure 4). Les informations sur l'environnement sont stockées, analysées et modélisées avant que l'agent puisse prendre une décision. Ce type d'architecture est dit cognitif.

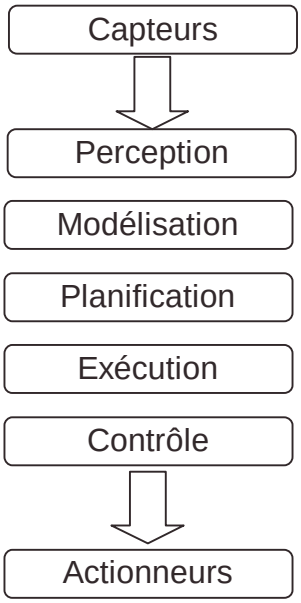


Figure 4 : Architecture traditionnelle de décomposition du programme de contrôle du robot en différents modules de fonctionnement (D'après [Brooks 86])

En 1986, Rodney A. Brooks [Brooks 86] propose une approche différente qui reste aujourd'hui encore une référence dans le domaine des systèmes multi-agents. Cette architecture, appelée " subsomption ", consiste à paralléliser les tâches. La Figure 5 décrit les différentes couches comportementales d'un agent.

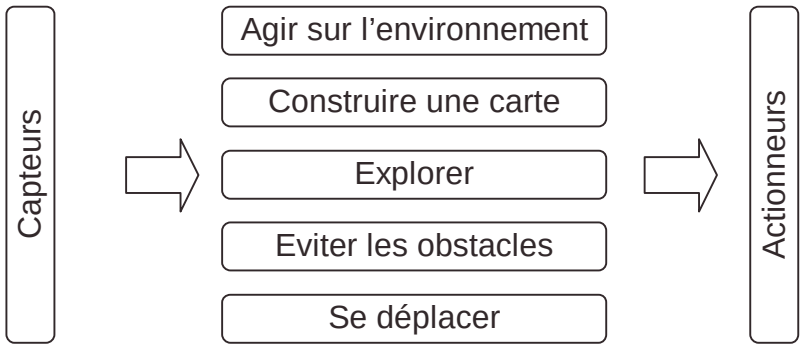


Figure 5 : Décomposition basée sur le comportement d'accomplissement de tâches (D'après [Brooks 86])

Chacune de ces couches relie les capteurs aux actionneurs et permet un comportement particulier ou une compétence spécifique, comme la locomotion, l'évitement d'obstacles ou la saisie d'objets. Ce type d'architecture permet notamment la décomposition d'une tâche complexe en plusieurs comportements réactifs. Ce type d'approche vise aussi à accroître la fiabilité du système. Dans l'architecture traditionnelle, si une panne survient sur l'un des modules, alors la panne se généralise à l'ensemble du système, chaque module étant essentiel au fonctionnement de l'ensemble. Dans l'architecture proposée par Rodney A. Brooks, même après la perte d'un module, le système peut continuer à fonctionner en mode dégradé, en inhibant ce module par exemple.

L'architecture de forçage / inhibition et celle à base de combinaison vectorielle sont deux exemples d'architectures réactives de type " subsomption "

Architecture de forçage / inhibition. La Figure 6 représente un module d'une telle architecture. Ses entrées peuvent être forcées pendant une certaine durée, de même les sorties peuvent être inhibées pendant un laps de temps. Des comportements plus complexes peuvent ainsi être construits en imbriquant les modules de base. La structure globale permettra d'inhiber les comportements non prioritaires et de forcer les fonctions vitales. Par exemple le signal d'entrée du module dédié à la locomotion peut être forcé pour le remplacer par un signal d'évitement d'obstacle afin d'empêcher une collision.

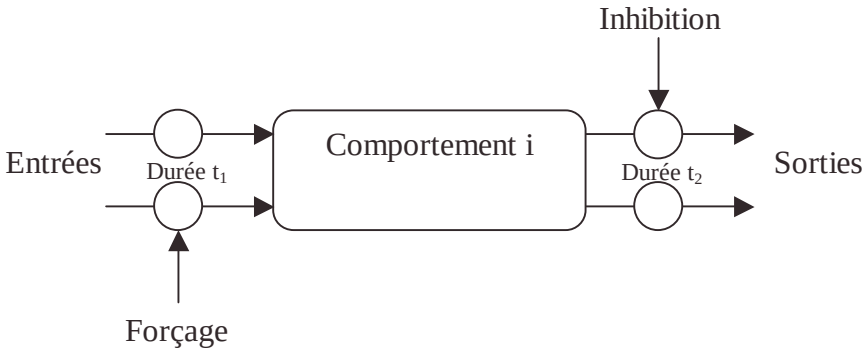


Figure 6 : Un module de l'architecture forçage / inhibition

Architecture à base de combinaisons vectorielles. Inhiber des comportements ne permet pas toujours d'obtenir un résultat assez fin. Il existe ainsi des architectures basées sur une combinaison vectorielle des sorties de chaque comportement. Un vecteur de sortie est constitué de l'ensemble des consignes calculées par un module. La consigne appliquée aux moteurs sera alors une combinaison vectorielle de l'ensemble des vecteurs de sortie. La Figure 7 montre un exemple d'architecture à base de combinaison vectorielle. Le vecteur de sortie est constitué de deux consignes Mg et Md, le vecteur effectivement appliqué sur les actionneurs sera une somme des vecteurs de sortie.

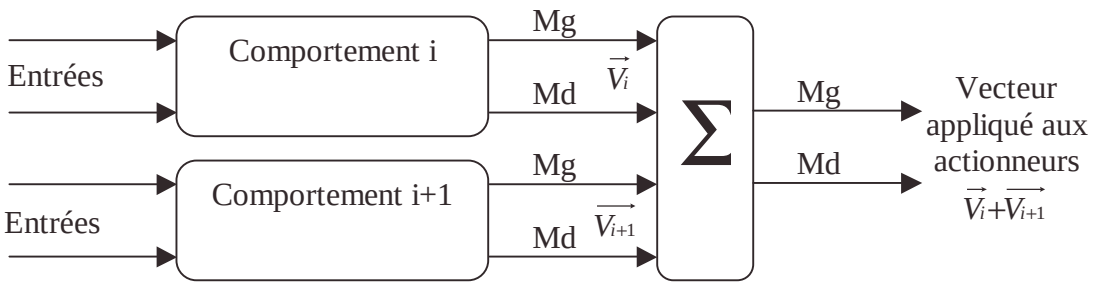


Figure 7 : Un exemple d'architecture à base de combinaison vectorielle

Sur l'exemple de la Figure 7, la combinaison vectorielle est une somme des vecteurs de sorties, mais il existe aussi d'autres architectures basées sur des combinaisons vectorielles plus complexes. Par exemple dans l'architecture satisfaction / altruisme proposée par J. Ferber et O. Simonin [Simonin 00][Lucidarme 02], chaque agent dispose d'une satisfaction personnelle et d'une satisfaction interactive qui est émise vers les autres agents. Ces deux satisfactions seront prisent en compte dans la combinaison des actions. L'architecture d'un agent est donnée sur la Figure 8.

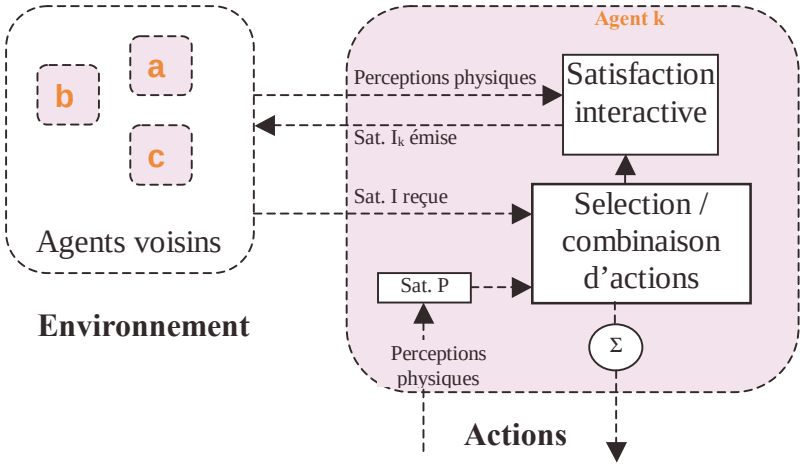


Figure 8 : Architecture satisfaction / altruisme (D'après [Simonin 00])

2.5 Les communications dans les systèmes multi-agents

2.5.1 Introduction

Très tôt, les communications ont été l'un des principaux atouts des systèmes multi-agents. Il existe différents types de communications. Une des communications les plus simples est basée sur l'exemple du Petit Poucet [Drogoul 92]. Elle consiste à déposer sur le terrain des marques qui pourront être captées ou ramassées par d'autres agents. Ces marques peuvent servir à définir un territoire qui a déjà été visité ou encore le chemin à suivre pour atteindre une zone pertinente de l'environnement. On appelle ce type de communications : les communications indirectes. Mais il en existe d'autres types : les communications directes. Elles peuvent servir à transmettre des informations sur la position, l'environnement proche ou encore l'état interne des agents. Par exemple, dans le modèle basé sur la satisfaction [Simonin 00] l'un des éléments principaux de la communication est la transmission d'un vecteur qui représente la satisfaction de l'agent.

Lorsque le nombre d'agents devient trop important, le problème des communications se complexifie. En effet, faire communiquer entre eux plusieurs dizaines de robots pose un certain nombre de problèmes. En premier lieu, lorsque le canal de communication est unique, les messages ne doivent pas entrer en collision. Un protocole de communication doit être instauré afin de pouvoir adresser des messages individuels aux agents ou à l'ensemble du groupe. La similitude avec les réseaux informatiques est grande. D'ailleurs, un point commun entre les réseaux et les communications dans les systèmes multi-agents réside dans la dynamique du système, la perte ou l'ajout d'un agent ne doit pas empêcher le système de continuer à fonctionner.

Toutefois, une particularité des systèmes multi-agents réside dans le concept de proximité. Les communications peuvent être globales ou locales. Dans ce deuxième cas, seuls les agents qui se trouvent à proximité de l'agent émetteur peuvent recevoir le message. Ce type de communication peut aussi être employé pour parvenir à une communication globale grâce à la propagation des messages. Dans l'architecture satisfaction/altruisme [Simonin 00], la satisfaction interactive des agents est constamment transmise et propagée au travers de l'environnement de façon implicite, par les agents.

2.5.2 Les protocoles de communication supervisé

Les premières méthodes permettant de faire communiquer des robots mobiles consistaient à centraliser tous les échanges de messages. Ce type de protocole existe toujours dans divers système multi-robots. Comme par exemple dans le projet Martha ou l'objectif est de coordonner plusieurs dizaines de robots mobiles transportant des containers dans des aéroports ou sur des quais [Alami 98]. Dans ce type de système, il y a un robot superviseur qui centralise les décisions et les communications.

Il existe aussi des systèmes distribués utilisant un système de communication centralisé comme l'illustrent les agents de L. Parker [Parker 99] qui utilisent une machine centrale qui supervise les messages, des communications par réseau central sans fil, comme les robots fourrageurs (Figure 9) de D. Goldberg et M.J. Mataric [Goldberg 99]. Ce type de système utilise un réseau TCP/IP avec des stations fixes.

Figure 9 : Les quatre robots IS Robotics R2e utilisés par D. Goldberg et M.J. Mataric (D'après [Goldberg 99])

Mais ce type d'approche supervisée est inadapté aux systèmes distribués pour deux raisons principales. Afin de conserver les propriétés de robustesse et d'autonomie de ces systèmes, il est préférable de ne jamais centraliser des informations essentielles sur un système unique. De plus, il faut constamment garantir que chaque agent puisse rester en contact avec la station ou le robot superviseur, ce qui n'est pas toujours possible, notamment dans des tâches d'exploration.

2.5.3 Les protocoles de communication distribué

La tendance actuelle évolue vers l'utilisation de protocoles Internet (TCP/IP) [Winfield 00]. Ce type de protocole nécessite un processeur embarqué puissant et l'utilisation de cartes PCMCIA Wireless LAN embarquée sur les robots. Ce type d'architecture onéreuse présente des consommations non négligeables. De plus, malgré l'utilisation d'émetteurs/récepteurs UHF à fort débit, le taux de communication global reste faible : 10 messages par robot par seconde dans l'application de A. Winfield et O. Holland. Nous allons maintenant présenter les trois protocoles les plus représentatifs :

Principe du multiplexage. Ce protocole est employé dans le cas d'un canal unique : chaque agent se voit attribué un laps de temps ou une fréquence du canal. Dans le cas d'un découpage temporel, les agents peuvent utiliser à tour de rôle la bande passante afin d'y envoyer leurs messages. Avec cette méthode, les taux de transferts décroissent fortement avec le nombre d'agents. De plus, l'utilisation de la bande passante est mauvaise, si par exemple un ou

plusieurs agents n'ont pas de message à envoyer pendant plusieurs périodes, leur laps de temps est quand même attribué et non utilisé. Ce type de communication demande de synchroniser parfaitement les agents et le nombre d'utilisateurs est limité par le découpage. De plus l'allocation dynamique (ajout ou suppression d'un agent) est envisageable, mais complexe à mettre en œuvre.

Principe de CSMA (Carrier Sense Multiple Access). Ce protocole, issu des réseaux informatiques, consiste à émettre des trames de façon " anarchique " mais de tolérer les collisions et d'accepter de devoir renvoyer une trame plus tard. Ce type de protocole écoute le canal et envoie la trame dès que celui-ci est libre. Si une collision est détectée, la trame sera à nouveau envoyée après un laps de temps aléatoire afin d'éviter une nouvelle collision. Evidement, ce type de protocole n'est efficace que lorsque le taux de charge du canal reste inférieur au maximum. Lorsque tous les agents veulent émettre une trame, les collisions constantes accumulent les messages réitérés, augmentant ainsi la charge du réseau.

Les protocoles de communication locaux. Dans le cas des systèmes multi-agents distribués, il n'est pas toujours possible (ou utile) de transmettre des messages à l'ensemble des agents. Dans le cas de l'architecture proposée par O. Simonin [Simonin 01], la propagation des signaux se fait de façon naturelle par l'architecture. Donc les communications locales sont amplement suffisantes. Dans ce type de systèmes, la configuration évolue constamment : des agents quittent et rejoignent fréquemment des groupes locaux communicants. C'est pour cette raison que l'allocation doit pouvoir être dynamique.

Il existe donc divers protocoles de communication spécifiques à différentes applications. Toutefois, ces protocoles, souvent inspirés des réseaux informatiques, ne permettent pas toujours de répondre aux problèmes spécifiques des systèmes multi-agents distribués. Nous proposons dans ce manuscrit un nouveau protocole de communication permettant de répondre spécifiquement aux besoins de nos systèmes [section 5.3.4].

2.6 Conclusion

L'objectif de ce chapitre est de présenter les systèmes multi-agents. Les travaux et résultats antérieurs nous permettent de définir certains concepts comme l'opposition entre les approches réactives et cognitives. L'approche réactive peut se présenter comme un complément ou une alternative aux architectures traditionnelles. Ou encore, cette même approche permet de résoudre des problèmes classiques de façon moins complexe et les solutions proposées peuvent être plus facilement mises en œuvre sur des robots réels. Nous avons présenté et décrit trois agents réactifs : Silly Robot, Tom Thumb et Chain-Making Robots. Ces trois exemples me semblent particulièrement pertinents et permettent de montrer les faits suivants :

- Il est possible de décomposer un problème complexe en une succession d'actions/réactions basiques.
- Une modification (même minime) dans un comportement réactif permet d'augmenter de façon significative les performances générales du système.
- L'apport de communications (même indirectes) dans un système multi-agents permet des modifications comportementales importantes, qui se traduisent par une amélioration des performances.

A l'issue de ce chapitre, il me semble important de préciser la philosophie générale de ces travaux. Dans les recherches antérieures, quand les approches mathématiques le permettent,

une démonstration de convergence ou une preuve d'optimalité permet de compléter ces travaux. Dans notre cas, nous avons choisi de privilégier la robustesse au détriment de l'optimalité. Nous préférons avoir un système moins efficace en terme de rendement, mais plus performant en terme d'adaptation. Le but est de fabriquer des systèmes capables de reconfigurer rapidement leur structure interne en cas de panne afin de pouvoir achever leur mission. Les approches traditionnelles demanderaient de prévoir toutes les perturbations possibles afin de proposer une solution à chacune d'entre elles. Toutefois, pour des systèmes complexes devant évoluer en milieu inconnu, envisager l'ensemble des problèmes est impossible, c'est pourquoi nous préférons utiliser une approche basée sur l'apprentissage, afin que les agents soient capables de s'adapter automatiquement aux changements. Le chapitre suivant présente les techniques d'apprentissage les plus courantes.

CHAPITRE

3

3. Les méthodes d'apprentissage

3.1 Introduction

Depuis les débuts de la robotique, l'approche traditionnelle de programmation des robots la plus utilisée est basée sur l'analyse et la géométrie. Cette approche a permis de résoudre un grand nombre de problèmes, d'obtenir des preuves de convergence et de stabilité. Un excellent exemple reste probablement la commande de bras manipulateurs, basée sur la géométrie et la cinématique (modèle de Denavit et Artenberg), elle permet d'obtenir des trajectoires rapides et précises. La stabilité de ces machines a pu être démontrée grâce notamment au critère de Liapounov. Toutefois, il persiste de nombreux problèmes que cette approche ne peut résoudre. Le principal étant qu'il faut d'envisager l'ensemble des cas possibles pendant la programmation. En effet, la réaction d'un système pré-programmé face à une situation inconnue est incertaine. Il a donc fallu trouver des solutions alternatives à l'approche mathématique pour permettre aux machines d'être plus autonomes.

Une fois de plus, la source d'inspiration nous vient principalement du vivant. Chacun de nous est un magnifique exemple de ce que la nature sait faire de mieux en matière d'apprentissage et d'adaptation. Nous sommes capables de résoudre une grande diversité de problèmes, de nous déplacer, de voir, d'analyser des situations inconnues etc. Pourtant, nous ne sommes pas obligés de calculer des changements de repères complexes pour nous déplacer. La nature a développé des stratégies alternatives afin de pouvoir analyser rapidement une situation et en déduire les meilleures réactions. Bref, l'étude de l'homme et du milieu animal reste probablement une des sources d'inspiration les plus riches au monde. Que ce soit pour l'homme ou pour l'animal, il n'y a que trois origines possibles à ces comportements :

- Ils sont soit le résultat d'une imitation. Certains singes disposent de cette faculté à transmettre par l'exemple certaines aptitudes à leur progéniture. Ils leurs montrent notamment comment utiliser à bon escient un bâton ou une pierre pour ouvrir une noix de coco. Le jeune singe, écarté de ses parents après la naissance, n'est pas capable d'acquérir par lui-même cette faculté. On parle dans ce cas de mimétisme.

- Ils peuvent être le fruit d'un apprentissage individuel. Prenons l'exemple d'un jongleur : si vous n'avez jamais appris à jongler et que vous essayez pour la première fois, c'est l'échec assuré. Donc, la capacité de jongler n'est pas inhérente à notre patrimoine génétique. Ce n'est pas non plus en regardant quelqu'un jongler pendant plusieurs heures que vous pourrez acquérir cette faculté. Le seul moyen d'apprendre à jongler est de s'entraîner. C'est donc bien le fruit d'un auto-apprentissage.

- Enfin, ils peuvent être le fruit de l'évolution : un jeune poulain peut se lever et tenir debout quelques heures seulement après sa naissance. Il est évident qu'il n'a pas appris en quelques heures à se tenir debout, il transportait donc dans son patrimoine génétique cette aptitude à se maintenir sur ses pattes.

Les principales techniques d'apprentissage en robotique peuvent être regroupées selon les trois exemples ci-dessus : les méthodes évolutionnistes, l'apprentissage par l'exemple enfin l'apprentissage par renforcement. Mis à part ce dernier, basé sur un outil statistique, les 2 autres sont directement inspirées de l'étude du vivant. Dans ce chapitre, nous allons présenter en détails les techniques d'apprentissage les plus utilisées.

En premier lieu, nous allons nous intéresser à une techniques d'apprentissage par l'exemple, à savoir les réseaux de neurones. Comme son nom l'indique, cette technique est directement inspirée de l'étude du cerveau. L'objectif consiste à montrer une série d'exemples, chaque exemple étant associé à un stimulus d'entrée. Après cette phase d'apprentissage, si le réseau est stimulé avec une configuration d'entrées, alors il saura automatiquement de quel exemple il s'agit. Les réseaux de neurones trouvent de nombreuses applications dans de nombreux domaines, notamment en classification de données ou en reconnaissance vocale.

Ensuite, nous présenterons les techniques d'auto-apprentissage. Ces méthodes, basées sur les processus markoviens, permettent notamment de construire un modèle d'apprentissage. Une fois ce modèle construit, la technique consiste à déterminer quelles sont les meilleures actions à réaliser. Lorsque ces actions permettent au système d'obtenir une récompense importante, leurs liens d'activation sont renforcés, on parle alors d'apprentissage par renforcement.

En dernier lieu, nous présenterons les algorithmes évolutionnistes. Inspirés de l'évolution darwinienne, le principe général consiste à attribuer à chaque individu une séquence génétique correspondant à un comportement. Régulièrement, les individus sont rassemblés pour pouvoir effectuer des croisements et des mutations sur leurs chaînes chromosomiques. L'apprentissage est donc découpé en générations successives et les performances de l'ensemble de la population sont ainsi accrues au fil de l'évolution.

Bien sûr, toutes ces techniques d'apprentissage nécessitent un critère à optimiser, comment estimer la performance d'un individu au fil de l'évolution ? Ou encore comment récompenser un agent dans le cadre de l'auto-apprentissage ? Avant de conclure ce chapitre, nous présenterons et définirons les différents moyens d'estimer une performance.

3.2 Les réseaux neuronaux

3.2.1 Introduction et historique

En 1943, deux bio-physiciens de l'université de Chicago, Mac Cullogh et Pitts proposent le premier modèle de neurone biologique [Cullogh 43]. Ce neurone formel, aussi appelé neurone à seuil, est inspiré des récentes découvertes en biologie. Ce sont des neurones logiques (0 ou 1).

En 1949, le psychologue Donald Hebb introduit le terme connexionisme pour parler de modèles massivement parallèles et connectés [Hebb 49]. Il propose de nombreuses règles de mise à jour des poids dont la célèbre " règle de Hebb ".

En 1958, le psychologue Frank Rosenblatt, combinant les idées de ses prédécesseurs, propose le premier perceptron [Rosenblatt 58]. Ce réseau, capable d'apprendre à différencier des formes simples et à calculer certaines fonctions logiques, est inspiré du système visuel.

Au début des années 60, les travaux de Rosenblatt suscitent un vif enthousiasme dans le milieu scientifique. Mais en 1969, deux scientifiques américains de renom, Minsky et Papert, publient un livre [Minsky 69] qui démontre les limites du perceptron proposé par Rosenblatt. En particulier, son incapacité à résoudre les problèmes non linéairement séparables, dont la fonction logique XOR est un célèbre exemple.

Les travaux ralentissent considérablement jusqu'aux années 80. En 1982, Hopfield démontre l'intérêt des réseaux entièrement connectés [Hopfield 82]. Parallèlement, Werbos conçoit un mécanisme d'apprentissage pour les réseaux multicouches de type perceptron: la rétropropagation (Back-Propagation). Cet algorithme, qui permet de propager l'erreur vers les couches cachées sera popularisé en 1986 dans un livre " Parallel Distributed Processing " par Rumelhart *et al* [Rumelhart 86].

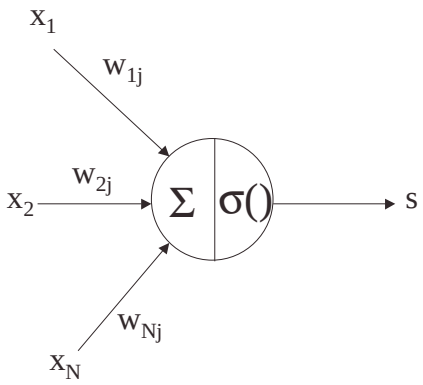
Depuis ces travaux, les applications des réseaux de neurones n'ont cessé de croître. Il a d'ailleurs été démontré qu'un réseau MLP (Multi Layer perceptron) avec seulement deux couches peut approximer n'importe quelle fonction de R^n dans R^m avec une précision arbitraire.

3.2.2 Notation

Nous rappelons brièvement le modèle général du neurone artificiel, qui est l'élément de base de beaucoup de réseaux. Il est composé des éléments suivants :

- une ou plusieurs entrées pondérées,
- un sommateur,
- une fonction de transfert,
- une sortie.

La Figure 10 montre le schéma général du neurone artificiel



avec

- x_i le stimulus d'entrée,
- w_{ij} la valeur du poids synaptique reliant le stimulus i au neurone j,
- $\sigma()$ la fonction de sortie du neurone,
- s la sortie du neurone.

Figure 10 : Schéma général d'un neurone artificiel

La fonction d'un neurone artificiel est donnée par l'équation suivante :

$$S=\sigma\left(\sum_{j=1}^{n} X_j.W_{ji}\right)$$

Équation 1 : Equation de la fonction de transfert d'un neurone

Un réseau de neurones n'est finalement qu'une représentation conviviale de fonctions mathématiques. En effet, chaque réseau peut s'écrire sous la forme d'une équation. La fonction de transfert de base des réseaux est donnée par l'Équation 1. La fonction de sortie des neurones est principalement utilisée pour mettre en forme les signaux de sortie des neurones. Si par exemple le système est binaire, une fonction de type seuil pourra être utilisée (Figure 11). Si le système est borné (par exemple un réseau dont la sortie est appliquée directement sur un actionneur) alors la fonction de sortie peut aussi être bornée (Figure 12).

Nous avons présenté ici deux fonctions fréquemment rencontrées, mais d'autres fonctions orientées vers d'autres applications plus spécifiques existent aussi. Nous verrons notamment la particularité des réseaux RBF (Radial Basis Functions) au chapitre 3.2.4.

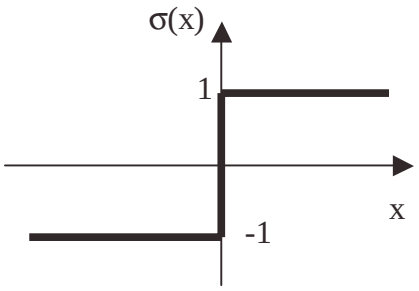


Figure 11 : Fonction de sortie de type seuil

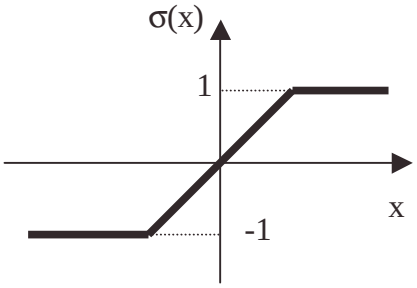


Figure 12 : Fonction de sortie bornée

3.2.3 La rétropropagation du gradient

La structure d'un réseau de neurones est constituée de neurones reliés entre eux par des liaisons synaptiques. A chacune de ces liaisons est associé un poids synaptique et ce sont précisément les modifications apportées à ces poids qui vont nous intéresser. De manière générale, l'apprentissage neuronal consiste à ajuster les valeurs des poids synaptiques du réseau pour que la réponse du réseau soit celle désirée. Dans un premier temps, nous allons présenter l'apprentissage des poids pour un réseau sans couche cachée.

- **Réseaux sans couche cachée.** Dans ce type de réseau, il y a autant de neurones que de sorties. Les entrées sont directement reliées aux neurones par l'intermédiaire des liaisons synaptiques. Le principe général consiste à calculer l'erreur de chaque sortie, en différenciant la sortie désirée et la sortie obtenue (Équation 2), puis à ajuster les poids afin de diminuer cette erreur (Équation 3). Le coefficient η représente le coefficient d'apprentissage, plus sa valeur sera faible, plus l'apprentissage sera long et stable.

Algorithme d'apprentissage d'un réseau sans couche cachée

1. Initialisation des poids à des valeurs aléatoires de faible grandeur,
2. Sélection d'un exemple dans la base d'apprentissage <x,d>,
3. Propager l'entrée à travers le réseau et calculer les sorties o_i
4. Pour chaque sortie du réseau calculer le terme d'erreur:

$$\Delta w_{ij} = \eta (d_i - o_i) x_i \qquad \textbf{Équation 2}$$

5. Mettre à jour chaque poids synaptique du réseau:

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \textbf{Équation 3}$$

6. Retourner en 2. tant que l'erreur est trop grande.

Algorithme 1 : Apprentissage neuronal d'un réseau sans couche cachée

Les réseaux de neurones sans couche cachée ne peuvent modéliser que des fonctions linéaires. Pour illustrer ces propos, prenons l'exemple classique d'un réseau devant modéliser une fonction logique : il possède deux entrées x_1 et x_2, un perceptron unique et une sortie s. La fonction de transfert du réseau est donnée par l'Équation 4. Il s'agit là d'une fonction linéaire et nous allons optimiser les paramètres de la droite w_{11} et w_{21}.

$$s = \sigma(x_1 . w_{11} + x_2 . w_{21})$$

Équation 4 : Fonction de transfert d'un réseau modélisant une fonction logique

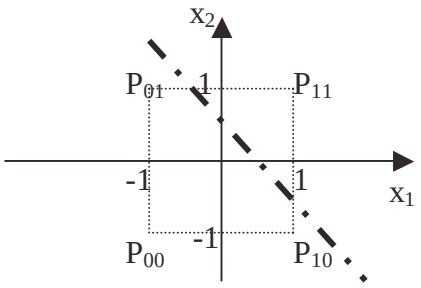


Figure 13 : Représentation du réseau dans l'espace des entrées

Par exemple, pour modéliser une fonction logique ET, il existe une infinité de droite permettant d'isoler le point P_{11} des points P_{00}, P_{01} et P_{10} (Figure 13). En revanche, si l'objectif est de modéliser une fonction " OU EXCLUSIF ", il n'existe aucune droite permettant d'isoler les points P_{11} et P_{00} des points P_{01} et P_{10}. Les réseaux sans couche cachée ne permettent pas de modéliser toutes les fonctions. L'ajout d'une couche intermédiaire permet donc de découper l'espace en zones. Si une fonction de sortie linéaire est conservée, alors les zones sont séparées par des droites, mais il y a plusieurs droites, ce qui permet de dissocier des zones non linéairement séparables. Les réseaux de neurones sont d'ailleurs fréquemment utilisés pour résoudre des problèmes de classification.

Réseaux avec couche cachée. La méthode la plus connue est basée sur la rétropropagation du gradient. Cette méthode, dite supervisée, consiste à présenter des exemples au réseau multicouches, puis à en propager l'erreur entre la sortie désirée et la sortie obtenue à travers le réseau afin de corriger tous les poids, même ceux des couches cachées. Un exemple est composé d'un vecteur d'entrées et de sorties désirées <x,d>. L'algorithme de rétropropagation du gradient est décrit sur l'Algorithme 2.

Algorithme de rétropropagation du gradient

1. Initialisation des poids à des valeurs aléatoires de faible grandeur,

2. Sélection aléatoire d'un exemple dans la base d'apprentissage <x,d>,

3. Propager l'entrée à travers le réseau,

4. Appliquer l'entrée x sur le réseau et calculer la sortie o de chaque couche,

5. Pour chaque sortie du réseau calculer le terme d'erreur:

$$\delta_k = (d_k - o_k)\sigma'(x_i) \qquad \textbf{Équation 5}$$

6. Pour chaque couche cachée, calculer le terme d'erreur:

$$\delta_k = \sum_{k=1}^{N} w_{ki}.\sigma'(x_i) \qquad \textbf{Équation 6}$$

7. Mettre à jour chaque poids synaptique du réseau:

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \textbf{Équation 7}$$

Où

$$\Delta w_{ij} = \eta.\delta_j.x_{ij} \qquad \textbf{Équation 8}$$

8. Retourner en 2. tant que l'erreur est trop grande.

Algorithme 2 : Algorithme de rétropropagation du gradient

La méthode présentée sur l'Algorithme 1 est en fait une version simplifiée de la rétropropagation du gradient puisque cela est équivalent à appliquer l'Algorithme 2 sur un système sans couche cachée. L'efficacité et la rapidité de convergence de cette méthode ne sont plus à prouver, seulement deux problèmes persistent : lors de la mise à jour des poids de la couche cachée, l'ensemble du réseau est modifié, c'est-à-dire que le comportement du réseau sera modifié pour tous les exemples entrés, même s'il n'a été mis à jour que pour un seul exemple. Dans l'espace des entrées, la modification n'est pas locale. Ensuite, il n'est pas toujours aisé d'estimer le nombre de neurones dans la couche cachée. Ce deuxième problème est principalement dû à la difficulté de donner une signification physique à cette couche cachée. Nous avons choisi de présenter ci-après une autre famille de réseaux de neurones qui permet justement de s'affranchir des deux problèmes précédents.

3.2.4 Les réseaux à fonctions radiales

Les réseaux à fonctions radiales (RBF : Radial Basis Functions) permettent l'apprentissage de fonctions complexes; ils sont basés sur une structure différente des perceptrons. Ces réseaux sont généralement composés de deux couches : une couche cachée

de neurones RBF (dont la fonction de transfert est donnée par l'Équation 9) et une couche de sortie constituée de perceptrons.

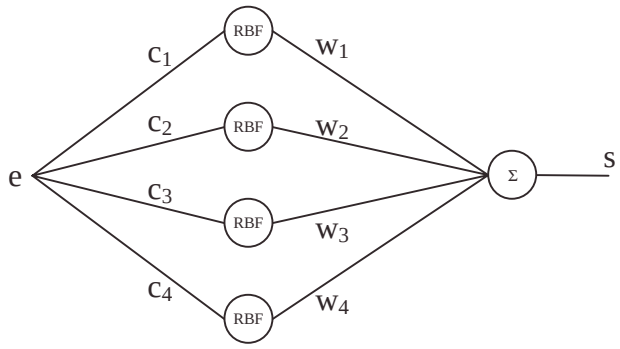


Figure 14 : Exemple de réseau RBF

L'exemple de la Figure 14 montre un réseau permettant l'approximation d'une fonction de $\mathcal{R}$ dans $\mathcal{R}$ (s=f(e)). Chaque fonction radiale possède un centre c_i, la sortie globale c'est la somme des sorties des neurones de la couche cachée. La Figure 15 montre la courbe à estimer en continu, ainsi que la sortie de chaque neurone de la couche cachée en pointillé. Chaque neurone a une influence locale sur la courbe estimée.

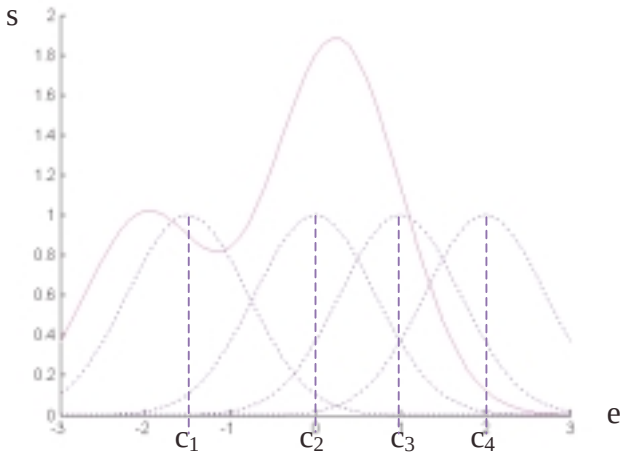


Figure 15 : Fonction à estimer et sortie de chaque neurone de la couche cachée.

Bien sûr, ce modèle peut être étendu à des fonctions de $\mathcal{R}^n$ dans $\mathcal{R}^m$. La Figure 16 montre un exemple de fonction radiale dans $\mathcal{R}^2$. La fonction de base la plus couramment utilisée est donnée par l'Équation 9.

$$f(x)=e^{-(x-c_i)^2}$$

Équation 9 : Fonction de transfert d'un neurone de type RBF

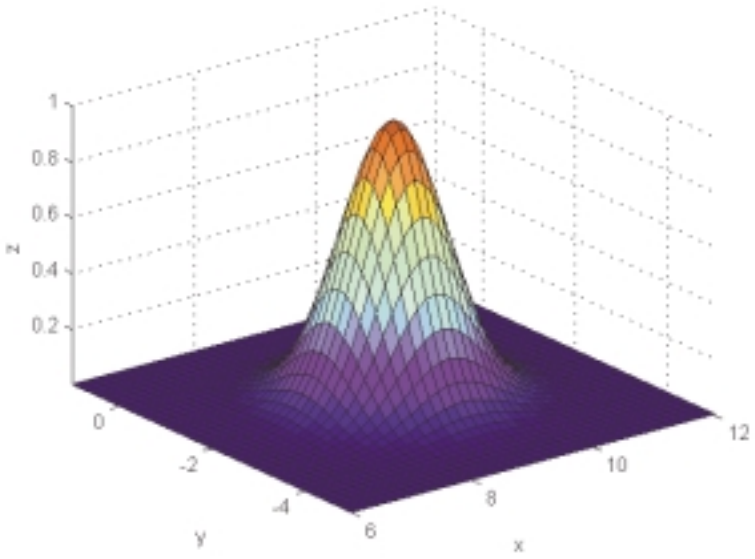


Figure 16 : Exemple de fonction radiale dans $\mathcal{R}^2$

Généralement, les centres sont fixés et seuls les w_i sont modifiés. La règle de rétropropagation du gradient est utilisée sur la seule couche de sortie. Comme le réseau ne possède pas de couche cachée devant être mise à jour, la méthode présentée sur l'Algorithme 1 est parfaitement adaptée pour l'apprentissage. C'est pour ces mêmes raisons que la méthode est simple et rapide à implémenter. Toutefois, le choix de la position des centres et le nombre de neurones reste généralement arbitraire. Ils sont généralement répartis uniformément sur l'intervalle de définition de la fonction f=s(e). Il existe des méthodes de positionnement automatique des centres: ces méthodes consistent à déplacer les centres vers les points où l'erreur de modélisation est grande. L'accumulation de fonctions autour des zones complexes à modéliser améliorent l'approximation, car les fonctions radiales ont une influence limitée autour de leurs centres.

3.2.5 Conclusion

Il existe un grand nombre de méthodes permettant de réaliser l'apprentissage neuronal. De manière générale, ces méthodes sont supervisées, elles nécessitent des exemples pour réaliser l'apprentissage et elles sont difficilement applicables dans les cas d'auto-apprentissage et d'adaptation où aucun exemple de comportement n'est connu.

3.3 L'apprentissage par renforcement

3.3.1 Introduction

L'apprentissage par renforcement a pour objectif de maximiser la performance du système en renforçant les meilleures actions. L'agent obtient des informations de l'état de l'environnement (nous les appellerons les perceptions) et agit également sur l'environnement puis reçoit une estimation de sa performance : la récompense. Dans les algorithmes d'apprentissage par renforcement, cette récompense peut être immédiate ou retardée. Le schéma général est représenté sur la Figure 17. L'agent connaît son état (S_t) dans

l'environnement, il a la possibilité d'agir sur ce dernier (a_t) et il reçoit une récompense (r_t). L'objectif de l'apprentissage par renforcement est d'associer à chaque état du système une action qui permet de maximiser la récompense.

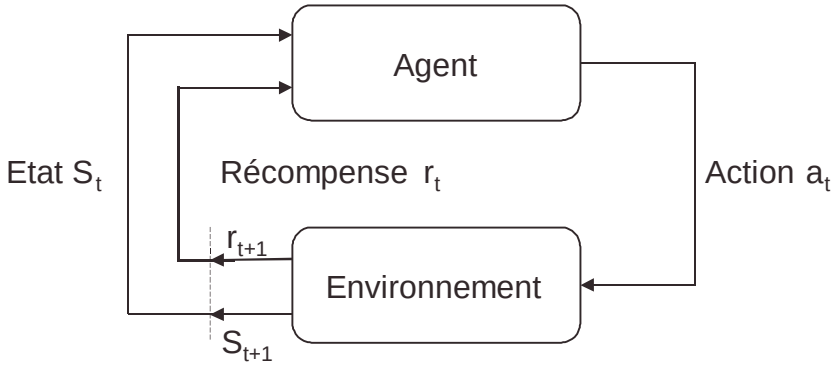


Figure 17 : Système d'apprentissage par renforcement.

On distingue deux types d'algorithmes; les stratégies "off policy" et les stratégies "on policy". Dans le premier cas, l'apprentissage est divisé en deux phases :

- une phase d'apprentissage, qui consiste à essayer des stratégies aléatoires afin de modéliser le système,
- une phase d'application où l'agent n'exécute que les meilleures actions apprises dans la phase précédente.

Dans le cas des stratégies "on policy", l'agent applique la stratégie apprise au fur et à mesure de sa progression et de son apprentissage.

3.3.2 Les processus markoviens

L'apprentissage par renforcement est basé sur l'apprentissage à temps discret des paramètres d'une chaîne de Markov. Les chaînes de Markov sont composées d'états et de transitions entre ces états. Prenons par exemple le cas d'un agent qui peut se déplacer dans un environnement discrétisé (Figure 18). L'environnement est décomposé en 16 cases représentant chacune un état du processus markovien. La position de départ est représentée par la case S et l'objectif est d'atteindre la case E. L'agent dispose de 4 actions possibles : se déplacer vers le nord, l'est, le sud ou l'ouest.

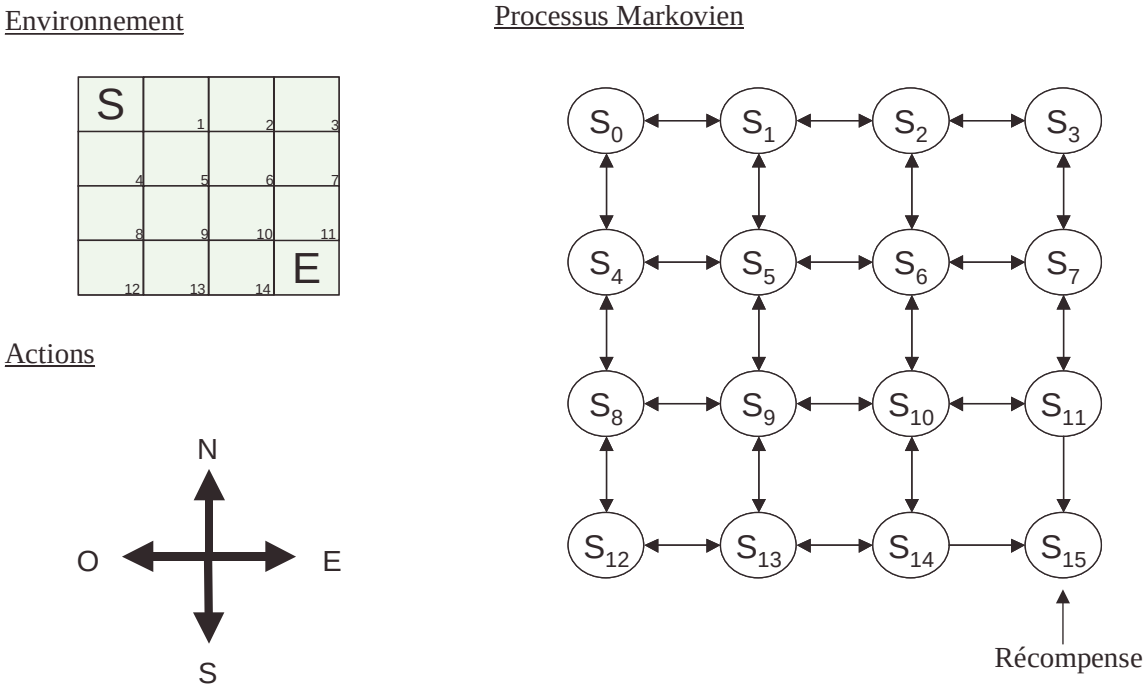


Figure 18 : Exemple de processus markovien

Dans ce cas, le système est déterministe, lorsque l'agent se trouve dans un état donné (S_n) et qu'il réalise une action donnée, l'état (S_{n+1}) suivant sera toujours le même. Dans un système non déterministe, la même action réalisée depuis le même état (S_n) peut aboutir à des états (S_{n+1}) différents. Dans ce cas, les transitions du processus markovien représentent la probabilité pour chaque état d'être atteint en fonction de l'action réalisée. Le schéma général de l'apprentissage par renforcement consiste à associer à chaque action une fonction de cette probabilité et de la récompense associée à l'action. La récompense moyenne espérée pour chaque action peut ainsi être déterminée. De cette façon, il ne reste plus qu'à exécuter l'action possédant la meilleure récompense espérée dans la phase d'application de la stratégie.

3.3.3 La programmation dynamique

Algorithme de la programmation dynamique

1. Initialiser V(s)=0, pour tout s ∈ V*,
2. Mettre V(s) à jour :

$$V(s) = \sum_{Actions} (r_{t+1} + \gamma . V(s_{t+1}))$$

avec :
- r_{t+1} : récompense instantanée
- $V(s_{t+1})$: Evaluation de l'état S_{t+1}

4. Retourner en 2 tant que l'erreur est trop grande.

Algorithme 3 : Algorithme de la programmation dynamique

La programmation dynamique s'applique pour les systèmes déterministes dont le modèle complet est connu, c'est-à-dire que tous les états et toutes les transitions sont connues. La méthode consiste à estimer la fonction d'évaluation V* associée à chacun des états du

système, puis à choisir des actions qui permettent d'atteindre l'état ayant la meilleure estimation. C'est un apprentissage "off policy", qui est détaillé sur l'Algorithme 3.

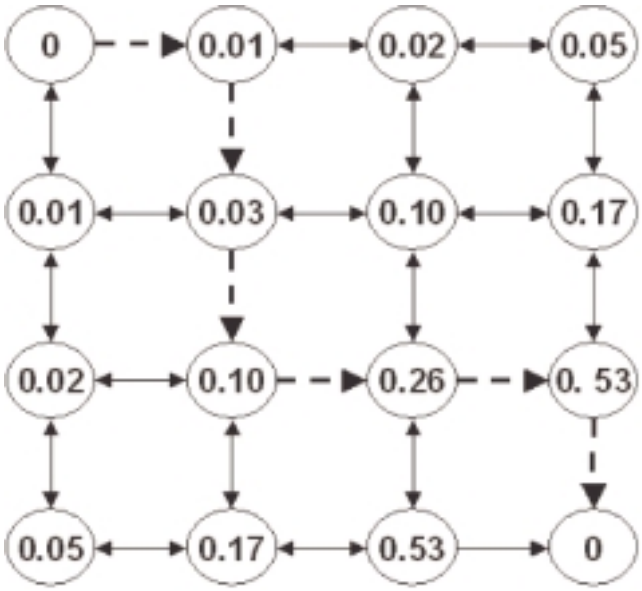


Figure 19 : V* après cinq cycles d'apprentissage

A la fin de la phase d'apprentissage, V* ≈ V. Si l'on considère par exemple le problème de la Figure 18, le résultat de l'apprentissage est représenté sur la Figure 19. On distingue la valeur associée à chaque état du système. Par exemple de l'état S_5, on peut atteindre quatre autres états : S_1, S_4, S_9 et S_6 (Figure 18). Les récompense espérée de chacun de ses états sont respectivement 0.01, 0.01, 0.10 et 0.10. La phase d'application de la stratégie consiste à choisir l'action qui permet d'atteindre l'état ayant la meilleure fonction d'évaluation. Les deux états S_9 et S_6 correspondent donc aux deux meilleurs successeurs de l'état S_5. Un exemple de stratégie optimale est représenté en pointillés sur la figure. La programmation dynamique permet de trouver la stratégie optimale lorsque le modèle complet est connu et déterministe. Ce type d'apprentissage est particulièrement bien indiqué pour l'apprentissage de stratégies, par exemple dans des jeux de réflexion. J'ai programmé un joueur pour le jeu de Nim[1] en 2000. Il s'agit d'un problème déterministe puisqu'il n'y a aucun tirage au sort dans le jeu. Le programme est constitué de deux phases : une phase d'apprentissage ou le joueur joue contre lui-même, puis une phase de jeu ou l'on peut affronter l'algorithme. La méthode utilisée est la programmation dynamique. A chaque configuration de jeu est associée la probabilité de gagner et le joueur tente constamment d'amener le jeu dans la meilleure configuration connue. La solution mathématique de ce jeu est connue et si deux joueurs parfaits s'affrontent, le joueur qui ne joue pas le premier coup est sûr de remporter la partie. Après quelques secondes d'apprentissage en jouant contre lui-même, le joueur joue les meilleurs coups au sens de la solution mathématique connue.

[1] Le jeu de Nim consiste à disposer 4 lignes de 1, 3, 5 et 7 allumettes. Les joueurs enlèvent chacun leur tour autant d'allumettes qu'ils le souhaitent sur la ligne de leur choix. Le joueur qui prend la dernière allumette perd la partie. Ce type de jeu a été rendu célèbre par le film d'Alain Resnais : « L'Année dernière à Marienbad » en 1961.

Mais en robotique, ces hypothèses déterministes sont rarement réunies. Il existe une autre méthode permettant de réaliser l'apprentissage avec une connaissance partielle du modèle: le Q-learning. Cet apprentissage permet également d'être appliqué aux systèmes non déterministes.

3.3.4 Le Q-learning

Proposé en 1989 par Watkins [Watkins 89], le Q-learning est probablement la méthode d'apprentissage par renforcement la plus étudiée actuellement. Les seules hypothèses de départ sont les suivantes :

- le système est modèlisable par une chaîne de Markov à états finis,
- l'agent dispose d'un jeu d'actions discret.

Il n'est pas nécessaire de connaître les transitions entre les états, c'est le grand intérêt du Q-learning. Contrairement à la programmation dynamique, l'objectif n'est pas d'estimer la fonction d'évaluation de chaque état, mais celle de chaque action, en fonction de l'état courant.

Algorithme du Q-learning

1. Initialiser Q (s,a) de façon arbitraire, pour chaque état s et action a
2. Choisir une action a
3. r ← récompense instantanée obtenue
4. s' ← nouvel état courant
5. Mettre Q(s,a) à jour selon l'équation suivante:

$$Q(s,a)=Q(s,a)+\alpha(r+\gamma\max_{a'}(Q(s',a'))-Q(s,a)) \qquad \text{Équation 10}$$

avec :

$\alpha=\frac{1}{1+visit(s,a)}$

γ coefficient d'amortissement

6. Incrémenter le nombre de visites de l'état (s,a) :

$$visit(s,a)=visit(s,a)+1 \qquad \text{Équation 11}$$

4. Retourner en 2.

Algorithme 4 : Algorithme du Q-Learning

La description du Q-learning est montrée sur l'algorithme 4. Le Q-learning est basé sur l'estimation de la matrice Q(s,a), qui représente la récompense espérée en exécutant l'action a depuis l'état s. Le point 2 est volontairement resté imprécis. En effet, le Q-Learning peut être utilisé comme une méthode " off-policy " ou " on-policy ". Dans le premier cas, il faudra alors choisir une action aléatoirement. Dans le deuxième cas, il faudra choisir une action en accord avec la stratégie apprise, c'est-à-dire l'action a qui maximise Q(s,a).

Le coefficient γ représente le coefficient d'amortissement. A la fin de l'apprentissage, la matrice Q peut être assimilée à un gradient qu'il suffit de remonter pour maximiser la récompense. La Figure 20 montre l'état de la matrice Q à la fin de l'apprentissage. On distingue clairement ce gradient pour chaque action. Le coefficient d'amortissement γ va

modifier la pente de ce gradient. Un coefficient de 1 attribue la récompense espérée maximale à tous les états, alors qu'un coefficient de 0 attribue une récompense espérée de 0 à tous les états sauf aux états finaux. Sur la Figure 20, le coefficient γ a été arbitrairement fixé à 0,9.

Il a été démontré que pour des systèmes déterministes le Q-Learning permet d'atteindre la solution optimale après un temps d'apprentissage infini [Szepesvari 97]. Il a d'ailleurs été démontré que la matrice Q converge vers l'espérance mathématique de la somme de la récompense instantanée et de la récompense espérée du meilleur état suivant possible (Équation 12). La même équation pouvant bien sûr s'écrire pour l'état suivant ($Q(S_{t+1},a')$), on en déduit que remonter le gradient emmènera l'agent vers la solution optimale.

$$Q(s_t,a) \approx E[r_{instantanée} + \gamma . \max_{a'}(Q(s_{t+1},a')]$$

Équation 12 : Espérance mathématique de la récompense espérée pour l'action a exécutée depuis l'état S_t

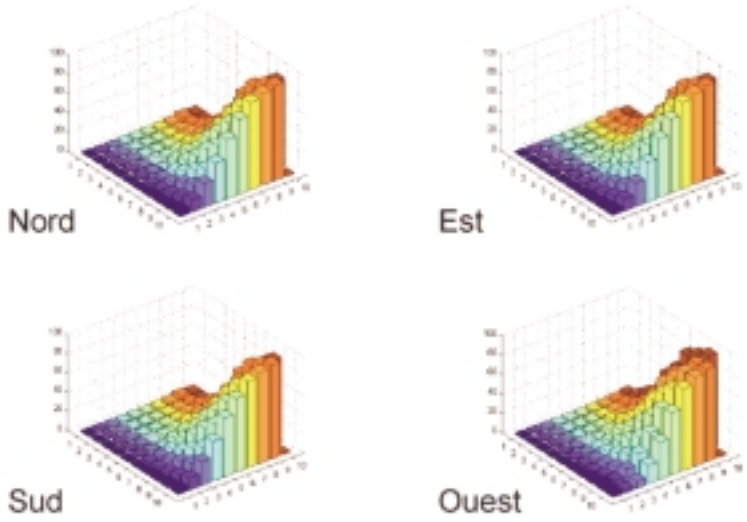


Figure 20 : Matrice Q pour chacune des actions.

3.3.5 Conclusion

L'apprentissage par renforcement est un outil performant pour réaliser différents types d'apprentissage. Il existe de nombreuses autres méthodes dérivées de la programmation dynamique et du Q-Learning, comme la différence temporelle (TD), Sarsa et Monte-Carlo. Toutes ces méthodes sont basées sur les processus markoviens et demandent de discrétiser l'environnement en états et les consignes sur les actionneurs en commandes échantillonnées. Pour parfaitement modéliser un problème réel appliqué à la robotique, la taille de la matrice V ou Q serait proportionnelle au nombre de combinaisons états/actions possibles. Il est évident qu'il est impossible de stocker autant de données sur un système embarqué. De plus l'apprentissage serait extrêmement long, puisqu'il demanderait à l'agent de visiter au moins une fois chaque combinaison état/action possible.

Même si ces méthodes semblent mal indiquées pour l'apprentissage de comportements réactifs continus, elles peuvent être utilisées pour gérer les couches supérieures de

l'architecture. En 2000, Maja Mataric et Dany Goldberg proposent une architecture multi-agents basée sur le principe des AMMs (Augmented Markovians Models) [Goldberg 00]. Il s'agit là d'une structure basée sur les processus markoviens, qui consiste à construire sa propre bibliothèque d'états au fur et à mesure de l'apprentissage. La technique permet de combiner deux états identiques et d'éviter de stocker des valeurs pour des états qui ne seront jamais rencontrés.

3.4 Les algorithmes évolutionnistes

3.4.1 Introduction

Les algorithmes évolutionnistes sont inspirés de l'étude du vivant et plus précisément de l'évolution darwinienne. Le principe général [Goldberg 89] consiste à disposer d'une population d'individus possédant chacun une chaîne chromosomique dans laquelle est codé son comportement. La méthode nécessite de connaître une estimation de la performance de chaque agent (fitness). Dans la première génération, les chaînes chromosomiques sont choisies aléatoirement. Les expériences sont divisées en générations, lorsque les performances de tous les individus de la première génération sont connues des croisements sont réalisés entre les individus afin de créer la génération suivante. Le principe général de la méthode est décrit sur l'Algorithme 5 issus de [Mitchell 97].

Cette citation résume à elle seule le principe général de la méthode :

" J'ai donné le nom de sélection naturelle ou de persistance du plus apte à cette conservation des différences et des variations individuelles favorables à cette élimination des variations nuisibles "

Charles Darwin, L'origine des espèces

Algorithmes évolutionnistes

1. Initialisation de la population P_1: remplir aléatoirement la chaîne chromosomique de chaque individu h_i.

2. Evaluer chaque individu de la population courante $F(h_i)$.

3. Créer la génération suivante en sélectionnant statistiquement les géniteurs. La probabilité qu'un individu soit utilisé pour générer la population suivante est donnée par :

$$\Pr(hi)=\frac{F(h_i)}{\sum_{j=1}^{Nbindividus} F(h_j)} \qquad \textbf{Équation 13}$$

4. Réaliser les croisements sur les chaînes chromosomiques.

5. Réaliser les mutations.

8. Retourner en 2.

Algorithme 5 : Principe général des algorithmes évolutionnistes (D'après [Mitchell 97])

Prenons pour exemple le problème du juste chiffre : nous disposons de quatre nombres et le but est de s'approcher le plus près possible d'un cinquième nombre en réalisant des opérations arithmétiques avec les 4 premiers. Nous supposerons que les opérateurs sont : l'addition, la multiplication, la soustraction et la division. Nous émettrons également comme hypothèse que l'opération arithmétique doit être constituée de trois opérateurs sans priorité (il n'y a pas de parenthèse dans l'opération). Par exemple, si les chiffres sont : 24 25 30 et 7 et que le résultat doit être 142, une bonne solution sera : 24 * 25 = 600 / 30 = 20 * 7 = 140. Si un algorithme génétique est utilisé pour résoudre ce problème, la chaîne chromosomique pourra être constituée de 14 bits : 8 bits pour les 4 nombres (Tableau 1) et 6 bits pour les 3 opérateurs (Tableau 2). Un exemple de chaîne chromosomique est donné sur le Tableau 3.

Codage	Nombres
00	24
01	25
10	30
11	7

Tableau 1 : Codage des chiffres dans la chaîne chromosomique

Codage	Opérateurs
00	+
01	-
10	*
11	/

Tableau 2 : Codage des opérateurs dans la chaîne chromosomique

0	1	0	1	0	0	0	0	1	0	1	0	1	1
25		-		24		+		30		*		7	

Tableau 3 : Exemple de chaîne chromosomique pour le problème du juste chiffre

3.4.2 Les opérateurs génétiques

Les opérateurs génétiques sont utilisés pour générer la population suivante. On distingue deux types d'opérateurs : les opérateurs de croisement, qui réalisent une opération sur deux individus et les opérateurs de mutation, qui n'opèrent que sur un seul individu.

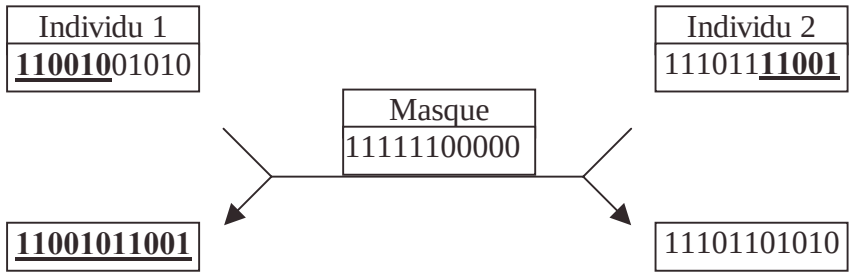
Les croisements : L'opérateur de croisement produit deux nouvelles chaînes à partir de deux chaînes initiales. Le masque de croisement est un mot binaire de même longueur que les chaînes génétiques. Ce masque contribue à déterminer lequel des deux parents sera géniteur de chaque bit de la génération suivante. Trois grandes familles de croisements sont distingués dans la littérature (voir Figure 21).

- Les croisements simple point. Les deux chaînes initiales vont être divisées en deux. La première partie de la première chaîne sera associée à la seconde partie de la seconde chaîne et inversement, deux nouveaux individus sont ainsi obtenus résultant d'un croisement entre les deux chaînes initiales.

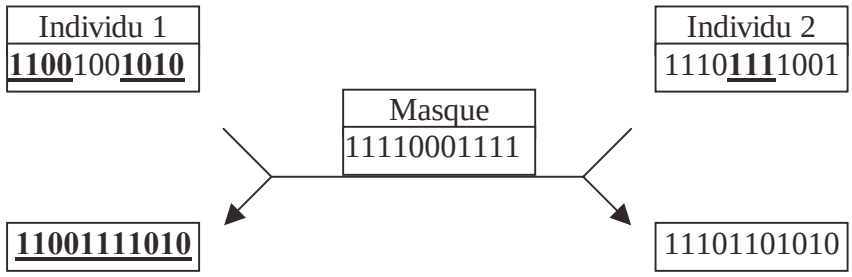
- Les croisements double points. Le principe est assez proche des croisements simple point, à cette différence qu'il y a deux points de séparation des chaînes, la chaîne initiale est divisée en 3 parties et la combinaison de ces 3 parties permet d'obtenir deux nouvelles chaînes.

- Les croisements uniformes. Le masque de croisement est choisi aléatoirement, chaque bit peut être choisi indépendamment du reste de la chaîne.

Croisement simple point :



Croisement double point :



Croisement uniforme :

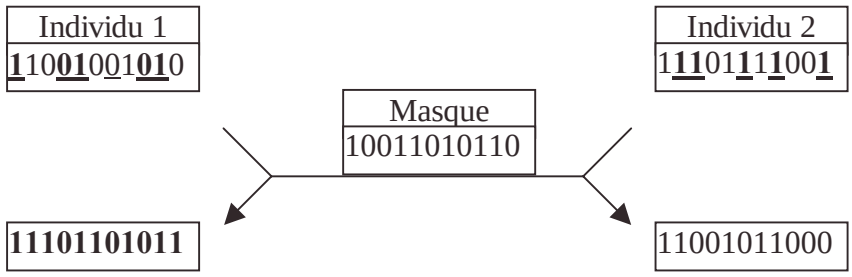


Figure 21 : Opérateurs de croisement sur des chaînes binaires

Il est assez courant d'utiliser la chaîne chromosomique pour y coder des valeurs binaires, entières ou réelles. De ce fait, les bits de la chaîne n'ont pas tous la même importance ; un changement sur un bit de poids fort n'a pas la même influence qu'une modification sur un bit de poids faible. Pour que le sens des croisements soit justifié, des opérateurs numériques sont parfois utilisés : on réalise par exemple une moyenne pondérée des valeurs numériques des deux parents.

Individu géniteur 1

0	1	**0**	**1**	0	0	0	**0**	1	0	**1**	**0**	**1**	1
25		-		24		+		30		*		7	

Individu géniteur 2

1	**0**	1	1	**0**	**1**	**1**	0	**1**	**1**	1	0	1	**1**
30		/		25		*		7		*		7	

Individu de la génération suivante

0	0	0	1	0	1	1	0	1	1	1	0	1	1
24		-		25		*		7		*		7	

Figure 22 : Exemple de croisement uniforme pour le problème du juste chiffre

Les mutations : Les mutations effectuent une modification à partir d'une seule chaîne initiale. Les mutations consistent à complémenter l'un des bits de la chaîne initiale (Figure 23). Les mutations permettent notamment d'obtenir des individus avec de nouvelles propriétés qui n'auraient pas été accessibles en ne réalisant que des croisements à partir de la population initiale.

Figure 23 : Exemple de mutation

Comme pour l'opérateur de croisement, lorsque la chaîne représente des valeurs numériques, une mutation sur un bit de poids fort n'a pas la même influence que la même opération sur un bit de poids faible. Des opérateurs de mutation numériques sont parfois utilisés, le principe est de réaliser un tirage aléatoire centré sur la valeur numérique codée dans la chaîne initiale.

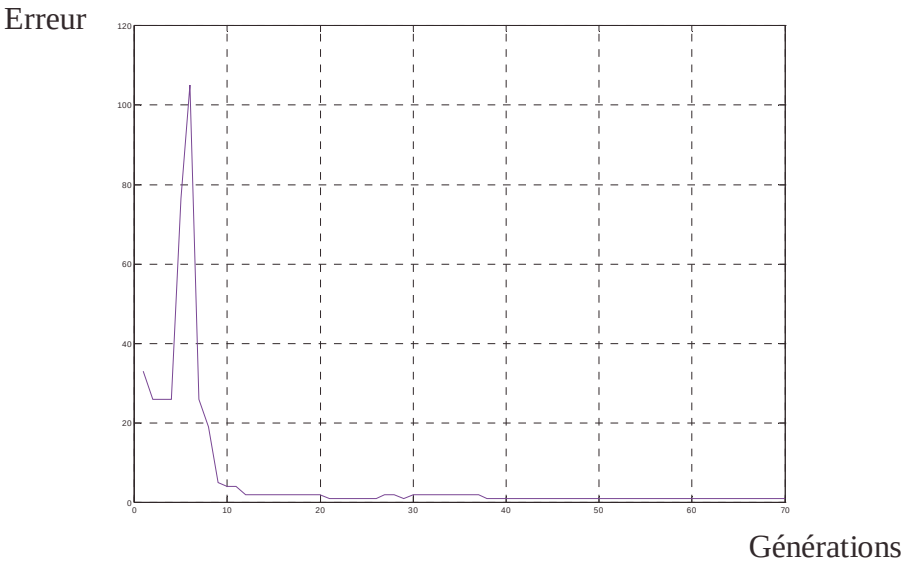


Figure 24 : Evolution du meilleur individu au fil des générations

Si l'on reprend l'exemple du juste chiffre présenté précédemment, un exemple de croisement est présenté sur la Figure 22. L'apprentissage a été réalisé sur une population de 20 individus. L'évolution du meilleur individu est montré sur la Figure 24. On distingue que la

solution optimale et la stabilité du système est atteint après une quarantaine de générations. En réalisant l'apprentissage plusieurs fois, on trouve 4 résultats équivalents (L'objectif était d'atteindre le chiffre 142) :

- ((7 * 25) – 7) – 25 = 143
- ((25 * 7) – 7) – 25 = 143
- ((7 * 25) – 25) – 7 = 143
- ((25 * 7) – 25) – 7 = 143

3.4.4 Conclusion

Les algorithmes évolutionnistes semblent particulièrement bien indiqués pour l'apprentissage des systèmes multi-agents. En premier lieu, il faut disposer d'une population d'individus, ce qui est le cas dans tout système multi-robots. Ensuite, la possibilité de coder toutes sortes de comportements dans la chaîne chromosomique ouvre un grand nombre de possibilités. Toutefois, la méthode est supervisée : après chaque expérience, l'ensemble des chaînes chromosomiques doit être rassemblé pour construire la génération suivante. Ce type d'apprentissage demande de centraliser les données. De plus, les agents ne peuvent pas toujours interrompre leur tâche courante pour réaliser les croisements.

3.5 L'estimation de la performance

Il était impossible de clôturer ce chapitre sans discuter d'un point clef de toute méthode d'apprentissage : l'estimation de la performance. Appelée différemment selon les méthodes (la récompense pour le Q-Learning, l'aptitude pour les algorithmes génétiques ou encore l'erreur pour les réseaux de neurones artificiels) l'estimation de la performance est un paramètre essentiel de l'apprentissage. Le principal intérêt de l'apprentissage est de laisser l'agent trouver par lui-même la meilleure stratégie pour maximiser la récompense. Si la récompense est trop contrainte, l'apprentissage ne présente aucun avantage par rapport aux méthodes traditionnelles. Prenons l'exemple extrême du Morpion (Tic-Tac-Toe en anglais) où deux adversaires s'affrontent en disposant des croix et des cercles sur une grille carrée de 3 cases de côté. Le premier qui aligne trois symboles remporte la partie. Si le joueur est récompensé de la façon suivante :

- +1 si c'est un bon coup,
- 0 sinon.

la récompense suppose de connaître *a priori* les bons coups. De ce point de vue, l'apprentissage n'a aucun lieu d'être utilisé, autant programmer tous les bons coups. En revanche, si l'agent est récompensé de la manière suivante:

- +1 en cas de victoire,
- -1 en cas de défaite,
- 0 en cas de match nul.

Dans ce cas, l'agent devra trouver les meilleurs coups à jouer pour maximiser sa récompense qui ne lui sera accordée qu'à la fin de la partie. Ce petit exemple peut paraître

évident, mais des récompenses trop contraintes sont parfois utilisées et elle verrouillent l'apprentissage dans une stratégie qui n'est pas nécessairement optimale. Dans [Madani 02], l'auteur explique "Pour notre application la fonction de renforcement choisie permet de délivrer un signal +0.5 lorsque le robot choisit une bonne action (récompense) et -0.5 pour une mauvaise (punition)". Si les bonnes et les mauvaises actions sont connues par avance, l'apprentissage n'a aucun intérêt. Il s'agit dans cet article d'un ralliement de cible avec évitement d'obstacles; une récompense moins contraignante aurait pu être :

- +1 lorsque la cible est atteinte,
- -1 en cas de collision avec un obstacle,
- 0 sinon.

Un autre exemple intéressant est l'apprentissage d'un comportement d'évitement d'obstacles [Floreano 94]. Il est important de préciser que l'expérience se déroulait dans un environnement très contraint. La récompense est donnée par l'Équation 14.

$$R_i = V.(1-\sqrt{\Delta v}).(1-i)$$

où :

- V est la vitesse moyenne de rotation des deux roues,
- Δv est la différence signée de la vitesse des deux roues,
- i est une valeur proportionnelle à la quantité d'obstacles proches du robot.

Équation 14 : Récompense instantanée attribuée à un robot mobile pour une tâche de navigation sans collision (D'après [Floreano 94])

Cette estimation de la performance se divise en trois termes: le premier tend à maximiser la vitesse du robot, le second, à récompenser les trajectoires rectilignes et le dernier à minimiser les collisions. Ma première analyse a été de penser que maximiser la vitesse des deux roues revenait à minimiser la vitesse de rotation du robot et que les deux premiers termes étaient redondants, voire contraignants. Francesco Mondada (qui est l'auteur de [Floreano 94]) a expliqué que l'environnement était tellement contraint qu'une des meilleures stratégies trouvée par le robot était de tourner en rond sur place, maximisant ainsi la vitesse moyenne des deux roues et minimisant le nombre de collisions. Il a donc choisi d'adapter la récompense à l'environnement et par conséquent l'apprentissage ne sera valide que dans des cas où l'environnement sera très contraint.

J'ai classé les différents types d'estimations de la performance en 4 catégories :

L'erreur : Utilisée dans les réseaux de neurones, l'erreur donne une information proportionnelle à la différence entre la sortie désirée et la sortie obtenue. De plus lorsque la sortie du système est un vecteur, l'erreur est également un vecteur de même dimension. Cette estimation de la performance est la plus contraignante car elle exige un modèle de comportement connu et fait tendre l'apprentissage à copier ce modèle.

Les récompenses instantanées : Les récompenses instantanées donnent une estimation de la performance correspondant à la stratégie courante. Contrairement à l'erreur, même si la sortie du système est un vecteur, la récompense instantanée est un paramètre unidimensionnel. La récompense présentée par l'Équation 14 est un exemple de récompense instantanée.

Le coût : La plupart des méthodes d'apprentissage sont présentées comme cherchant à maximiser la récompense. L'objectif de la tâche peut aussi être de minimiser un critère, par exemple une consommation d'énergie ou un temps de déplacement. Dans ces cas, on parle de coût et non de récompense. Mathématiquement, minimiser une fonction de coût peut toujours se ramener à maximiser une récompense.

Les récompenses retardées : Une récompense retardée est attribuée à la fin d'une séquence d'actions. Elle estime la performance de l'ensemble de ces actions. Un exemple de récompense retardée est le Tic-Tac-Toe présenté ci-dessus. Le joueur est récompensé à la fin de la partie pour l'ensemble de ses coups. Ce type de récompense est parmi les moins contraignants.

3.6 Conclusion

Nous avons présenté différentes techniques d'apprentissage possédant chacune leurs propres spécifications : les réseaux de neurones peuvent apprendre à mimer des fonctions mathématiques à partir d'une base d'exemples, l'apprentissage par renforcement permet d'apprendre des stratégies sans intervention extérieure, ni même d'exemple, enfin les algorithmes évolutionnistes peuvent faire évoluer des comportements afin d'en améliorer leurs performances.

Mais aucune de ces techniques ne peut résoudre à elle seule la problématique qui nous est imposée ici. Les réseaux de neurones demandent des exemples pour pouvoir fonctionner, ils sont incapables de générer d'eux-mêmes une stratégie permettant de résoudre un problème ou un conflit. Si l'apprentissage par renforcement permet d'apprendre sans exemple une stratégie optimale, cette technique demande de discrétiser l'univers en états et actions échantillonnés. Or, une quantité importante de données doit pouvoir être stockée, ce qui peut alors être disproportionné au regard des capacités de stockage actuelles. Enfin, si les algorithmes évolutionnistes semblent bien appropriés à notre problématique, il n'en reste pas moins que c'est une technique supervisée. Le point faible réside dans le rassemblement des séquences génétiques pour pouvoir réaliser les croisements et les mutations.

Les techniques présentées dans ce chapitre représentent pour nous des outils qui permettent d'appliquer l'apprentissage sur nos systèmes. Comme nous venons de le voir, aucune de ces techniques ne peut être appliquée tel quel afin de répondre à la problématique donnée. C'est pourquoi nous allons, dans les chapitres suivants, présenter des versions améliorées, voire combiner plusieurs techniques afin de répondre aux critères imposés. Afin de tester et valider les techniques proposées nous disposons d'une plate-forme expérimentale sur laquelle les méthodes sont testées. Cette plate-forme est présentée dans le chapitre suivant.

CHAPITRE

4

4. Notre plate-forme expérimentale

4.1 Introduction

Les travaux sur l'apprentissage que nous avons étudiés précédemment n'étaient pas toujours appliqués à la robotique. Les applications les plus fréquentes sont la résolution de problèmes mathématiques, la recherche d'optimum dans des fonctions complexes ou encore la recherche de stratégies gagnantes dans des jeux comme les échecs ou le célèbre joueur de backgammon programmé par Tesauro [Tesauro 94]. Ces applications, comme les premiers travaux appliqués à la robotique, étaient simulées sur un ordinateur. Rapidement, des différences notables entre le monde simulé et l'environnement réel ont empêché une transposition exacte des résultats simulés vers de véritables plate-formes. C'est pour cette raison que la nécessité d'expérimenter les méthodes proposées sur de véritables robots mobiles s'est faite ressentir. Parmi les contraintes imposée : les temps d'apprentissage doivent être inférieurs aux temps de décharge des accus, les méthodes doivent être robustes vis à vis des pannes, de la décharge des batteries et du bruit, qu'il s'agisse aussi bien de celui des capteurs, que celui des actionneurs.

Afin de tester et comparer les méthodes, nous avons mis au point une plate-forme expérimentale complète. Cette plate-forme est constituée de quatre robots mobiles (Type 1); ces robots ont principalement été construits dans l'optique de réaliser des expériences sur des systèmes hétérogènes. Cette même plate-forme a d'ailleurs été utilisée pour valider les travaux d'Olivier Simonin et de Jacques Ferber en 2000 [Lucidarme 02] portant sur l'étude de l'architecture Satisfaction / Altruisme.

Ces travaux de thèse s'intègrent au sein des recherches actuelles du laboratoire. C'est pourquoi notre plate-forme expérimentale doit pouvoir rester générique et non dédiée à une application typique. Nous avons choisi de compléter le parc de robots homogènes par une plate-forme hétérogène inspirée du concept de l'aveugle et du paralytique : ce concept, proposé par A. Liègeois et P. Rongier [Rongier 01], consiste à classer les agents en deux catégories :

- les aveugles : ce sont des agents qui ne possède soit aucune perception de l'environnement, soit une perception réduite. En contrepartie, ces agents peuvent agir sur l'environnement, se déplacer, prendre des objets etc,
- les paralytiques : ces agents ne peuvent interagir avec l'environnement, ils ne sont pas équipés de bras manipulateur, voire de moyen de locomotion. En revanche, ils ont une perception plus fine de l'environnement.

L'hypothèse de départ était de travailler sur des systèmes en mode dégradé : les aveugles sont des agents dont le système de vision est défaillant et les paralytiques des agents dont le système de locomotion n'est plus utilisable. L'idée globale de ce genre de système est inspirée d'une fable de Jean Pierre de Florian de Clarisse où un aveugle et un paralytique s'allient afin de diminuer leurs misères respectives. Grâce aux communications directes entre les agents, il est facile d'imaginer un système où les agents paralytiques guident les agents aveugles dans leurs actions. Notre plate-forme expérimentale a été conçue tout en restant fidèle à ce concept. Nous avons construit en premier lieu un manipulateur mobile miniature(M^3) de type aveugle qui ne possède pas de vision fine, en revanche il peut doublement agir sur l'environnement : par la locomotion et par l'utilisation du bras manipulateur. Nous avons ensuite construit un système de vision stéréoscopique (BOP) qui ne dispose d'aucun moyen d'action sur l'environnement, mais qui peut avoir une perception très fine du monde.

4.2 Le robot Type 1

4.2.1 Introduction

Le robot Type 1 représentant la base de la plate-forme est un robot de petite taille et évolutif. Nous disposons d'une plate-forme de quatre robots mobiles Type 1. Voici les principales contraintes qui forment le cahier des charges :

- puissance de calcul embarquée,
- 45 minutes d'autonomie énergétique minimum,
- vitesse du robot atteignant les 1 $m.s^{-1}$,
- ceinture de capteurs de proximité,
- communications locales entre les robots,
- possibilité de rajouter des cartes avec de nouvelles fonctionnalités.

4.2.2 Description matérielle

Type 1 est un robot mobile autonome équipé de deux roues différentielles montées sur des moteurs à courant continu. La photo de la Figure 25.a montre le châssis du robot équipé des moteurs, des roues et des billes porteuses qui assurent la stabilité du robot. Ce robot est de forme cylindrique, d'environ 13 cm de diamètre et 12 cm de hauteur. Les plans des cartes électroniques sont détaillé dans l'Annexe B.

Les capteurs de proximité : Une ceinture de capteurs infrarouges (IR) permet aux robots de détecter les obstacles, mais aussi de communiquer localement. Cette ceinture est constituée de 8 capteurs et 16 émetteurs. Ces 8 capteurs infrarouges sont uniformément répartis sur la périphérie du robot (Figure 25.c). La portée des capteurs peut être ajustée de quelques centimètres à quelques mètres, soit par logiciel, soit de façon matérielle. Ces capteurs (développés par la société Wany Robotics) sont basés sur la mesure de l'amplitude de l'onde réfléchie par les obstacles. Ils sont également utilisés pour permettre aux robots de communiquer localement. Dans le cas de communications locales, la portée est supérieure au double de la celle mesurée dans les cas de détection d'obstacles. En effet, en détection d'obstacles, l'onde IR doit parcourir deux fois le trajet entre le robot et l'obstacle, une fois à l'aller puis une fois au retour. De plus, une partie de l'onde est absorbée par l'obstacle. Dans le cas d'une communication directe, l'onde doit juste parcourir la distance entre les deux robots.

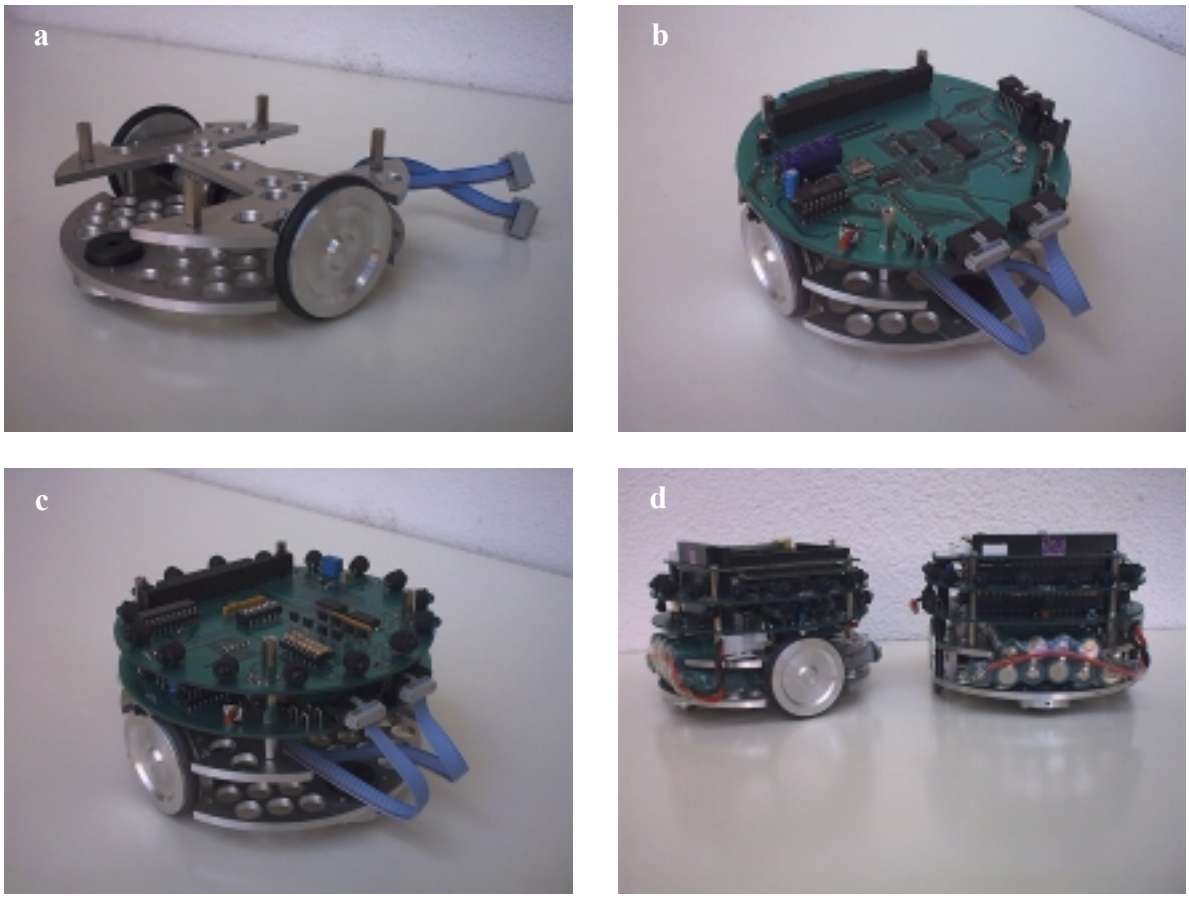


Figure 25 : Détails du robot Type 1

Afin de pouvoir utiliser les capteurs de l'agent, nous avons étudié leur réponse en fonction de l'obstacle. La Figure 26 montre la réponse du capteur sur quatre types d'obstacles : un obstacle blanc, un bleu, un rouge et un obstacle constitué de papier

réfléchissant. Afin de pouvoir généraliser la fonction de transfert liant la valeur numérique renvoyée et la distance avec l'obstacle, nous l'avons interpolé en utilisant la méthode des moindres carrés. Les droites en pointillés discontinus représentent les interpolations locales pour chaque type d'obstacles. Les courbes en pointillés représentent la réponse mesurée du capteur et la droite continue représente la fonction de transfert effectivement utilisée dans le robot. Dans les conditions expérimentales de notre laboratoire, les obstacles sont majoritairement constitués de polystyrène blanc, ce qui n'endommage pas les robots en cas de collisions et permet une bonne réflexion IR. Nous avons naturellement pris cette réponse comme référence pour nos expériences (droite continue).

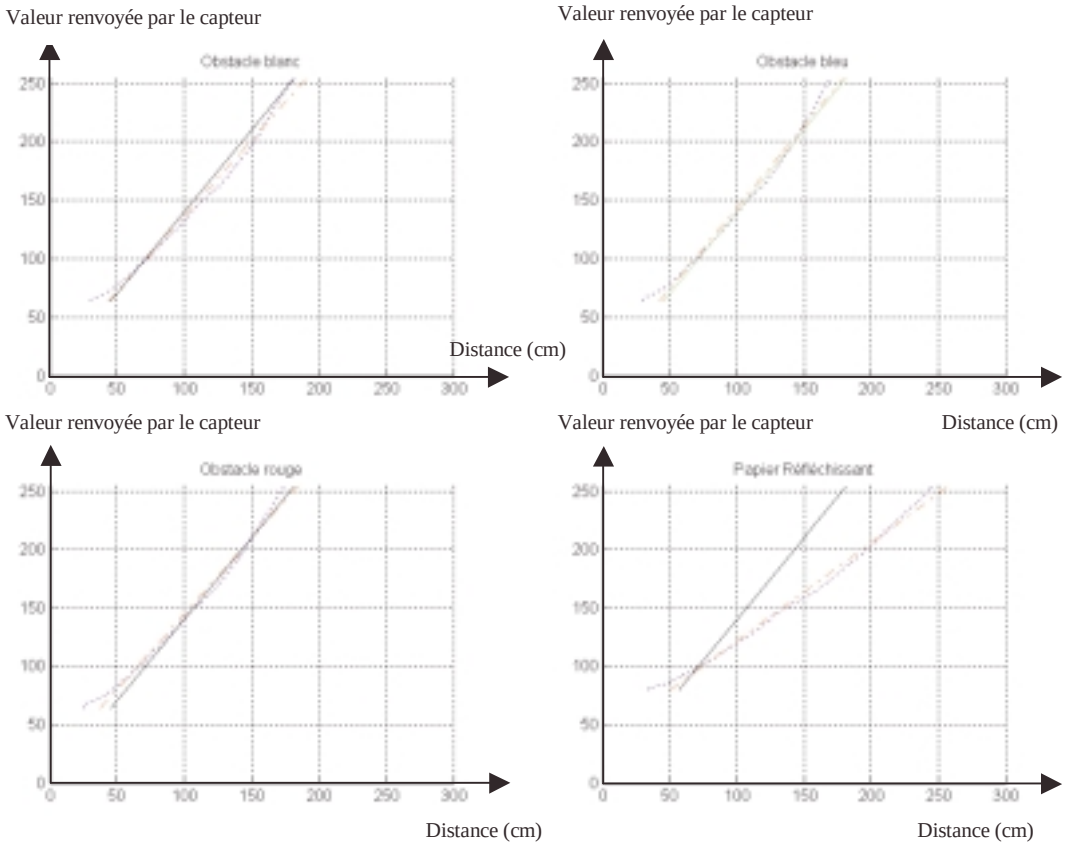


Figure 26 : Réponse des capteurs en fonction de la structure de l'obstacle

Odométrie : les robots sont également équipés de capteurs proprioceptifs : chaque moteur est équipé d'un codeur incrémental magnétique. Ces codeurs disposent de deux canaux de 16 impulsions par tour de moteur, soit 352 impulsions par tour de roue. Ces capteurs sont utilisés pour l'odométrie et permettent notamment de calculer la position du robot. Le modèle géométrique sera présenté au chapitre 4.3.1. Il faut noter que la dérive est importante et que l'odométrie devra être couplée avec un autre capteur pour estimer la position absolue du robot. En revanche, elle peut être utilisée localement lors de courts déplacements, par exemple reculer d'une distance fixe en cas de collision avec un obstacle. Les codeurs sont utilisés dans la boucle de retour de l'asservissement. Ils renvoient une information sur la vitesse de chacune des roues.

Énergie : Le robot est alimenté par deux accus NiMH de 1300mAH chacun. Un accus de 12V alimente toute la partie électronique et un second de 6V alimente la partie puissance (moteurs et émission infrarouge).

Les parties électroniques et puissance ont volontairement été découplées afin que l'électronique ne soit pas perturbée par les pics de consommation des moteurs. Théoriquement, en utilisation normale, le robot peut être utilisé pendant 97 minutes avant d'atteindre la décharge complète des accus. En pratique, l'expérience la plus longue a duré 90 minutes.

Processeur : Comme processeur, nous avons choisi d'utiliser un PC embarqué sur le robot. Ce type de processeur, assez gourmand en énergie (environ 10W) présente les avantages suivants :

- possibilité d'enregistrer un grand nombre de données et de les conserver hors tension,
- possibilité d'utiliser des noyaux temps réels,
- grande puissance de calcul,
- grand nombre d'entrées / sorties (BUS ISA).

Dans les expérimentations présentées ici, nous avons utilisé des architectures basées sur des processeurs 486DX2x66 MHz, avec 8 Mo de RAM et 64 Mo de disque dur Compact Flash. Le BUS PC104 nous permet d'empiler les cartes en mezzanine et de rajouter des cartes par-dessus le PC afin d'ajouter de nouvelles fonctionnalités au robot.

4.2.3 Description du logiciel

Choix du système d'exploitation : en premier lieu, il a fallu choisir le système d'exploitation et, étant donné les contraintes imposées par nos applications, nous avions la possibilité d'installer les systèmes suivants : Linux, Windows 95 ou le DOS.

Dans un premier temps, une distribution de type Linux semblait le meilleur choix. Effectivement, des distributions Linux de petite taille sur lesquelles il est possible d'ajouter un noyau temps réel sont distribuées gratuitement. Seulement, les distributions Linux et " Real Time Linux " sont particulièrement délicates à installer sur des disques durs de type Compact Flash. De plus, il est impératif de quitter proprement le système, c'est-à-dire qu'une extinction brutale du PC ou une coupure d'alimentation peut endommager les données de façon irréversible.

Un système de type Windows 95 quant à lui permet principalement de disposer d'une interface homme-machine conviviale. Mais ce type de système ne garantit aucunement l'aspect temps réel des tâches. De plus, comme pour Linux, une extinction brutale du PC peut endommager le système.

Nous nous sommes donc orientés vers un système de type DOS. Il s'agit d'un DOS 6.0 dans sa version de base. Ce choix peut aujourd'hui paraître archaïque, mais présente nombre d'avantages : le DOS s'installe sans problème sur ce type de machine, c'est un OS de petite taille. Même si l'aspect temps réel n'est pas garanti sur ce type de système, nous avons l'assurance que le programme en cours d'exécution est le seul à utiliser le processeur du PC, car le DOS n'est pas un système multi-tâches. Pour des systèmes plus complexes, les limitations du DOS se feront vite ressentir : la gestion du réseau, du BUS PCI et des ports USB ne sont pas prise en compte.

Compilateur : il existe un grand nombre de compilateurs sous DOS, les plus connus étant le BASIC, le Turbo Pascal et le C/C++. Afin d'assurer la portabilité des programmes du robot

vers d'autres plate-formes, nous avons opté pour un compilateur C/C++ (Nous avons utilisé un compilateur C/C++ distribué gratuitement par la société Borland).

Aspect temps réel : même si le DOS n'est pas un système temps réel, il est possible de garantir l'exécution cyclique d'une tâche. Pour l'asservissement, la consigne appliquée aux moteurs doit être mise à jour toutes les 10 ms. Nous avons reprogrammé le timer servant de base à l'horloge du PC de sorte à déclencher un TOC toutes les 10ms a lieu des 19.8ms d'origine. Ensuite l'interruption a été déroutée afin de lancer l'exécution de notre routine d'asservissement. Diverses mesures ont montré que cette méthode permettait l'exécution de la routine avec une précision supérieure à 99.9%. Ce timer sert de base pour le reste du programme, si une tâche doit être exécutée pendant 2 secondes, le programme attendra l'appel de 200 routines d'interruption, ce qui correspondra à un temps de 200*10ms = 2 secondes. Cette méthode est complètement transparente pour le reste du programme, puisque lors de l'appel d'une interruption, les registres du processeur sont sauvegardés, puis restaurés à la fin de l'exécution de la routine.

Asservissement : chaque moteur est asservi en vitesse, ce qui nous donne la garantie que les consignes appliquées seront respectées indépendamment de la charge des accus. Cela est très important, surtout lorsque la récompense de l'agent est une fonction de la distance parcourue ou de la vitesse moyenne du robot. Après avoir identifié les paramètres de notre système, nous avons expérimenté deux types d'asservissement : un asservissement classique de type PID et une méthode adaptative (la règle du MIT [Astrom 95]). Le principal avantage de la seconde est de s'adapter automatiquement à la charge du robot. Mais nous avons estimé que la complexité des calculs implémentés ne justifiait pas la légère différence de performances entre les deux méthodes. En effet, le PID donne un résultat d'excellente qualité et cette performance varie peu selon la charge du robot.

4.3 Le manipulateur mobile miniature(M^3)

4.3.1 Introduction

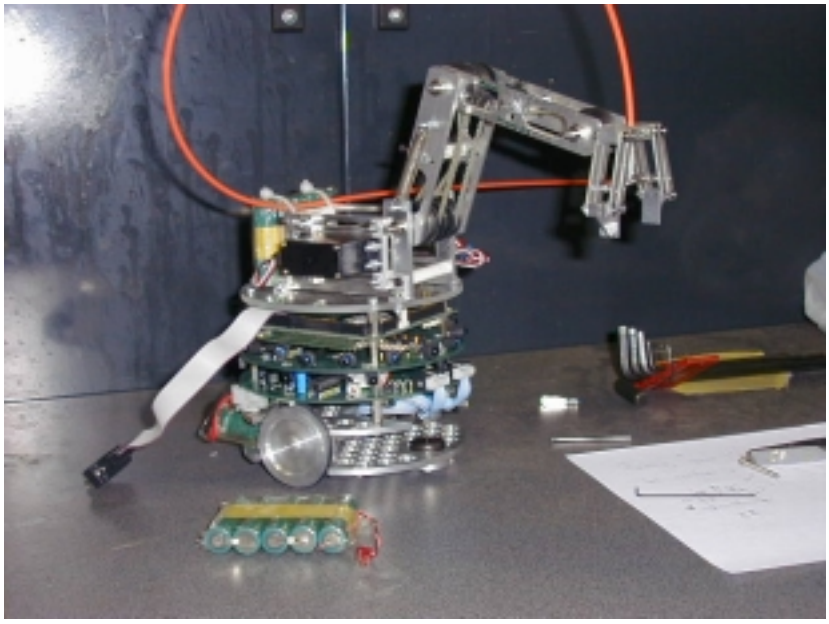


Figure 27 : Notre manipulateur mobile miniature M^3

La plupart des manipulateurs mobiles réalisés pour la recherche sont constitués d'un bras manipulateur industriel monté sur une plate-forme mobile conventionnelle. Une telle configuration engendre une redondance des 3 degrés de liberté de la base, mais en général les deux composants sont commandés séparément : la base mobile amène le manipulateur dans la zone de travail. Puis la base reste fixe et la manipulation commence. De cette façon la base et le manipulateur sont asservis séparément. Quelques travaux antérieurs ont déjà traité la modélisation et la commande d'un manipulateur mobile comme un seul ensemble, incluant le couplage entre le bras et la plate-forme. Dans le cas de la manipulation de lourdes charges ou quand des accélérations importantes sont nécessaires, le couplage en force et inertiel doit être pris en compte afin d'optimiser la commande des moteurs et d'assurer la stabilité de l'ensemble.

Dans notre cas, le bras a été spécialement dessiné pour s'adapter sur le robot mobile Type 1. Nous avons construit un prototype basé sur le principe de la Figure 28. Le bras est constitué de trois moteurs qui actionnent les axes 1 et 2, ainsi que l'organe terminal. Tous les moteurs sont fixés sur le châssis, la transmission est réalisée grâce à des poulies, des courroies et un câble. Ce type de construction permet d'alléger la partie mobile du bras et d'obtenir ainsi une dynamique particulièrement performante (aucun moteur ne doit supporter le poids d'un autre actionneur).

4.3.2 Repères de référence

Ce manipulateur mobile a été conçu pour être utilisé sur un sol parfaitement plat. Les coordonnées absolues sont référencées dans le repère fixe (O, X, Y, Z), où le plan (O, X, Y) est horizontal. Un repère de référence (O_i, x_i, y_i, z_i) est également attribué à chaque corps du robot : la base Type1 (Corps 1), le bras (Corps 2), l'avant bras (Corps 3) et l'organe terminal (Corps 4). Ces 4 derniers repères, visibles sur la Figure 29, restent constamment parallèles les uns aux autres grâce à une transmission à base de courroies et de poulies (Figure 28). Les poulies ont des diamètres égaux, de telle façon que le bras se comporte comme un double parallélogramme. Les points de référence O_i sont localisés au milieu des axes, dans le plan de symétrie du manipulateur mobile. Sur la Figure 29, les lettres majuscules décrivent les coordonnées relatives aux repères fixes et les lettres minuscules les repères mobiles. Les vecteurs sont représentés en italique.

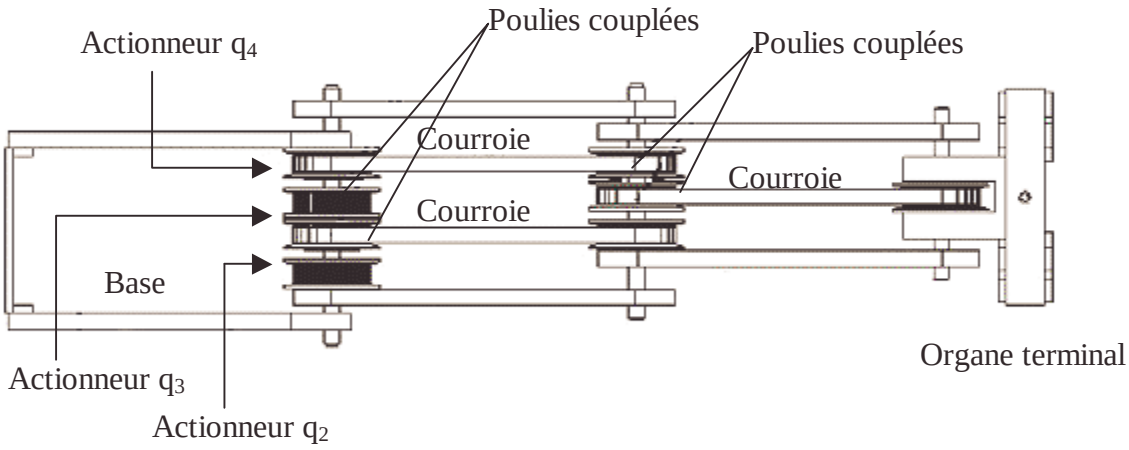


Figure 28 : La cinématique du bras

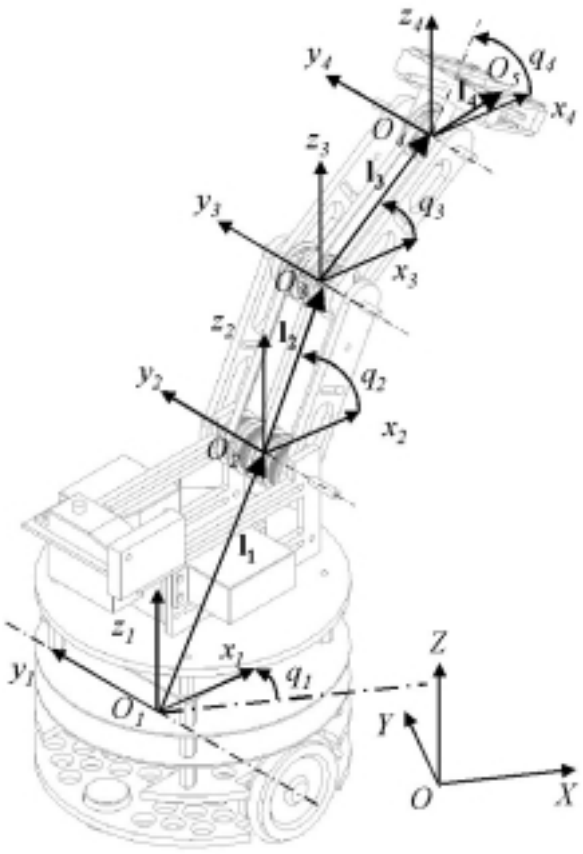


Figure 29 : Disposition des différents repères

4.3.4 Positions et orientations

Les positions et orientations du véhicule sont définies par X_{O_1}, Y_{O_1} et q_1 (Figure 29). La hauteur Z_{O_1} est constante. Posons ω_r et ω_l la vitesse angulaire respective des roues droite et gauche, R le rayon d'une roue et $2d$ l'empattement. Si l'on assure constamment un point de contact entre les roues et le sol, les équations cinématiques du véhicule sont données par l'équation 1.

$$\begin{cases} v = R\dfrac{\omega_r + \omega_l}{2} \\ \dot{X}_{O_1} = vC_1 \\ \dot{Y}_{O_1} = vS_1 \\ Z_{O_1} = R \\ \dot{q}_1 = R\dfrac{\omega_r - \omega_l}{2d} \end{cases}$$

Avec q_1, l'angle de rotation autour de l'axe vertical du véhicule.

Équation 15 : Equations cinématiques

Nous poserons :

$$\begin{cases} C_i = \cos(q_i) \\ S_i = \sin(q_i) \end{cases}$$

Les positions et orientations des autres repères de référence sont faciles à calculer (Équation 16, Équation 17 et Équation 18).

$$\begin{cases} (x_{O_2})_1 = (l_1)_x \\ (y_{O_2})_1 = 0 \\ (z_{O_2})_1 = (l_1)_z \end{cases}$$

Équation 16 : position du repère 2.

$$\begin{cases} (x_{O_3})_1 = (l_1)_x + l_2 C_2 \\ (y_{O_3})_1 = 0 \\ (z_{O_3})_1 = (l_1)_z - l_2 S_2 \end{cases}$$

Équation 17 : position du repère 3

$$\begin{cases} (x_{O_4})_1 = (l_1)_x + l_2 C_2 + l_3 C_3 \\ (y_{O_4})_1 = 0 \\ (z_{O_4})_1 = (l_1)_z - l_2 S_2 - l_3 S_3 \end{cases}$$

Équation 18 : position du repère 4

Calculons maintenant la position du dernier point de référence lié à l'organe terminal (Équation 19).

$$\begin{cases} (x_{O_5})_1 = (l_1)_x + l_2 C_2 + l_3 C_3 + (l_4)_x C_4 + (l_4)_z S_4 \\ (y_{O_4})_1 = 0 \\ (z_{O_4})_1 = (l_1)_z - l_2 S_2 - l_3 S_3 - (l_4)_x S_4 + (l_4)_z C_4 \end{cases}$$

Équation 19 : position du repère 5

Finalement, nous pouvons obtenir la matrice de transformation homogène (Équation 20).

$${}^0\mathbf{T}_1 = \begin{bmatrix} C_1 & -S_1 & 0 & X_{O_1} \\ S_1 & C_1 & 0 & Y_{O_1} \\ 0 & 0 & 1 & R \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Équation 20 : matrice de transformation homogène

Les positions dans le repère fixe peuvent ainsi être calculées (Équation 21, Équation 22, Équation 23 et Équation 24).

$$\begin{cases} X_{O_2} = X_{O_1} + (l_1)_x C_1 \\ Y_{O_2} = Y_{O_1} + (l_1)_x S_1 \\ Z_{O_2} = R + (l_1)_z \end{cases}$$

Équation 21 : position absolue du repère 2

$$\begin{cases} X_{O_3} = X_{O_1} + ((l_1)_x + l_2 C_2) C_1 \\ Y_{O_3} = Y_{O_1} + ((l_1)_x + l_2 C_2) S_1 \\ Z_{O_3} = R + (l_1)_z - l_2 S_2 \end{cases}$$

Équation 22 : position absolue du repère 3

$$\begin{cases} X_{O_4} = X_{O_1} + ((l_1)_x + l_2 C_2 + l_3 C_3) C_1 \\ Y_{O_4} = Y_{O_1} + ((l_1)_x + l_2 C_2 + l_3 C_3) S_1 \\ Z_{O_4} = R + (l_1)_z - l_2 S_2 - l_3 S_3 \end{cases}$$

Équation 23 : position absolue du repère 4

$$\begin{cases} X_{O_5} = X_{O_1} + ((l_1)_x + l_2 C_2 + l_3 C_3 + (l_4)_x C_4 + (l_4)_z S_4) C_1 \\ Y_{O_5} = Y_{O_1} + ((l_1)_x + l_2 C_2 + l_3 C_3 + (l_4)_x C_4 + (l_4)_z S_4) S_1 \\ Z_{O_5} = R + (l_1)_z - l_2 S_2 - l_3 S_3 - (l_4)_x S_4 + (l_4)_z C_4 \end{cases}$$

Équation 24 : position absolue du repère 5

La dynamique du manipulateur mobile est décrite en Annexe A.

4.4 Bird Of Prey (BOP)

4.4.1 Introduction

Comme son nom l'indique, Bird of Prey est un système de vision inspiré des oiseaux de proies. Certains oiseaux peuvent repérer précisément leurs proies à plusieurs dizaines de mètres. Grâce à un mouvement vif de la tête, ils peuvent estimer précisément la position relative de leur cible. Nous nous sommes inspirés de cette technique pour créer BOP. En effet, le système est équipé de deux caméras. Ces deux caméras sont montées sur un châssis mobile permettant l'acquisition de deux séries images. Nous disposons donc au final de quatre images prises de quatre angles de vue différents. En connaissant les positions relatives des 4 prises de vues, il est alors possible de calculer précisément la position d'un objet fixe dans l'environnement. Mais cette technique permet également de connaître la position et la vitesse d'un objet mobile. Mathématiquement, une acquisition de deux images permet de connaître la position d'un objet dans l'espace, donc deux acquisitions successives donnent deux positions espacées d'un écart de temps Δt. Avec ces deux positions, il est alors très simple de calculer la vitesse instantanée de l'objet traqué.

4.4.2 Description générale

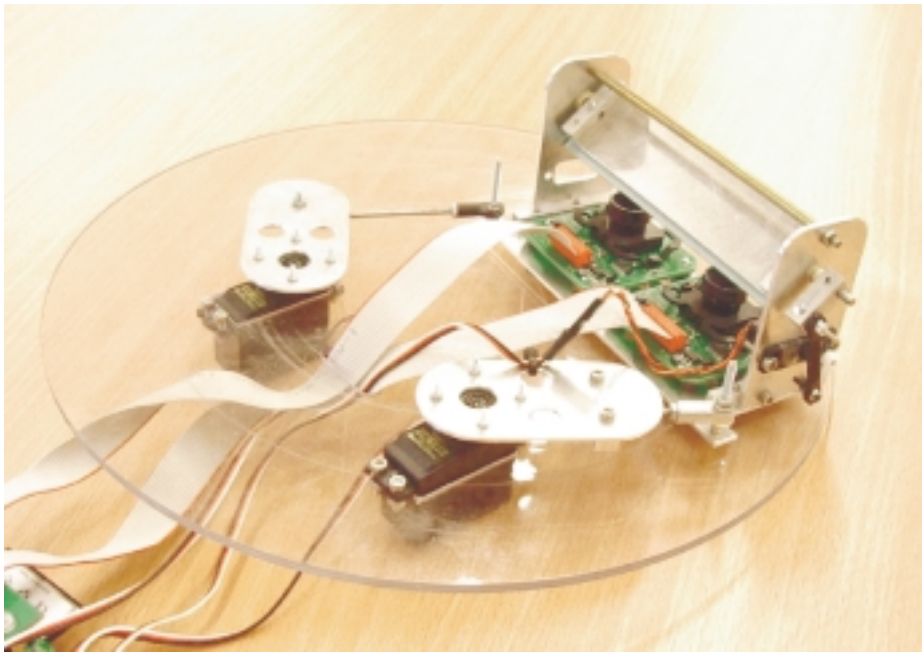


Figure 30 : Le système de vision stéréoscopique BOP

La Figure 30 présente une vue d'ensemble du système. Les deux caméras sont fixées l'une par rapport à l'autre par l'intermédiaire de la platine. Les objectifs des deux caméras sont orientés vers le haut, un miroir mobile situé au-dessus des caméras permet d'orienter le champ de vision. Ce système permet de modifier l'angle de vue rapidement sans avoir à bouger les caméras. La Figure 31 montre un schéma du mécanisme terminal. Le miroir est actionné par un servomoteur miniature. Pour alléger au maximum les pièces mobiles, cet actionneur devait être fixe sur le châssis et la transmission devait se faire par l'intermédiaire d'un câble. Mais, étant donné le poids d'un servomoteur miniature (8 grammes), nous avons choisi de l'embarquer sur la platine. La transmission jusqu'au miroir se fait via une petite bielle.

Pour que le système puisse être viable, il faut que le temps séparant deux acquisitions soit le plus court possible. Afin d'augmenter la dynamique du système, nous avons fait le choix de n'embarquer aucun moteur sur les parties mobiles (à l'exception du servomoteur miniature présenté précédemment). Une architecture mécanique du type série était donc à exclure. Nous nous sommes inspirés des mécanismes parallèles: l'organe terminal est mu par deux servomoteurs qui sont fixes sur le châssis. Le mécanisme forme une chaîne fermée constituée de la séquence suivante :

- un servomoteur (servomoteur 1),
- un axe fixe sur le servomoteur (axe 1),
- une liaison pivot reliant l'axe et l'organe terminal,
- l'organe terminal,
- une liaison rotule reliant l'organe terminal et l'axe suivant,
- un axe (axe 3),
- une liaison rotule reliant les axes 2 et 3
- un axe fixé sur le servomoteur 2
- et le servomoteur 2.

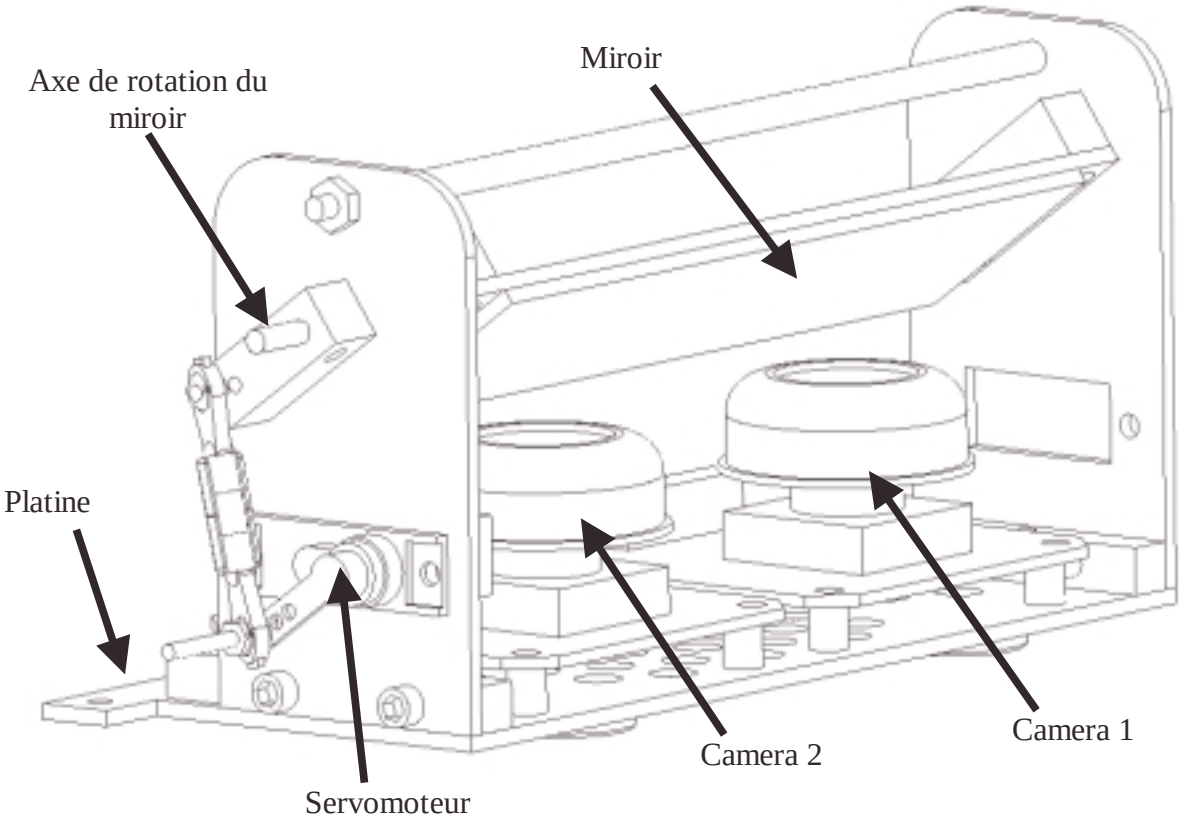


Figure 31 : Organe terminal

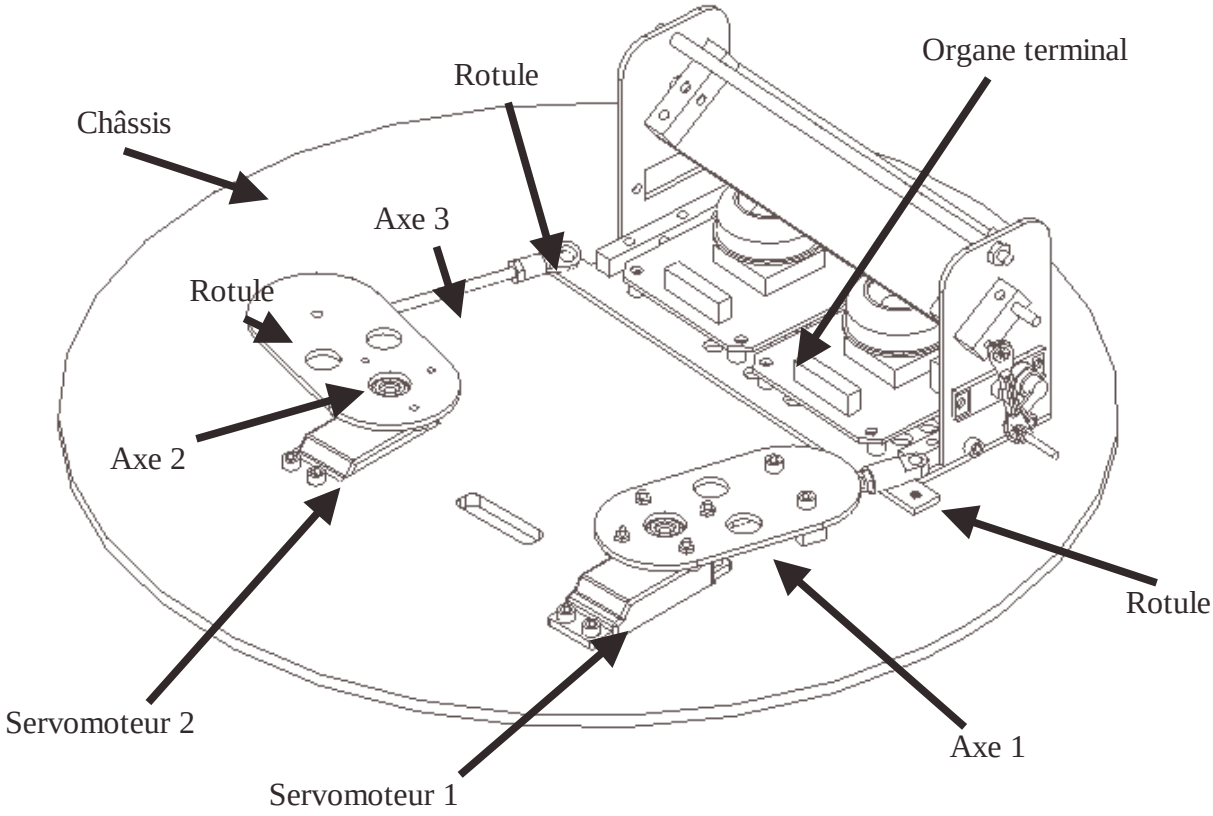


Figure 32 : Schéma global de notre système de vision BOP

Un schéma de l'ensemble du mécanisme est proposé sur la Figure 32. Le servomoteur 1 permet de positionner l'organe terminal. Le servomoteur 2 déforme le parallélogramme formé par les axes afin de modifier l'orientation de l'angle de vue. Un exemple d'acquisition est présenté sur la Figure 33 : Exemple de positions d'acquisition. Les deux actionneurs permettent de modifier à la fois la position de l'organe terminal, mais aussi son orientation. Le mécanisme étant de petite taille (le diamètre du châssis est de 30 cm), nous avons choisi d'utiliser du matériel de modélisme pour la réalisation de ce premier prototype. Les servomoteurs utilisés présentent des accélérations suffisantes pour permettre au système d'obtenir la dynamique désirée.

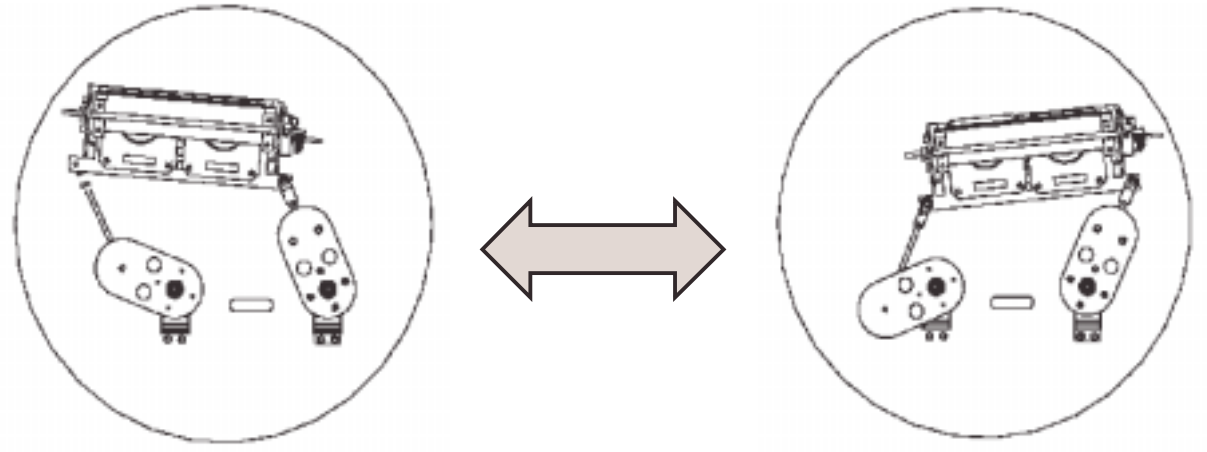


Figure 33 : Exemple de positions d'acquisition

4.4.3 Positions et orientations

La Figure 34 montre la disposition des différents points et repères. Afin de ne pas alourdir le schéma, les longueurs des axes n'ont pas été reportées sur la figure. Voici les principales longueurs utilisées dans la suite des calculs :

- l_1 : distance entre l'origine O et le point M_1,
- l_2 : distance entre le point M_1 et le point M_2,
- l_3 : distance entre le point M_2 et le point M_3,
- l_4 : distance entre le point M_3 et le point M_4,
- l_5 : distance entre le point M_4 et le point M_5,
- l_6 : distance entre le point M_6 et l'origine O.

Le calcul des positions et des orientations se fait entièrement dans le repère fixe (O, X, Y). L'objectif est de calculer les coordonnées des points M_3 et M_2. Commençons par calculer la position des points M_1, M_2, M_4 et M_5 (Équation 25).

$$\begin{cases} X_{M1} = l_1 \\ Y_{M1} = 0 \end{cases} \quad \begin{cases} X_{M5} = -l_6 \\ Y_{M5} = 0 \end{cases}$$

$$\begin{cases} X_{M2} = X_{M2} + l_2.\sin(\theta_1) \\ Y_{M2} = Y_{M2} + l_2.\cos(\theta_1) \end{cases}$$

$$\begin{cases} X_{M4} = X_{M5} + l_5.\sin(\theta_2) \\ Y_{M4} = Y_{M5} + l_5.\cos(\theta_2) \end{cases}$$

Équation 25 : Coordonnées des points M_1, M_2, M_4 et M_5

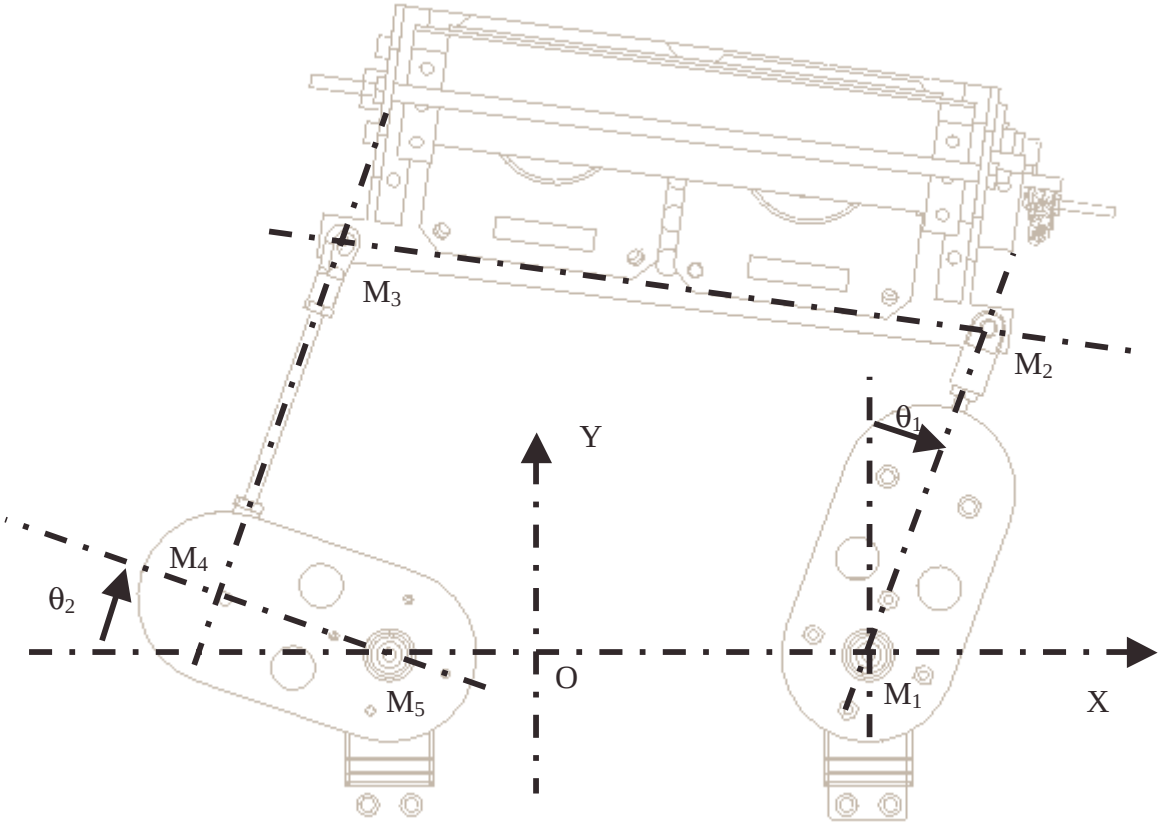


Figure 34 : Disposition des différents points et repères

Le problème réside dans le calcul des coordonnées du point M3. La méthode consiste à calculer l'intersection des deux cercles de centres M_4 et M_2. Ces cercles ont des rayons respectifs de longueur l_4 et l_3. Les équations des cercles sont données par l'Équation 26.

$$\begin{cases} (X_{M3} - X_{M4})^2 + (Y_{M3} - Y_{M4})^2 = l_4^2 \\ (X_{M3} - X_{M2})^2 + (Y_{M3} - Y_{M2})^2 = l_3^2 \end{cases}$$

Équation 26 : Equation des deux cercles de centre M_4 et M_2

La résolution de ce système de deux équations à deux inconnues permet de trouver les coordonnées de l'Équation 27.

$$\begin{cases} X_{M3}=c-bY_{M3} \\ Y_{M3}=\dfrac{-B+\sqrt{B^2-4AC}}{2A} \end{cases}$$

avec :

$$a=2X_{M2}-2X_{M4}$$
$$b=\frac{(2Y_{M2}-2Y_{M4})}{a}$$
$$c=\frac{(l_2^2-l_3^2+X_{M2}^2-X_{M4}^2+Y_{M2}^2-Y_{M4}^2)}{a}$$
$$A=b^2+1$$
$$B=2bX_{M4}-2cbY_{M4}-2Y_{M4}$$
$$C=c^2-2cX_{M4}+X_{M4}^2+Y_{M4}^2-l_2^2$$

Équation 27 : Coordonnées du point M_3

4.5 Conclusion

Afin d'expérimenter nos architectures d'apprentissage dans des conditions réelles, nous avons dessiné et construit notre plate-forme expérimentale. Cette plate-forme est constituée d'un système de vision stéréoscopique (BOP) et de cinq robots mobiles Type 1 dont un est équipé du manipulateur mobile miniature (M^3). Chacun de ces robots a été décrit en détail dans ce chapitre. Bien sûr, les méthodes présentées dans les chapitres suivants n'ont pas été implémentées directement sur la plate-forme expérimentale. Les algorithmes ont été testés en premier lieu en simulation. Un simulateur 2D a été programmé sous Matlab afin de simuler le comportement d'un ou plusieurs robots mobiles. Ce simulateur permet principalement de gérer les communications entre les robots, les 8 capteurs infrarouges la détection d'obstacles et les collisions. Le déplacement des robots est géré par le modèle géométrique du robot Type 1, deux consignes sont appliquées aux moteurs droit et gauche. Ce simulateur a principalement été conçu pour disposer de résultats rapidement afin de pouvoir comparer les méthodes et les réglages de paramètres.

CHAPITRE

5

5. Algorithmes évolutionnistes pour les systèmes multi-robots homogènes

5.1 Introduction

Ce chapitre va s'intéresser à l'apprentissage de comportements pour des ensembles de robots homogènes. Tous les agents formant le système multi-robots seront strictement identiques et disposeront des mêmes capacités de perception et d'action. Les systèmes multi-agents distribués peuvent facilement tirer parti du parallélisme du système. C'est-à-dire que l'on peut diviser une tâche complexe et distribuer à chaque agent une partie de la problématique. Ici, c'est la tâche d'apprentissage qui sera distribuée. Quelques travaux antérieurs considèrent le temps de calcul comme le paramètre à optimiser. Or, dans le cas d'apprentissage appliqué sur des systèmes réels, ce temps de calcul est négligeable par rapport au temps global nécessaire à l'apprentissage. Dans une expérimentation sur robots réels, le point le plus long est l'estimation de la performance de chaque comportement. Prenons l'exemple des algorithmes évolutionnistes : le point le plus long dans l'apprentissage ne sera pas la réalisation des croisements ou des mutations, mais l'estimation de la population. Pour

estimer la performance d'un comportement, il faut s'assurer que le système a rencontré suffisamment de configurations possibles pour obtenir une performance significative. Dans le cas d'une tâche d'évitement d'obstacles, il faut que la durée d'une expérience soit suffisamment longue pour rencontrer un maximum de configuration d'obstacles (obstacle devant, à droite, derrière etc.). Ce temps d'expérimentation est relatif à une durée mécanique, alors que les temps de calcul de l'algorithme d'apprentissage sont relatifs à une durée logicielle. Le plus long dans l'apprentissage sera bien l'estimation de la performance.

Afin de diminuer les durées d'apprentissage, le parallélisme des systèmes multi-agents doit être utilisé à profit. Et ce qui est distribué sur l'ensemble des agents sera l'élément le plus coûteux (en temps) de l'apprentissage, c'est-à-dire l'estimation de la performance. Ce qui revient à distribuer à chaque individu de la population un comportement à tester. Il semble donc que les algorithmes évolutionnistes soient la méthode la plus à-même pour réaliser l'apprentissage dans le cas des systèmes multi-robots homogènes. Il faut toutefois ne pas s'éloigner des hypothèses de départ, à savoir que le système ne doit pas être supervisé. Or, les algorithmes évolutionnistes tel qu'ils ont été formulés par D. Goldberg, demandent de centraliser les séquences génétiques. Un algorithme évolutionniste distribué sera présenté dans la deuxième partie de ce chapitre.

Des travaux antérieurs ont déjà été menés sur l'utilisation des algorithmes évolutionnistes appliqués à l'apprentissage de comportements réactifs. Bien que ces travaux aient été menés avec une approche supervisée, la pertinence des résultats obtenues justifie que ces travaux soient présentés dans la première partie de ce chapitre.

5.2 Un algorithme évolutionniste supervisé

5.2.1 Introduction

Les résultats présentés dans cette première partie ont été publiés en 1994 par Dario Floreano et Franscesco Mondada. L'ensemble des résultats et figures présentées ici est extrait d'un article de Dario Floreano [Floreano 94]. L'expérience consiste à faire apprendre un comportement réactif à un robot mobile, il s'agit en l'occurrence d'une tâche d'évitement d'obstacles. La méthode d'apprentissage utilisée est basée sur les algorithmes génétiques.

5.2.2 Description de la tâche

Un seul robot sera utilisé pour l'expérience. Afin de pouvoir expérimenter de grandes populations d'individus, les comportements des 80 individus sont testés successivement sur le même robot. Il s'agit du robot mobile Khepera [Mondada 89] montré sur la Figure 35. Ce robot est de forme cylindrique, son diamètre est de 55 mm. Sa petite taille permet de faire évoluer plusieurs robots dans des environnements de taille restreinte. Ici l'unique robot évolue dans un environnement de 80x50 cm, montré sur la Figure 36.

L'objectif est donc d'évoluer en sécurité dans cet environnement c'est-à-dire d'éviter les obstacles et d'avancer le plus rapidement possible. La récompense instantanée attribuée à l'agent est donnée par l'Équation 28. Ce type de récompense instantanée a été discuté dans le chapitre 3.

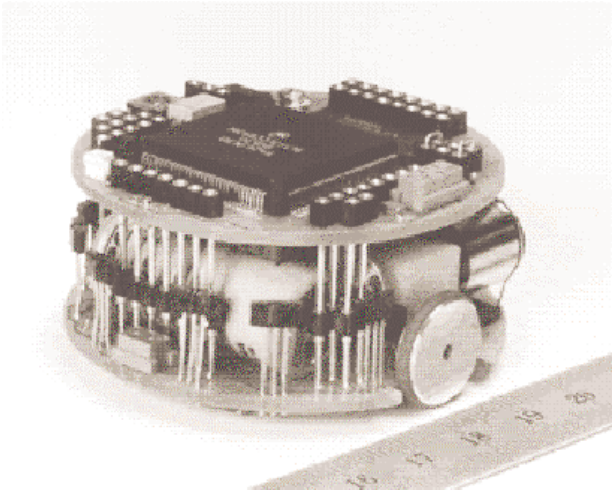


Figure 35 : Le robot mobile Khepera

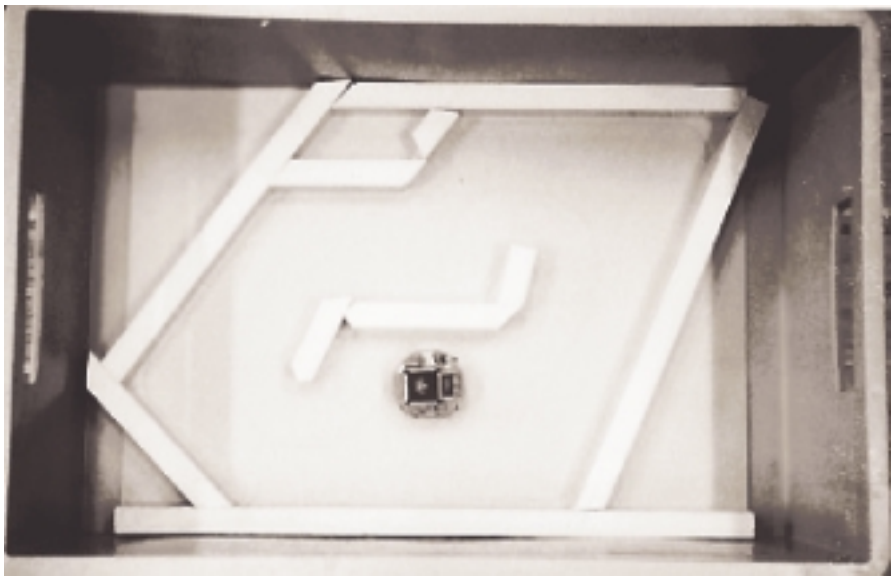


Figure 36 : L'environnement de l'expérience (D'après [Floreano 94])

$$R_i = V.(1-\sqrt{\Delta v}).(1-i)$$

où :

- V est la vitesse moyenne de rotation des deux roues,
- Δv est la différence signée de la vitesse des deux roues,
- i est une valeur proportionnelle à la quantité d'obstacles proches du robot.

Équation 28 : Récompense instantanée attribuée à l'agent (D'après [Floreano 94])

Le comportement de l'agent est régi par un contrôleur neuronal. Le but de l'apprentissage est donc de déterminer les poids synaptiques optimaux de ce réseau de neurones. Les poids sont codés dans la séquence chromosomique de chaque agent.

Voici les paramètres utilisés pour l'algorithme génétique :

Taille de la population	: 80 individus
Nombre de générations	: 100
Probabilité de croisement	: 0.1
Probabilité de mutation	: 0.2
Intervalle de mutation	: ± 0.5
Intervalle initial des poids	: ± 0.5
Intervalle final des poids	: non borné
Durée de vie d'un individu	: 80 actions
Durée d'une action	: 300ms

5.2.3 Résultats

Le robot mobile Khepera a été capable d'apprendre une stratégie d'évitement d'obstacles en moins de 100 générations. L'évolution de la performance est montrée sur la Figure 37. L'estimation d'une génération dure environ 39 minutes. Après 50 générations environ, les meilleurs individus ont développé une stratégie proche de l'optimale, leurs trajectoires sont fluides et ils évitent les obstacles. Ce qui leur permet de faire le tour complet du corridor. La disposition des capteurs n'est pas symétrique sur un robot Khepera. L'avant du robot dispose de six capteurs et l'arrière seulement deux. La fonction de récompense ne prenant pas en compte la vitesse signée du robot, aucun sens de déplacement n'est imposé au robot. Et pourtant, une direction prioritaire émerge au fil des générations, le robot se déplace dans la direction la mieux représentée par les capteurs.

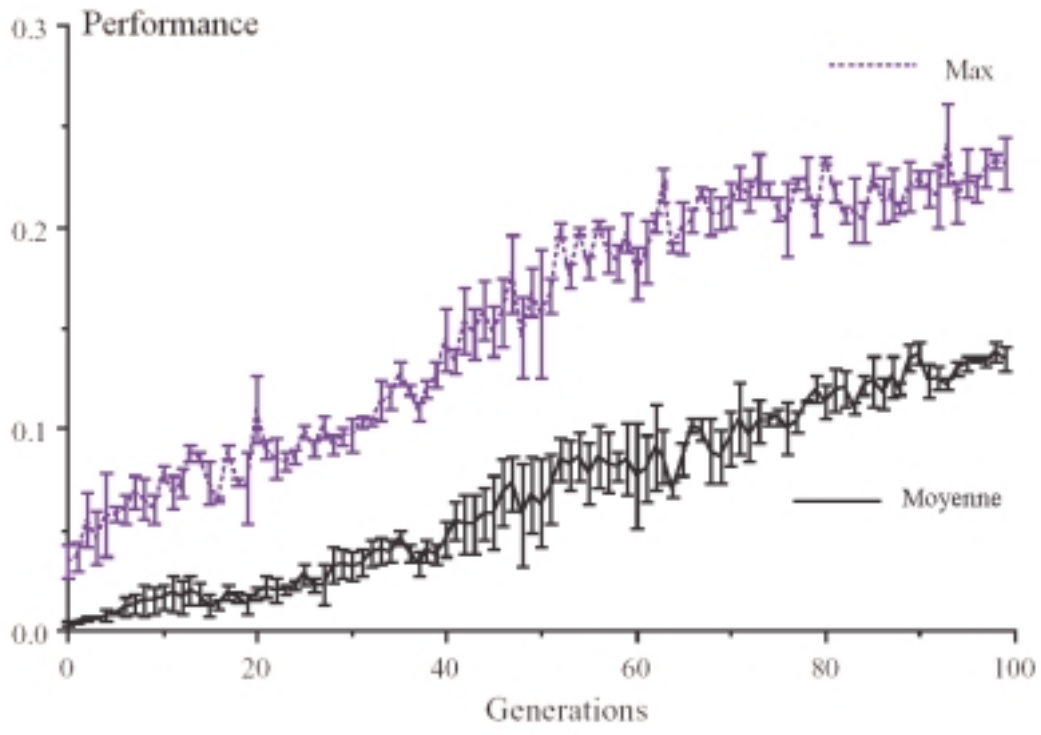


Figure 37 : Performance moyenne de la population et du meilleur individu en fonction des générations (D'après [Floreano 94])

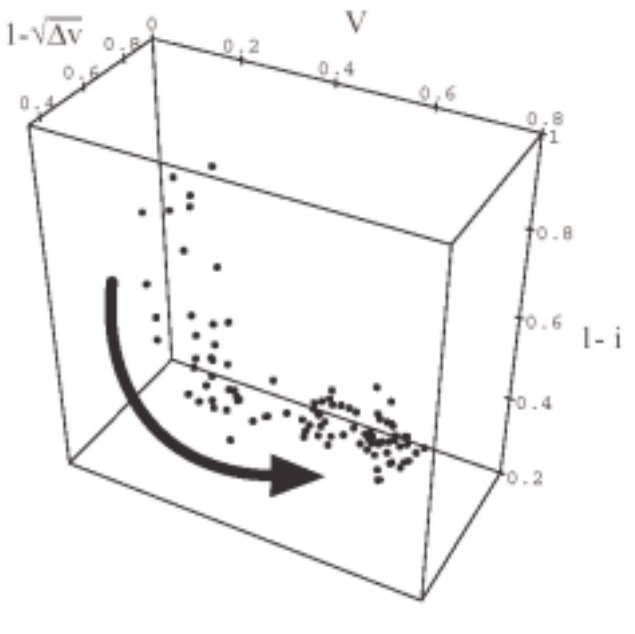


Figure 38 : Evolution de chacun des paramètres de la récompense au fil des générations (D'après [Floreano 94])

Les trois composantes de la récompense ont été découplées afin de pouvoir étudier l'évolution de l'apprentissage (Figure 38). Chaque point représente la position dans l'espace d'état du meilleur agent de la génération. Dans un premier temps, les agents cherchent à diminuer le nombre de collisions. En réalité le contrôleur neuronal sait de mieux en mieux discriminer les obstacles des zones libres. Une fois que l'agent sait se déplacer en évitant les obstacles, il incrémente progressivement la vitesse de déplacement en conservant ses propriétés d'évitement d'obstacles. D'autres résultats montrent également la stabilité du système. L'expérience consiste à écarter les comportements du point d'adaptation et à observer la réaction du système: les agents retournent automatiquement dans la position initiale d'équilibre.

5.2.4 Conclusion

Nous venons de présenter des travaux et résultats antérieurs, qui démontrent déjà la possibilité d'appliquer des méthodes évolutionnistes sur des ensembles de robots mobiles. Bien que ces résultats expérimentaux n'aient été obtenus que sur un seul robot mobile, cela n'enlève en rien aux conclusions scientifiques relatives aux systèmes multi-robots. D'ailleurs, disposer de 80 robots évoluant en parallèle dans des environnements différents ne changerait pas les résultats.

Ces résultats montrent qu'une population d'agents peut apprendre un jeu de stratégies et de comportements en s'adaptant à sa propre structure et à son environnement. Deux critères principaux ont émergé : la sélection naturelle d'un sens de déplacement et l'adaptation automatique de la vitesse de l'agent. L'agent a naturellement déterminé la vitesse optimale de déplacement permettant de maximiser la récompense, sans pour autant percuter d'obstacles. A la vue de ces résultats, l'analogie avec le vivant se fait clairement ressentir. Et c'est une fois de plus le vivant qui va nous inspirer la suite de ses travaux. Afin de pouvoir appliquer ces méthodes sur des systèmes multi-robots, une version distribuée des algorithmes évolutionnistes est proposée dans la deuxième partie de ce chapitre.

5.3 Un algorithme évolutionniste distribué

5.3.1 Introduction

L'expérience menée par Dario Floreano et Franscesco Mondada montre la possibilité d'apprendre des comportements réactifs en utilisant les algorithmes génétiques. La méthode employée est dite supervisée, elle nécessite de centraliser l'ensemble des chaînes chromosomiques afin de produire la génération suivante. Une panne du système central engendre immédiatement une panne globale sur l'ensemble du système. De plus, si le système de communication n'est pas global, les agents doivent se rassembler pour transmettre leurs codes génétiques. Les méthodes supervisées peuvent s'appliquer sans problème aux systèmes simulés ou logiciels, mais dans le cas des systèmes multi-robots, la robustesse n'est pas garantie.

Cette méthode est directement inspirée de l'hypothèse d'évolution des espèces proposée par Darwin. La population actuelle de la terre est estimée à 6 milliards d'individus. Pour créer une nouvelle génération, les 6 milliards de séquences ADN ne sont pas réunies pour produire 6 milliards de nouveaux individus. D'ailleurs, la taille de la population peut varier sans pour autant perturber le système. Afin de conserver les propriétés inhérentes aux systèmes distribués, c'est une fois de plus dans le vivant que nous allons trouver une nouvelle source d'inspiration. Lorsque deux individus vont se rencontrer et que toutes les conditions seront réunies, leurs chaînes chromosomiques seront échangées pour créer deux nouveaux individus.

5.3.2 Description de la tâche

La tâche étudiée est basée sur la navigation d'un ensemble de robots mobiles dans un environnement inconnu. Il ne s'agit en aucun cas de construire une carte ou une quelconque représentation de l'environnement, mais de déterminer un ou plusieurs comportements réactifs permettant aux robots de naviguer dans l'environnement en toute sécurité. Les expériences ont été menées sur les robots mobiles Type 1. Les entrées du système sont les capteurs de distances et les sorties, les deux moteurs du robot. Afin de diminuer les temps d'apprentissage les entrées et sorties du système ont été grossièrement discrétisées. Les entrées du système sont rassemblées en 5 états et les sorties en 4 actions. L'objectif de l'apprentissage étant d'associer à chaque état la meilleure action. Les différents états sont présentés sur le Tableau 4 et les actions sur le Tableau 5.

Ce système présente 1024 (4^5) combinaisons, néanmoins l'objectif n'est pas de montrer que les algorithmes évolutionnistes peuvent résoudre des problèmes complexes. Mais simplement de montrer qu'il existe des alternatives distribuées aux méthodes supervisées. Si les hypothèses considérées avaient été exactement les mêmes que pour l'expérience menée à l'EPFL décrite précédemment, il aurait fallu mettre en œuvre simultanément 80 robots. En supposant que la relation liant les temps d'apprentissage au nombre de robots et de générations soit linéaire, la durée des accus ne nous aurait pas permis de dépasser la 11ème génération avec 5 robots. Alors qu'il en faut une centaine pour obtenir la convergence. C'est pourquoi nous avons choisi de discrétiser les états et actions.

Etat 1	Pas d'obstacle
Etat 2	Un obstacle à gauche
Etat 3	Un obstacle à droite
Etat 4	Un obstacle devant
Etat 5	Le robot est bloqué

Tableau 4 : Différents états du système

Action 1	Avancer
Action 2	Tourner à droite
Action 3	Tourner à gauche
Action 4	Reculer

Tableau 5 : **Jeu d'actions possibles**

La chaîne chromosomique est une concaténation de N mots {0,1} dans le sous-espace M. N est le nombre d'entrées et M le nombre de sorties. Bien sûr il n'y a qu'un seul 1 dans chaque espace N (Figure 39). Les générations de robots vont évoluer selon les transformations de ces chaînes embarquées sous l'action des opérateurs génétiques.

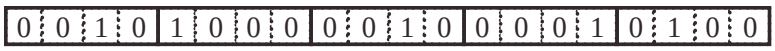


Figure 39 : Un exemple de chaîne chromosomique

5.3.3 Les opérateurs

Dorénavant, les croisements ne se font plus sur l'ensemble de la population, mais sur un couple d'individus. Il a donc fallu réviser les conditions et les opérateurs de mutations et de croisements.

Initialisation : au début, les différentes chaînes chromosomiques peuvent être remplies aléatoirement ou toutes initialisées à la même valeur. Dans les expériences décrites dans le présent chapitre, les chaînes sont initialisées aléatoirement.

Estimation de la performance individuelle : Dans les algorithmes génétiques classiques et d'autres techniques d'apprentissage, c'est parfois la couche supérieure qui calcule la performance de chaque individu. Ces individus sont alors classés avant d'y appliquer les opérateurs génétiques. Ici, au contraire, comme nous recherchons une estimation autonome de la performance, une capacité d'auto-évaluation locale de la performance est attribuée à chaque individu qui n'est jamais conscient de la performance globale de l'ensemble de la population.

En suivant les principes de base des algorithmes d'auto-apprentissage, chaque individu calcule sa propre récompense courante (Équation 29).

$$R_{N(i)}(i) = (1-\alpha(i))R_{N(i)-1}(i) + \alpha(i)F_{N(i)}(i)$$

Où :

$$\alpha(i) = \frac{1}{1+N(i)}$$

N(i) est le nombre de pas élémentaires depuis le début de l'estimation par l'agent *i*
$R_{N(i)}(i)$ est la récompense courante estimée à l'instant *N(i)*
$F_{N(i)}(i)$ est la récompense instantanée à l'instant *N(i)*

Équation 29 : Calcul de la récompense courante de l'agent i

Dans le problème d'exploration étudié ici, la récompense instantanée est calculée comme étant la distance signée parcourue durant un pas élémentaire. De fait, la récompense courante associée à la stratégie en cours (liée à la chaîne chromosomique) est la distance moyenne parcourue depuis l'initialisation ou depuis le dernier changement dans la chaîne chromosomique par croisement ou mutation. Une telle récompense pénalise les trajectoires non rectilignes.

Croisements : la plupart des solutions existantes font généralement appel à une communication globale entre les agents. De cette façon, l'approche traditionnelle est utilisée : la sélection est réalisée sur l'ensemble de la population. Afin de contourner les problèmes liés à cette approche, la solution proposée ici est basée sur des communications locales. De cette façon, un couple de robots se rencontrant peut ainsi réaliser des croisements. Il résultera de ces croisements deux nouveaux individus. Les conditions formelles sont les suivantes :

- la distance inter-robots est faible,
- les deux robots n'ont pas récemment réalisé de mutations ou de croisements.

La première condition est nécessaire pour garantir les communications entre les robots. La seconde évite qu'un robot ne change sa chaîne chromosomique avant d'avoir complètement évalué sa stratégie courante durant une durée suffisante. Quand les croisement sont réalisables, les agents i et j disposent de deux nouvelles chaînes chromosomiques selon la probabilité donnée par l'Équation 30.

$$P(i) = \frac{R_{N(i)}(i)}{R_{N(i)}(i) + R_{N(j)}(j)}$$

Équation 30 : Probabilité que la chaîne i se retrouve dans les générations futures suite à un croisement entre les agents i et j

Les croisements ne sont pas suffisants pour assurer la convergence du système vers la solution globalement optimale, particulièrement lorsque la taille de la population est faible. Dans ce cas, les agents peuvent être attirés par un minimum local et la population restera

bloquée dans cet état final. Les mutations sont donc nécessaires pour extirper le système des solutions locales et pour explorer plus largement l'espace d'état.

Mutations : dans les algorithmes génétiques traditionnels, les mutations sont réalisées de façon aléatoire. Ici, cela pourrait être un inconvénient, puisque l'objectif est de réaliser un apprentissage de type "on-policy". Il n'est pas admissible qu'un robot puisse changer sa stratégie pour une moins bonne et attende longtemps pour retrouver une chaîne chromosomique acceptable par croisement ou mutation. Pour résoudre ces problèmes, la stratégie de mutation suivante a été adoptée :

- Les agents n'ont pas réalisé de mutation récemment
- La performance courante de l'agent est faible

Comme pour les croisements, la première condition garantit que l'agent dispose de suffisamment de temps pour estimer correctement sa propre performance. La seconde condition évite de dégrader une stratégie qui serait correcte. La probabilité de mutation d'un agent est une fonction de l'estimation courante de la performance. Cela permet de garantir que les stratégies les plus mauvaises ont davantage de chances de muter, contrairement aux bonnes stratégies qui, elles ont une faible probabilité (non nulle, ce qui leur permet de s'extraire des minimum locaux).

5.3.4 Protocole de communication

Différents protocoles de communication ont été présentés dans le chapitre consacré aux systèmes multi-agents. Tous ces protocoles sont globaux, or, ici les agents ne réalisent que des communications locales. De plus, les robots utilisent les mêmes capteurs pour les communications et pour la détection d'obstacles. Il a donc été nécessaire de proposer un protocole dédié à cette application. Voici les hypothèses inhérentes au système :

• pas de synchronisation,
• utiliser la même bande passante pour les capteurs et pour les communications.

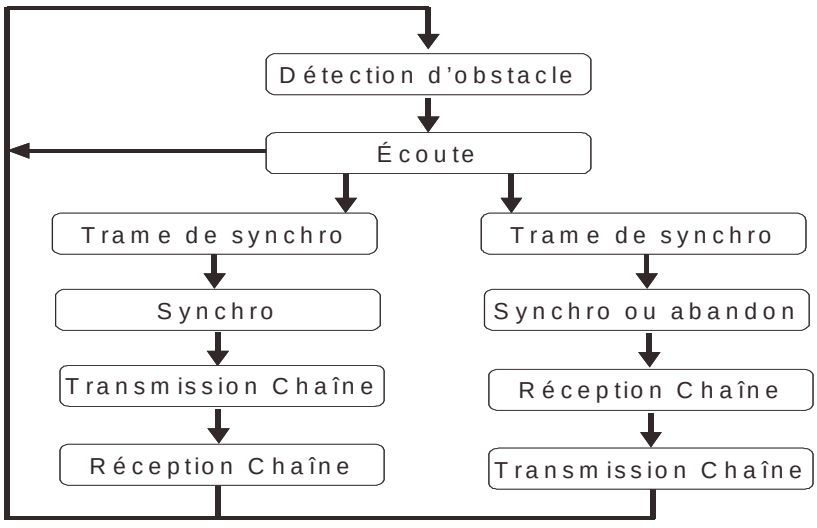


Figure 40 : Détail du protocole de communication.

Une trame du protocole commence systématiquement par une détection d'obstacle, la carte locale est mémorisée afin de déterminer l'état courant du système. Ensuite vient une phase d'écoute (l'agent n'émet aucune information). Si l'agent ne détecte aucun autre agent, il termine son cycle sensori-moteur. Si des infrarouges sont perçus durant la phase d'écoute, cela signifie que l'agent n'est pas seul. Deux possibilités apparaissent alors :

- Les informations reçues sont décodées comme une trame de synchronisation valide, l'agent émet sa propre trame, puis les deux agents s'échangent leurs chaînes chromosomiques et leurs estimations de la performance.
- Les informations reçues ne sont pas décodées comme une trame de synchronisation valide, l'agent émet sa propre trame et attend celle de l'autre agent. Une fois la synchronisation réussie, les deux agents s'échangent leurs chaînes chromosomiques et leurs estimations de la performance.

Ce protocole ne nécessite pas de synchronisation globale entre les agents. Une synchronisation s'effectue entre les deux agents sur le premier front descendant valide durant la réception de la trame de synchronisation.

5.3.5 Résultats

Les expérimentations ont été réalisées en simulation, mais également sur la plate-forme réelle. Le simulateur a été réalisé sous Matlab, ce qui nous permet de réaliser une analyse rapide des résultats. Le code source est composé de deux parties : un environnement rectangulaire composé d'objets fixes et les agents mobiles. Les agents ne calculent jamais ni leur position courante, ni leur orientation dans l'environnement. La connaissance de ces variables n'est utilisée que pour l'affichage. Les capteurs sont simulés et offrent une perception locale aux agents. Le modèle géométrique du robot Type 1 est implémenté dans le simulateur. Une capture d'écran du simulateur est montrée sur la Figure 41 (Les zones délimitées en pointillés montrent la distance de communication entre les agents.).

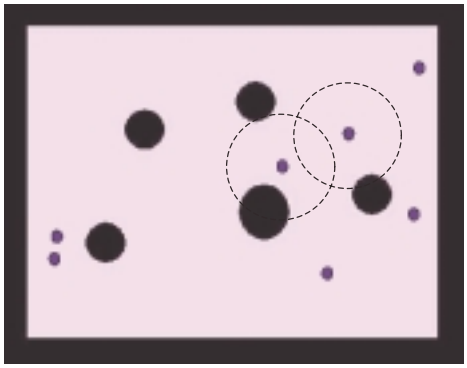


Figure 41 : Capture d'écran du simulateur avec des obstacles circulaires

Les mêmes expériences ont été réalisées sur la plate-forme expérimentale (Figure 42). Le site expérimental mesure 4.80 x 3.60 m. Quatre robots autonomes Type 1 ont été utilisés pour l'expérience. Les obstacles peuvent être déplacés, ajoutés ou supprimés. La vitesse maximale des robots est de 1m/s, mais elle a été limitée à 0.3m/s pour protéger le matériel des collisions violentes. La portée des infra-rouges utilisés pour les capteurs et pour les

communications peut être ajustée. Une portée typique de 0.5m a été adoptée : ce qui permet un compromis entre le besoin de réaliser des croisements et la sécurité quant à la distance de détection des obstacles.

Figure 42 : Photo d'une expérimentation

Pendant un cycle sensori-moteur élémentaire, chaque agent met à jour son état courant et continue de réaliser des mutations et des croisements (si cela est possible). Les simulations ont été utilisées pour étudier l'influence de chaque paramètre sur le système, en commençant par le délai de mutation. Si celui-ci est trop court, les mutations sont réalisées fréquemment, l'espace d'état est rapidement exploré, mais les estimations sont erronées. Au contraire, lorsque le délai est long, les estimations sont excellentes, mais l'exploration de l'espace d'état est longue. Le délai entre deux croisements a également été étudié: les conclusions sont proches de celles des mutations. Si le délai est trop court, les agents vont constamment réaliser des croisements avec le même congénère. Au contraire, si ce délai est long, l'exploration de l'espace d'état est fastidieuse. La condition suivante a été choisi pour déterminer le temps minimal entre deux croisements : deux agents qui viennent de réaliser des croisements se sont suffisamment éloignés pour ne pas recommencer juste après. En simulation, 300 cycles sont nécessaires. En considérant les valeurs numériques précédentes, le temps minimum entre deux croisements successifs a été réglé à 3 secondes sur les robots pour les mêmes raisons que précédemment.

La Figure 43 montre la fin d'une simulation. La première observation est que l'ensemble de l'environnement est exploré (en blanc): c'est un comportement émergent dû à la réactivité du système multi-agents.

1 : Pas d'obstacle	1 : Avancer
2 : Un obstacle à gauche	2 : Tourner à droite
3 : Un obstacle à droite	3 : Tourner à gauche
4 : Un obstacle devant	2 ou 3 : Tourner à droite ou à gauche
5 : Le robot est bloqué	4 : reculer

Tableau 6 : Séquence chromosomique optimale

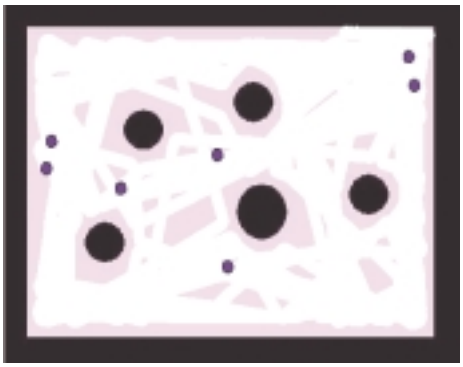


Figure 43 : Exploration de l'environnement

La stratégie optimale est constamment atteinte quelques soient les conditions initiales du système. Un autre paramètre important est la taille de la population, c'est-à-dire le nombre d'agents dans le système. Intuitivement, les temps de convergence semblent inversement proportionnels au nombre d'agents. Les expériences ont été stoppées lorsque l'ensemble de la population a trouvé la solution optimale, c'est-à-dire que tous les individus ont comme chaîne chromosomique la séquence présentée sur le Tableau 6. Pour chaque taille de population, 20 simulations ont été réalisées et le temps de convergence moyen a été enregistré. Les résultats sont présentés sur la Figure 44.

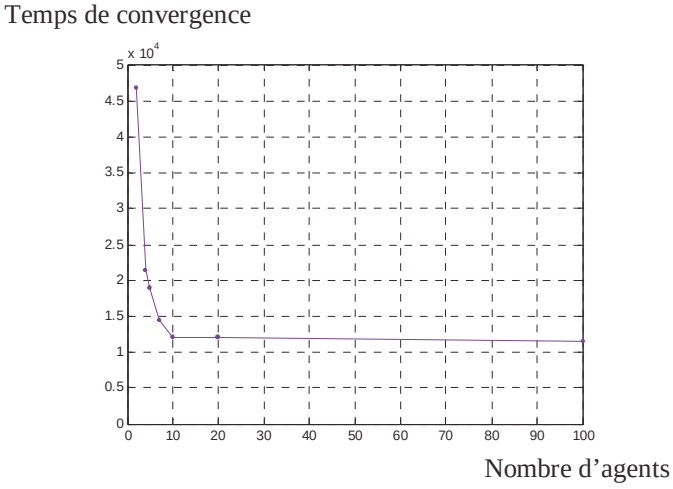


Figure 44 : Temps de convergence en fonction de la taille de la population

La Figure 44 montre qu'en dessous de 10 robots, le temps de convergence moyen décroît rapidement avec le nombre de robots. Au-delà de 10 robots, le temps de convergence devient constant. Ce phénomène est dû à la propagation de la solution optimale : lorsque la population est grande, l'espace d'état est rapidement exploré et la solution optimale est vite trouvée. En revanche, il faut un temps non négligeable pour la propager à travers toute la population et il n'est pas possible de descendre en dessous de ce temps minimal avec un protocole de communication locale.

L'évolution de la stratégie peut se décrire de la façon suivante :

1. Au début, les agents tentent des stratégies aléatoires. Une collision intervient rapidement et le robot est bloqué. La récompense courante décroît, permettant aux mutations de se multiplier jusqu'à ce que l'état 5 (le robot est coincé) soit associé à l'état 4 (Reculer).
2. Les agents évoluent dans l'environnement au gré des mutations et l'un d'eux finit par associer l'état 1 (aucun obstacle) à l'action 1 (avancer). Il parcourt généralement de longues distances dans l'environnement tout en propageant (par croisement) sa chaîne chromosomique aux autres agents.
3. Comme les agents commencent à se déplacer avec fluidité dans l'environnement, les rencontres se multiplient et davantage de croisements sont réalisés. La stratégie optimale est attribuée à un des agents, qui voit sa performance augmenter.
4. Dès qu'un agent réalise la stratégie optimale, il voyage à travers l'environnement sans collision. Il rencontre et transmet sa chaîne chromosomique à d'autres agents, propageant rapidement la stratégie à l'ensemble de la population.

5.4 Conclusion

Ce chapitre a traité de l'apprentissage de comportements réactifs pour les systèmes homogènes. L'hypothèse de départ est la suivante : les méthodes évolutionnistes sont les plus à-même de traiter ce genre de problème. L'estimation de la performance étant la phase de l'apprentissage la plus coûteuse, c'est cette phase qu'il nous est paru judicieux de distribuer. Dans la première partie de ce chapitre, une expérience d'apprentissage évolutionniste mené à l'EPFL a été décrite.

L'objectif était de faire apprendre à un robot mobile une stratégie d'évitement d'obstacles. Le contrôleur du robot est un réseau de neurones et l'apprentissage servait à déterminer les poids synaptiques optimaux permettant une navigation fluide sans collision. L'expérience a montré qu'après l'évolution de 100 générations de 80 individus une stratégie optimale est effectivement trouvée. Cette évolution peut se découpler en paliers qui correspondent chacun à une phase de l'apprentissage. Toutefois la méthode employée est de type supervisée, c'est pour cette raison que nous avons proposé une méthode évolutionniste distribuée.

Dans la deuxième partie de ce chapitre, une méthode totalement distribuée a été proposée. Les croisements et les communications ne se font plus sur l'ensemble de la population, mais sur un couple d'individus. Il a donc fallu redéfinir les opérateurs de croisements, de mutations, mais aussi un protocole de communication distribué et non synchronisé entre les agents. Enfin, les expériences menées en simulation et sur la plate-forme robotisée ont montré l'efficacité et la convergence de la méthode. Ces expériences ont notamment montré qu'il existe un nombre idéal de robots permettant d'optimiser les temps de convergence.

CHAPITRE

6

6. Méthodes d’apprentissage pour les systèmes multi-robots hétérogènes

6.1 Introduction

Dans le chapitre précédent, nous avons étudié et proposé des méthodes d’apprentissage basées sur les algorithmes évolutionnistes pour les systèmes multi-agents homogènes. Ce type d’apprentissage ne peut s’appliquer que sur un ensemble de robots identiques. Si les agents sont différents dans leur structure ou dans leur perception du monde, les chaînes chromosomiques n’auront pas le même sens d’un agent à l’autre. De ce fait, un comportement performant sur un agent ne sera pas nécessairement robuste sur un agent différent. C’est pour ces raisons que les méthodes présentées précédemment ne peuvent être appliquées sur un système hétérogène. Or, pour bon nombre d’applications, l’utilisation d’un système hétérogène peut être nécessaire.

Dans le cas d’un système hétérogène, un apprentissage collectif n’est pas envisageable puisque chaque agent doit apprendre une stratégie qui lui est propre et dépend de sa structure.

C’est pour ces raisons que nous allons nous intéresser maintenant à l’apprentissage de comportements réactifs sur un agent. Nous allons donc isoler un agent des autres et le laisser évoluer seul dans l’environnement. Dans ce type de système, la coopération se fait dans une couche supérieure que nous ne traiterons pas ici, mais des architectures du type Alliance [Parker 94] ou Satisfaction/Altruisme [Simonin 00b] sont parfaitement adaptées pour ce type de coopération.

Précédemment, nous profitions de la taille de la population pour distribuer des solutions à tester à chaque individu afin d’accélérer les temps d’apprentissage. Avec ces nouvelles hypothèses, l’apprentissage ne peut plus être réparti sur une population d’individus. Pour les raisons exposées précédemment, nous nous refusons à créer un modèle cognitif de l’environnement ou de l’agent. La seule solution envisageable pour l’apprentissage consiste à tester successivement les comportements sur le même agent et de conserver les meilleurs. La première méthode présentée dans ce chapitre consiste à apprendre les paramètres d’un contrôleur neuronal en utilisant le recuit simulé. Des résultats utilisant la technique du recuit simulé classique sont présentés. Puis une version étendue de la méthode est décrite, permettant notamment de s’adapter aux pannes et aux diverses perturbations de l’environnement.

Nous allons ensuite présenter une seconde méthode utilisant l’apprentissage par renforcement. Les modèles classiques d’apprentissage par renforcement sont basés sur les processus markoviens. Les stratégies que nous avons à apprendre se placent dans le domaine continu. Or les processus markoviens sont discrets, ils découpent le système en états, actions et transitions. Nous avons alors deux stratégies :

- Nous discrétisons les états et les actions continus afin de pouvoir disposer d’un système discret, ce qui présente deux inconvénients : d'une part il n'est pas possible d'obtenir des comportements fins puisque les actions sont discrétisées, d'autre part si l’on veut un pas d’échantillonnage suffisamment précis, alors la taille des données devient trop grande pour être stockée sur un système embarqué.
- La seconde solution consiste à modéliser le système. Dans notre cas, ce n’est pas tout à fait le processus markovien que nous allons modéliser mais la récompense associée à chaque configuration du système et ceci par un réseau de neurones. Lorsque ce dernier est entraîné, nous recherchons dans ce modèle continu l’action qui nous permet d’espérer la meilleure récompense.

C’est cette seconde stratégie que nous allons étudier en présentant les différents points clef de l’apprentissage : le choix d’un réseau de neurones pour la modélisation, les différentes techniques de recherche du maximum et les résultats.

6.2 Recuit simulé

6.2.1 Introduction

L’idée générale de cette approche consiste à fixer la structure d’un contrôleur réactif bas niveau et à optimiser ses paramètres afin de maximiser la récompense. Pour garantir la portabilité de la méthode, il est préférable de choisir un contrôleur générique. Le choix du contrôleur sera présenté dans le paragraphe suivant. Reste à optimiser les paramètres de notre système pour maximiser la récompense. Ce problème peut se ramener à une recherche de maximum d’une fonction. Dans notre cas, nous considérons que cette fonction est inconnue et

par conséquent sa dérivée aussi. De plus, cette fonction inconnue peut présenter des maxima locaux. L'ensemble des méthodes dérivées de la dichotomie ou de la descente du gradient sont donc à exclure. Nous avons choisi d'utiliser le recuit simulé. Le recuit simulé est une méthode de recherche du maximum inspirée du refroidissement d'un matériau en fusion. Dans l'industrie, cette méthode permet d'obtenir l'orientation des cristaux désirée. Pendant le refroidissement du matériau, l'orientation des cristaux est perturbée. Le principe consiste à remonter la température de fusion si l'orientation est insatisfaisante. La méthode mathématique du même nom permet notamment de trouver le maximum d'une fonction présentant des maxima locaux. Cette méthode de recherche semble la mieux indiquée à notre problème, puisque la connaissance de la fonction n'est pas nécessaire et que la convergence vers le maximum global est assurée sous certaines conditions, relatives notamment à la vitesse de la descente de la température.

Dans cette première partie liée au recuit simulé, nous allons nous focaliser sur une tâche d'évitement d'obstacles. Les hypothèses sont les mêmes que celles présentées au chapitre précédent et nous avons choisi comme récompense de maximiser la vitesse moyenne signée du robot. La récompense instantanée est donc la moyenne des vitesses de rotation des roues du robot.

6.2.1 Structure du contrôleur

Le contrôleur est une fonction mathématique, statistique ou logique qui lie les entrées du système (les capteurs du robot) aux sorties (les actionneurs). Nous avons choisi d'utiliser un contrôleur neuronal car les réseaux de neurones peuvent modéliser la plupart des fonctions de R^n dans R^m. Nous avons utilisé un réseau sans couche cachée, inspiré du véhicule de Braitenberg [Brainteberg 84]. Ce réseau est constitué de deux perceptrons dont les entrées sont les informations directement issues des capteurs et les sorties les consignes appliquées aux moteurs. Le schéma du réseau est montré sur la Figure 45. Le réseau possède 8 entrées et deux perceptrons, il y a aura donc 16 poids synaptiques, ce sont ces 16 paramètres que le recuit simulé permettra d'optimiser. Les travaux antérieurs (dont ceux de Braintenberg) ont montré qu'un réseau sans couche cachée permettait de résoudre ce problème d'évitement d'obstacles. Afin de ne pas allonger les temps d'apprentissage, nous avons choisi de conserver cette structure, mais l'utilisation d'un réseau multi-couches est envisageable pour des problèmes plus complexes.

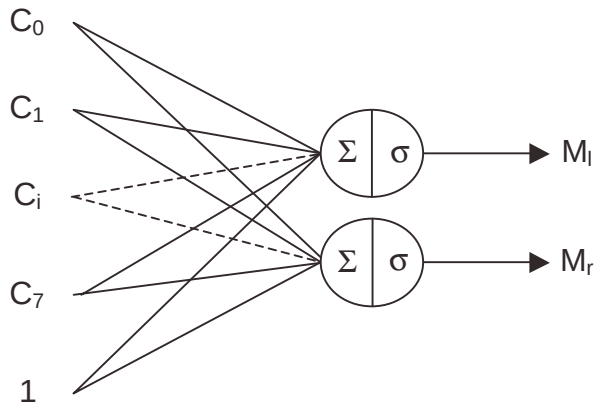


Figure 45 : Le contrôleur neuronal

Pour homogénéiser l'ensemble du système, toutes les valeurs ont été ramenées à l'échelle unitaire. Les informations des capteurs et les commandes appliquées aux moteurs sont comprises entre 0 et 1, les valeurs des poids synaptiques sont bornées dans l'intervalle [-1, 1]. La disposition des capteurs et actionneurs utilisés est représentée sur la Figure 46.

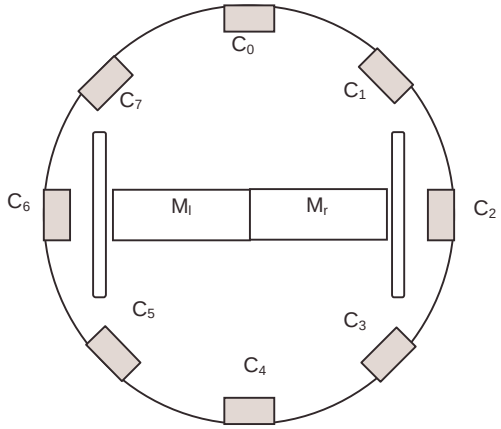


Figure 46 : Disposition des capteurs et actionneurs

6.2.2 Approche classique

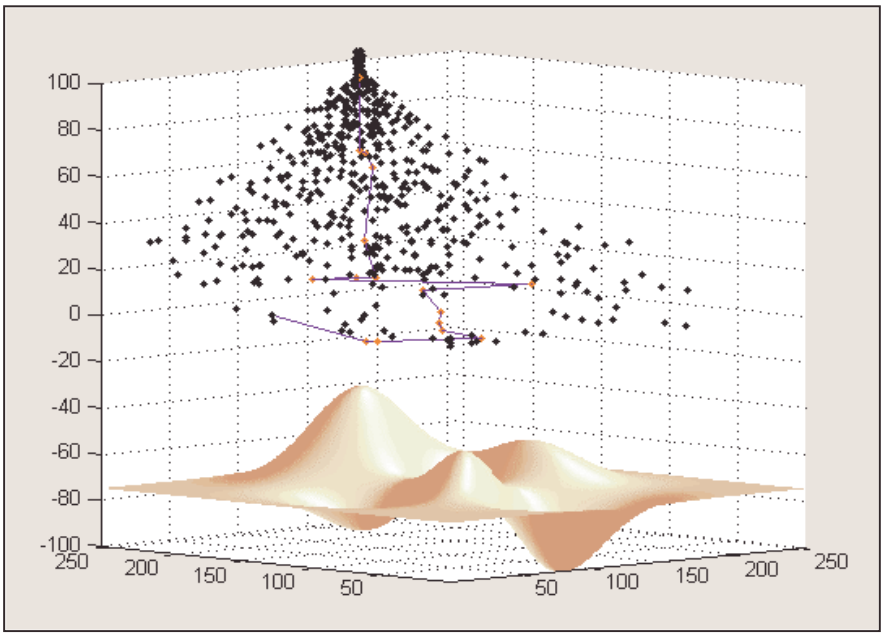


Figure 47 : Recherche du maximum d'une fonction

Nous allons dans un premier temps appliquer la méthode du recuit simulé traditionnel à notre problème. Le principe général est de tirer les valeurs des poids au hasard avec une répartition proportionnelle à la température. Le tirage aléatoire est centré sur la meilleure valeur connue. Un exemple de recherche du maximum est montré sur la Figure 47. L'axe vertical représente le nombre d'itérations pour l'intervalle [0, 100] et la valeur de la fonction à maximiser pour l'intervalle [-100, 0]. L'objectif est de trouver le maximum de la fonction représentée par une surface. La fonction de décroissance de la température est linéaire. Les premiers tirages sont répartis sur l'ensemble de l'espace de définition de la fonction. Le tracé continu représente l'évolution du meilleur point connu. Cette figure montre que l'algorithme

se fait happer par les maxima locaux avant de converger vers le maximum global. Le système n'est jamais complètement gelé pour permettre la convergence vers la valeur exacte du maximum. L'algorithme est détaillé sur l'Algorithme 6.

Initialisation

$R_{max} \leftarrow 0$
Initialiser chaque poids (w_j) avec une petite valeur.
$T° \leftarrow F_t(0)$

Boucle principale

```
Tant que (T° > petite valeur)
{
Appliquer la stratégie courante et mesurer sa performance R_N(i).
Si (R_max<R_N(i))
{
      R_max=R_N(i)
      Pour chaque poids: W_max(j) ← w_j
      }
      T° ← F_t(cycle)
}
Pour chaque poids (w_i), choisir aléatoirement une nouvelle valeur centrée
sur W_max(i) avec une répartition proportionnelle à T°.
}
```

Algorithme 6 : Algorithme utilisé pour entraîner le réseau

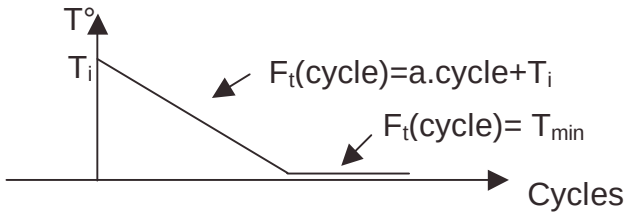


Figure 48 : Evolution de la température en fonction du temps

La fonction de température utilisée est présentée sur la Figure 48. Nous avons arbitrairement choisi une fonction linéaire mais d'autres fonctions auraient pu être testées, toutefois les résultats n'auraient pas été pertinents dans le sens où ils auraient été spécifiques à cette application. Le point essentiel est de ne pas descendre la température trop vite pour permettre au système de s'extirper des minima locaux. Pour permettre à l'algorithme de glisser vers le maximum global, la température n'est jamais égale à 0. La valeur minimale T_{min} garantie de ne jamais figer les paramètres du système et de converger vers l'extremum, qu'il soit optimal ou global.

Un inconvénient majeur à l'utilisation du recuit simulé réside dans le réglage des paramètre: nous allons décrire comment nous avons choisi ceux utilisés dans ces travaux. La fonction de la température est décroissante linéairement jusqu'à atteindre une valeur faible ($T_{min} = 5.10^{-3}$). Cette valeur est suffisante pour verrouiller le système et permettre à l'algorithme de glisser vers la valeur optimale absolue. La température initiale (T_i) est égale à 1. Durant les simulations sous Matlab, nous avons volontairement choisi une valeur négative très faible pour a = -5.10^{-3}. Diminuer lentement la température de cette façon garantit que l'ensemble de l'espace d'état est exploré et que la solution optimale est trouvée. Pour les expérimentations sur les robots réels, nous nous sommes basés sur l'autonomie qui est d'approximativement 90 minutes. Ce temps a été découpé en deux parties : environ une heure

pour l'apprentissage, puis 30 minutes pour la température minimale T_{min}. Une évaluation de la stratégie dure 23 secondes, pour atteindre T_{min} en une heure, le paramètre a doit valoir -6.10^{-3}.

6.2.3 Résultats expérimentaux I

Résultats de simulation : De nombreuses simulations ont été réalisées afin d'étudier l'influence des paramètres. La première constatation est que la méthode converge toujours vers la stratégie optimale à condition que la température décroisse lentement et que les temps d'évaluation soient suffisamment longs pour garantir une estimation correcte de la performance. La Figure 49 présente l'évolution typique de la meilleure stratégie connue au cours du temps. Dans un premier temps, le robot tourne sur place, garantissant ainsi de ne pas rencontrer d'obstacles. La trajectoire du robot décrit des cercles de plus en plus grands, pour finir par parcourir des trajectoires rectilignes tout en évitant les obstacles. Le robot dispose d'un capteur central C_0; lorsqu'un obstacle est détecté par celui-ci, l'agent dispose de deux stratégies: tourner à droite ou à gauche. Il existe donc deux stratégies optimales dans notre système.

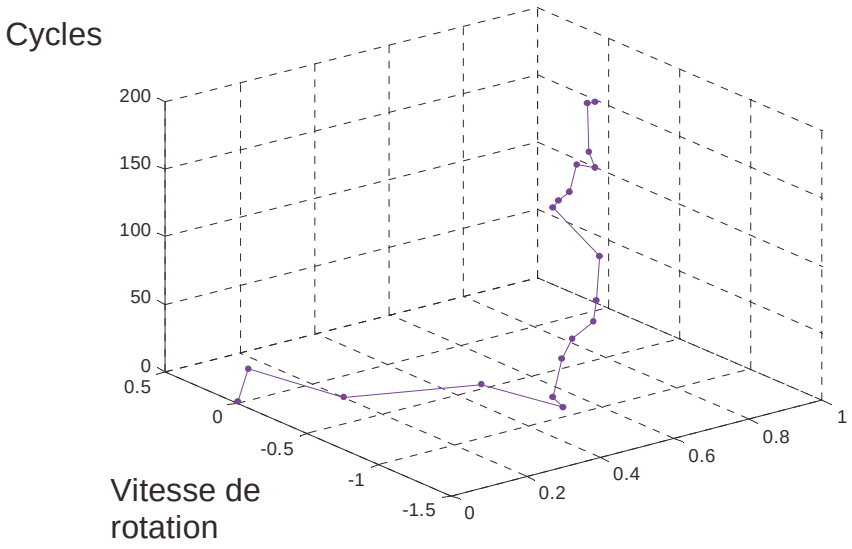


Figure 49 : Evolution de la meilleure stratégie connue

Résultats expérimentaux : les résultats de simulations étant satisfaisants, des expérimentions sur un robot mobile Type 1 ont donc été menées. Les résultats d'une expérience sont présentés sur la Figure 50. En premier lieu, la convergence est atteinte et la récompense est maximisée (Figure 50.b). Toutefois, l'analyse des résultats montre que le système ne converge pas toujours vers la solution optimale. Le même comportement peut donner des performances différentes avec un écart-type important. Si une évaluation donne une récompense importante dès le début de l'apprentissage et que cette évaluation est erronée, alors aucune autre expérience ne pourra écraser cette mauvaise évaluation. Prenons cet exemple critique : l'agent réalise la stratégie suivante " aller tout droit quelque soient les obstacles détectés ". Si sa configuration initiale lui permet d'aller droit sans rencontrer d'obstacles, la récompense attribuée sera élevée. Et pourtant ce comportement est loin d'être optimal. Cette erreur est due à la configuration de l'environnement, la position initiale du robot et le bruit. Pour contrer ce problème, il faudrait augmenter la durée d'une période d'évaluation afin de diminuer l'écart-type autour de la valeur moyenne de la performance, mais les temps d'apprentissage seraient augmentés. De plus, la méthode n'est pas adaptative: une fois que la température a atteint son

minimum, plus aucune variation des poids n'est envisageable, donc le système est incapable de s'adapter à une quelconque panne ou perturbation. L'approche adaptative proposée dans le paragraphe suivant va nous permettre de contrer ces deux problèmes en même temps.

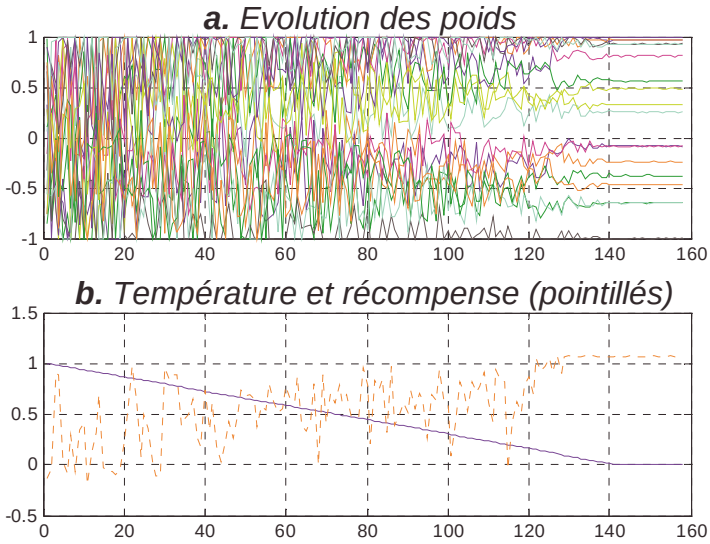


Figure 50 : Résultats d'une expérimentation

6.2.4 Approche adaptative

Nous avons vu dans le chapitre précédent que le recuit simulé classique ne permet pas d'adapter le contrôleur aux perturbations extérieures. Le seul moyen de détecter de tels événements (sans considérer de représentations complexes ni de l'environnement ni de l'agent) est d'exploiter l'information retournée par la récompense. Si un changement notable survient, la récompense va diminuer, sinon c'est que ce changement n'altère pas les performances du système, auquel cas il n'est pas nécessaire de modifier le contrôleur. L'idée maîtresse de cette méthode adaptative est d'autoriser la remontée de la température quand la récompense est faible. Cette méthode qui autorise la remontée de la température est plus fidèle à l'inspiration initiale du recuit simulé, puisque dans le refroidissement d'un matériau en fusion, les remontées de températures sont fréquentes.

Mathématiquement, la température est une fonction de la récompense du meilleur comportement connu. L'inconvénient majeur réside dans le fait que le système sera probablement attrapé dans les maxima locaux. Cette nouvelle hypothèse annule les propriétés de convergence globale du recuit simulé. Notre philosophie est la suivante : si la récompense est correctement attribuée, que la solution trouvée soit un maxima local ou global n'a pas d'importance, tant que la récompense est maximisée. La récompense du meilleur comportement R_{max} connu est diminuée à chaque cycle de la boucle principale. Si la stratégie est constamment performante, R_{max} est constamment mis à jour et le contrôleur reste stable. En revanche, si cette valeur diminue (ce qui signifie que la récompense attribuée n'est pas constante), la température sera augmentée. La méthode est détaillé sur l'Algorithme 7.

Initialisation

$R_{max} \leftarrow 0$
Initialiser chaque poids (w_j) avec une petite valeur.
$T° \leftarrow F_t(0)$

Boucle principale

```
Tant que (vrai)
{
Appliquer la stratégie courante et mesurer la performance R_N(i).
Si (R_max<R_N(i))
{
      R_max=R_N(i)
      Pour chaque poids w_i : W_max(i) ← w_i
      }
      T° ← F_t(R_max)
      Diminuer R_max
Pour chaque poids (w_i), choisir aléatoirement une nouvelle valeur centrée
sur W_max(i) avec une répartition proportionnelle à T°.
}
```

Algorithme 7 : Algorithme adaptatif utilisé pour entraîner le réseau

Les paramètres du réseau sont identiques à ceux présentés précédemment. La nouvelle fonction de la température est toujours linéaire. L'objectif est d'obtenir des températures faibles pour les grandes valeurs de la récompense et inversement. Comme la meilleure récompense espérée est proche de 1, nous avons simplement choisi la fonction représentée sur la Figure 51. $T_{min}=5.10^{-3}$ comme précédemment et a=-1 de façon à atteindre linéairement T_{min} quand R_{max} est proche de 1. Un décrément de R_{max} est égal à 7.10^{-2} , cette valeur garantie le verrouillage du comportement quand la récompense est élevée. Ce paramètre représente la faculté d'adaptation du système. Une grande valeur autorise le système à rapidement basculer vers une autre stratégie, en revanche une stratégie prometteuse peut ne pas être complètement explorée.

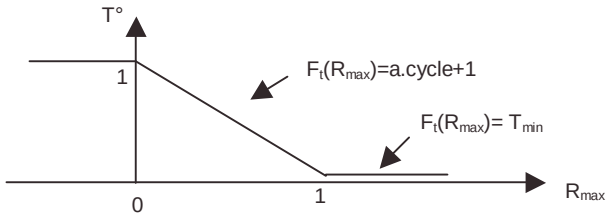


Figure 51 : Evolution de la température en fonction de la récompense du meilleur comportement connu

6.2.5 Résultats expérimentaux II

Résultats de simulation : Les résultats de simulations et expérimentaux sont proches, nous ne présenterons que les seconds, réalisés sur un véritable robot.

Résultats expérimentaux : En premier lieu, la convergence est rapidement atteinte. Le système est vite happé dans un maxima local. La Figure 52.a montre l'évolution des poids du réseau. Après quelques minutes (environ 15 cycles) une stratégie acceptable est découverte. La température décroît soudainement et verrouille le contrôleur autour de cette stratégie. Comme la température n'est pas égale à 0, les valeurs des poids glissent lentement vers la meilleure solution locale. La Figure 53 représente l'influence de chaque capteur sur le comportement global (les flèches fines indiquent les obstacles). Les deux valeurs indiquent la commande appliquée aux moteurs. Il n'y a pas de couche cachée dans notre réseau, donc le

comportement global est une combinaison linéaire de chaque diagramme. Par exemple, la Figure 53.a montre la direction courante du robot quand aucun obstacle n'est détecté : le robot avance en ligne droite. Cette figure montre que la stratégie atteinte n'est pas la solution optimale : sur la Figure 53.f si un obstacle est détecté par le capteur C_6 (Voir la Figure 46 pour la disposition des capteurs), le robot continue de rouler en ligne droite au lieu de tourner à droite comme le voudrait la solution optimale. Malgré cela, le robot est capable d'éviter les obstacles et de maximiser sa récompense[2].

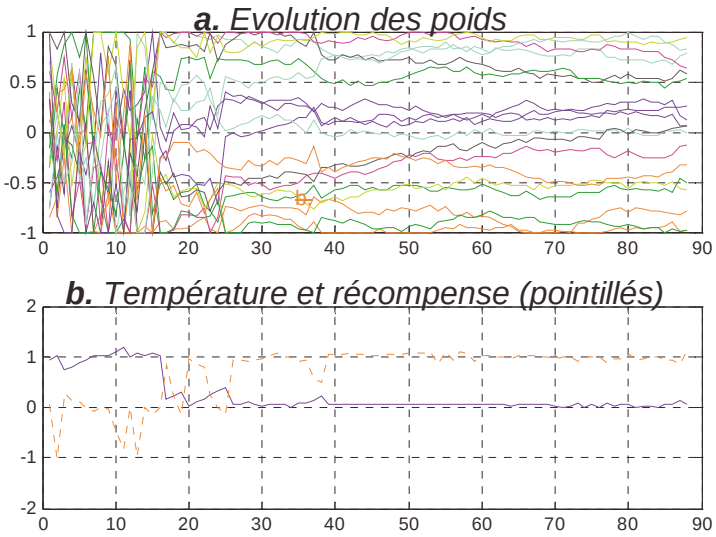


Figure 52 : Résultats d'une expérience

Au 25ème cycle, le contrôleur est verrouillé sur la stratégie locale. 37 cycles après le début de l'expérience, nous avons simulé une panne de capteur en obstruant le récepteur C_1 afin de tester la robustesse de notre système. La Figure 52.b montre l'évolution de la récompense et de la température (ligne pointillée). On discerne sur ce graphique la chute de la récompense. Comme une nouvelle solution est très proche dans l'espace des solutions, l'algorithme renforce l'influence des capteurs les plus proches (C_0 et C_2) pour compenser l'absence de C_1 et le système redevient stable à nouveau. D'autres pannes plus importantes ont été testées et le système retourne dans une phase d'exploration comme dans les premiers cycles de l'expérience. Si la panne est trop importante, l'agent ne reçoit pas de récompense suffisante. De fait, la température ne décroît pas, le système ne converge pas vers une solution, même locale.

[2] L'influence importante du capteur C_7 (Figure 53.e) compense le manque de réactivité du capteur C_6.

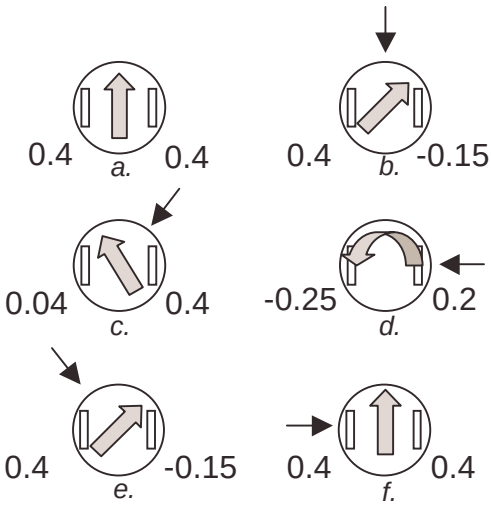


Figure 53 : Influence de chaque capteur sur le comportement global

6.2.6 Conclusion

Nous venons de présenter des résultats d'expériences basées sur le recuit simulé avec pour objectif d'apprendre des comportements réactifs. Nous avons présenté en premier lieu une méthode permettant d'optimiser les paramètres d'un contrôleur neuronal. Dans un environnement sain, la méthode permet à l'agent d'atteindre la solution optimale, mais les temps d'apprentissage sont longs. De plus, la convergence n'est pas assurée quand le système est perturbé par des éléments extérieurs. Le recuit simulé ne peut pas être appliqué directement pour nos applications, ce qui a motivé l'implémentation d'une seconde méthode permettant d'adapter le contrôleur aux perturbations et aux pannes. Cette méthode ne garantit pas de pouvoir atteindre la solution globalement optimale et n'est pas capable de s'adapter aux pannes importantes. Mais elle permet de trouver une solution acceptable rapidement et reste robuste face à certaines pannes en adaptant automatiquement les paramètres. Il ne faut pas nier que le point critique de ces deux méthodes réside dans le réglage des paramètres. Ces paramètres peuvent, selon nous, difficilement être génériques à diverses tâches: ils doivent donc être adaptés en fonction de la nature de la mission à accomplir, de sa complexité, mais aussi de la récompense attribuée. Cette méthode, présentée dans le cadre d'un apprentissage pour des systèmes multi-agents hétérogènes pourrait être appliquée à des systèmes homogènes. Il serait par exemple possible de combiner la méthode évolutionniste présentée au chapitre précédent avec le recuit simulé. Toutes les méthodes présentées précédemment ont été comparées en simulation sur une recherche de maximum (Figure 54). Les algorithmes génétiques (AG) présentent le temps de convergence le plus long et le plus grand écart-type. L'emploi d'une méthode numérique pour les mutations et les croisements permet d'augmenter significativement les performances. Le recuit simulé traditionnel permet une convergence encore plus rapide. Ici l'écart-type est nul, puisque le temps de convergence est fixé par la fonction de décroissance de la température. Enfin la combinaison du recuit simulé et des algorithmes génétiques présente les meilleures performances, toutes méthodes confondues.

Temps de convergence

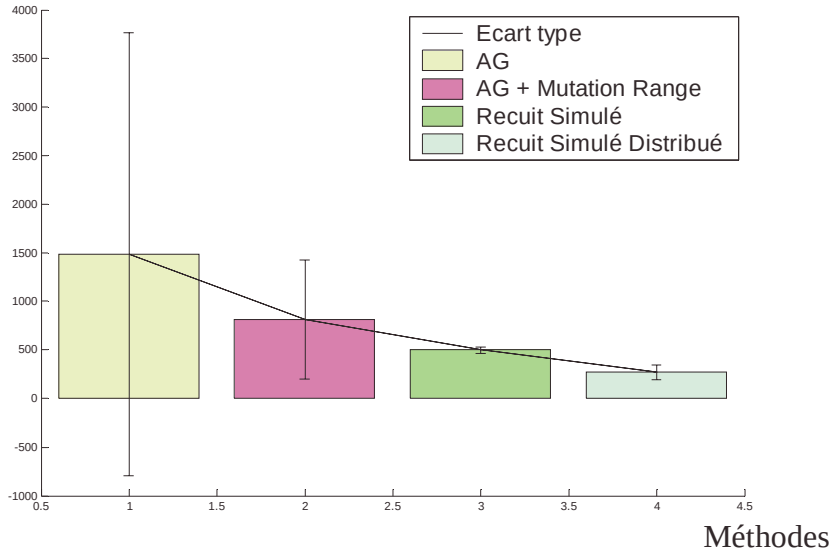


Figure 54 : Comparaison des différentes méthodes

6.3 Extension de l'apprentissage par renforcement au domaine continu

6.3.1 Introduction

Les méthodes d'apprentissage présentées précédemment ne sont pas basées sur la construction d'un modèle. Ces méthodes consistent à tester successivement (ou de façon distribuée) des solutions choisies aléatoirement, pour ne conserver que la ou les meilleures. Lorsque l'agent ne dispose d'aucune information ni modèle, comme c'est le cas pour l'auto-apprentissage, l'apprentissage doit nécessairement commencer par une phase de recherche aléatoire. La différence entre l'apprentissage par renforcement et les techniques présentées jusqu'ici réside dans la construction d'un modèle. La phase initiale aléatoire va servir à construire un modèle markovien de la tâche à apprendre. Une fois cette phase terminée, la phase d'application de la politique consiste à exploiter ce modèle (qui peut continuer sa mise à jour). A complexité équivalente, ce type d'apprentissage est plus rapide que les méthodes basées sur une recherche purement aléatoire. Comme nous l'avons expliqué au chapitre 3, ce type de modèle nécessite de stocker un grand nombre de données. La taille de cette base de données augmente exponentiellement par rapport à la dimension du problème. De plus, les processus markoviens sont discrets et nous préférons appliquer l'apprentissage sur des structures continues.

Il existe pourtant une solution permettant de diminuer la taille des données : utiliser un réseau de neurones[3] pour approximer la matrice Q. Cette méthode a notamment été employée pour programmer le joueur de Backgammon de Tésauro [Tésauro 94]. Le réseau utilisé était un réseau multi couches classique, possédant 198 neurones d'entrées et entre 40 et 80 neurones dans la couche cachée selon les versions. Un autre exemple spectaculaire de cette méthode est l'apprentissage de comportements réactifs pour des robots footballeurs. Il existe

[3] Initialement, la méthode consistait à réaliser une approximation par la méthode des moindres carrés.

dans la coupe du monde de robotique (Robocup) une catégorie simulée (Simulation League). Dans cette catégorie, l'ensemble du terrain et des robots est simulé et les équipes s'affrontent *via* le réseau (Figure 55). La méthode a été employée pour réaliser l'apprentissage de comportements type défense / attaque. A notre connaissance, cette méthode appelée Neuro-Q n'a jamais été appliquée sur un système réel. Et pour cause, dans les deux exemples précédents l'apprentissage est extrêmement long : plusieurs heures, voire plusieurs jours sur des systèmes simulés. Il est évidemment que les temps d'apprentissage seraient disproportionnés sur un système réel.

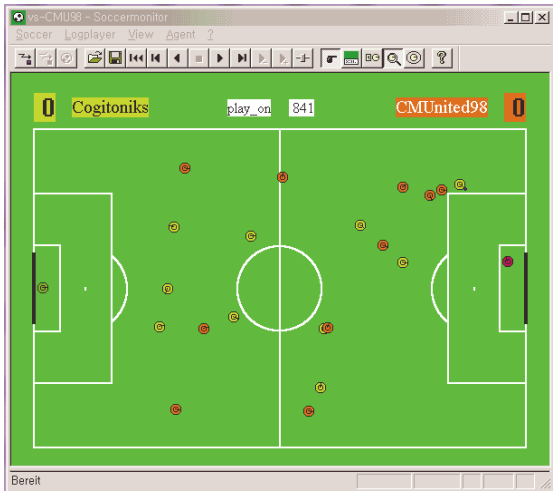


Figure 55 : Capture d'écran du simulateur utilisé pour la Robocup

Nous allons présenter dans la suite de ce chapitre une méthode inspirée du Neuro-Q. Cette méthode consiste à utiliser un réseau de neurones pour modéliser la Q-fonction. Au lieu d'utiliser une méthode de type global comme la rétropropagation du gradient, nous allons utiliser une méthode d'approximation locale. Des travaux antérieurs ont déjà été réalisés dans ce sens. Hormis les travaux cités précédemment, nous noterons principalement deux méthodes : une approximation de la Q-fonction utilisant le CMAC [Miller 90] et une autre méthode basée sur les fonctions radiales [Anderson 93]. En 1997, C.W. Anderson et R. Kretchmar ont comparé les deux méthodes [Kretchmar 97]. Ils arrivent à la conclusion que la méthode utilisant les fonctions radiales proposent une meilleure approximation de la Q-fonction. Nous allons dans la suite de ce chapitre utiliser ces résultats pour proposer une méthode d'apprentissage par renforcement pouvant s'appliquer au domaine d'états et d'espaces continus.

6.3.2 Approximation de la Q-fonction

La méthode d'approximation consiste donc à utiliser un réseau RBF (Radial Basis Functions) qui permet une modélisation locale. Nous allons nous placer dans un premier temps dans le cas d'un système déterministe. L'équation de la Q-fonction selon Watkins est donné par l'Équation 31.

$$\hat{Q}(s,a) \leftarrow r + \gamma . \max(\hat{Q}(s',a'))$$

avec γ : coefficient d'amortissement

Équation 31 : Mise à jour de la Q-fonction dans le cas d'un système déterministe

Nous allons utiliser l'Équation 31 pour actualiser notre réseau de neurones (Figure 56).

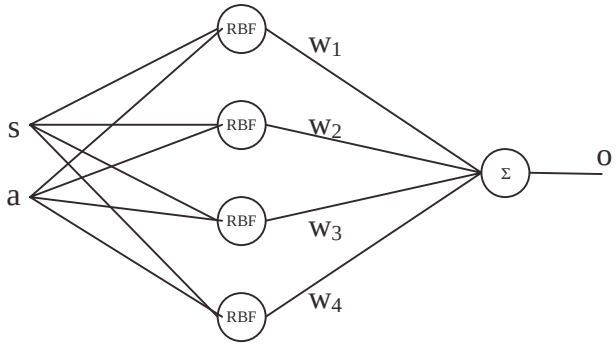


Figure 56 : Architecture du réseau RBF utilisé

$$w_i = w_i + \eta(r + \gamma \max_{a'}(O(s',a')) - O(s,a))Oc_i(s,a)$$

Avec :

- W_i : poids mis à jour
- O(s,a) : sortie du réseau soumis au couple d'entrée s,a
- Oc_i(s,a) : sortie du neurone de poids synaptique i soumis à l'entrée s,a

Équation 32 : Fonction de mise à jour des poids du réseau

L'Équation 32 donne la fonction utilisée pour faire converger le réseau vers la Q-fonction. Cette fonction de mise à jour peut aussi être utilisée dans le cas des systèmes non déterministes. En effet, dans le cas d'un système non déterministe, l'équation de mise à jour de la Q-fonction consiste à calculer la moyenne des récompenses acquises au fur et à mesure des expériences. Dans le cas présent, l'apprentissage par réseau de neurones moyenne ces récompenses espérées automatiquement. De plus, contrairement au Q-learning, la méthode permet de réaliser une moyenne pondérée des récompenses : les coefficients des dernières expériences seront plus importants. La méthode permet un oubli des expériences anciennes au profit des plus récentes. Ce qui signifie concrètement que la méthode sera adaptative en cas de panne ou de changement dans l'environnement. Le facteur d'oubli sera dépendant de la valeur du coefficient d'apprentissage η. Il est également possible d'obtenir le même phénomène sur la version traditionnelle du Q-Learning en bornant la matrice Visit(s,a).

Un inconvénient de cette méthode est de déterminer le nombre et la position des centres. Il existe des méthodes adaptatives [Mulgrew 96][Burdsall 97] basées sur le principe suivant : à chaque mise à jour du réseau, les centres sont rapprochés de ce point proportionnellement à l'erreur de modélisation. La méthode consiste à concentrer les centres aux environs des zones difficilement modélisables afin de minimiser l'erreur.

L'Équation 32 demande de connaître la valeur de Q estimée maximale pour une entrée donnée. Q étant estimé par le réseau, l'objectif est de trouver la valeur maximale du réseau pour un état donné.

6.3.3 Recherche du maximum

Dans les réseaux RBF, la fonction gaussienne la plus souvent utilisée est donnée par l'Équation 33. Ce type de fonction est dérivable sur R. La première approche utilisée consiste à calculer la fonction de transfert globale (Équation 34) du réseau (o=f(s,a)), la dériver par rapport à la variable d'entrée a (Équation 35) et calculer les zéros de cette dérivée.

$$f(x)=e^{-(x-c_i)^2}$$

Équation 33 : Fonction de transfert d'un neurone de type RBF

$$o=f(s,a)=\sum_{i=0}^{n} w_i\left(e^{-(s-c_{si})^2}.e^{-(a-c_{ai})^2}\right)$$

avec :

- W_i : poids synaptique,
- C_{si} : position du centre dans l'espace d'entrée des états,
- C_{ai} : position du centre dans l'espace d'entrée des actions.

Équation 34 : Fonction de transfert du réseau pour un couple d'entrée à 1 dimension

$$\frac{do}{da}=\sum_{i=0}^{n} -w_i.e^{-(s-c_{si})^2}.(a-c_{ai})e^{-(a-c_{ai})^2}$$

Équation 35 : Dérivée de la fonction de transfert du réseau

Après calcul, il s'avère que ce type d'équation sous forme de somme d'exponentielles ne présente pas de solution analytique. Pour nous affranchir de ce problème, nous avons considéré une autre fonction de transfert pour les neurones de la couche cachée. Plutôt que d'utiliser une fonction exponentielle pour obtenir la gaussienne, nous avons utilisé un polynôme du 4ème ordre borné sur l'intervalle [-1, 1]. La courbe montrée sur la Figure 57 représente le tracé de l'équation $f(x)=x^4-2.x^2+1$.

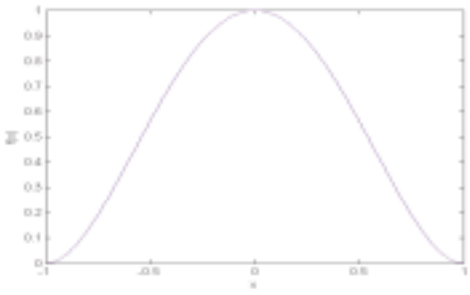


Figure 57 : Gaussienne obtenue en utilisant un polynôme d'ordre 4

Le problème de cette fonction réside dans son espace de définition. En effet, cette fonction n'est définie que sur l'intervalle [-1, 1]. Ici, la fonction de transfert globale sera dérivable par intervalle et présentera une solution analytique sur chaque intervalle. Pour

trouver le maximum, il faudra calculer la solution analytique sur chaque intervalle, puis prendre le maximum des maxima. Nous avons estimé que cette solution était trop lourde à implémenter, d'autant plus que l'algorithme est différent selon les positions des centres (selon le nombre de gaussiennes qui peuvent se chevaucher).

Bien qu'une solution analytique aurait présenté de nombreux d'avantages, comme la simplicité de l'implémentation ou la rapidité des temps de calcul, nous avons dû renoncer à cette approche. Nous nous sommes orientés vers une solution de recherche du maximum numérique. La fonction et sa dérivé sont parfaitement connues et la fonction peut présenter des maxima locaux. Nous avons choisi de prendre l'hypothèse suivante afin d'accélérer les temps d'apprentissage : nous estimons que le maximum se trouve à proximité de la gaussienne associée au plus grand poids synaptique. Bien que cette hypothèse ne puisse pas être mathématiquement vérifiée, aucun contre-exemple n'a été rencontré pendant les expérimentations. Dans les cas critiques ou l'hypothèse n'est pas vérifiée, la méthode prendra un des maxima locaux (probablement le second, la statistique est inversement proportionnelle au rang du maximum) donc l'erreur n'aura pas une grande influence sur l'approximation de la Q-fonction.

6.3.4 Résultats expérimentaux 1

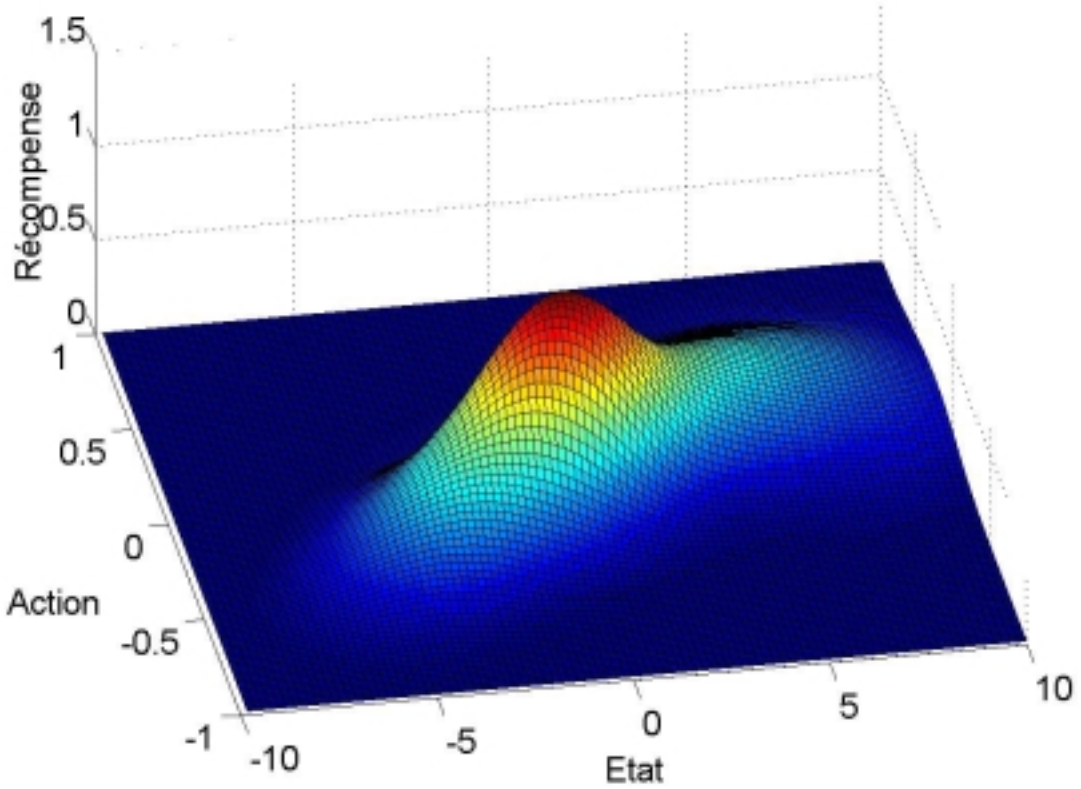


Figure 58 : Modélisation de la Q-valeur

La suite de ces expérimentations a été réalisée par Bastien Jacquot, stagiaire de DEA. Les expérimentations ont été réalisées sur BOP, le système de vision stéréoscopique présenté précédemment. La première expérience, utilisée pour valider la méthode d'apprentissage a été

réalisée sur un problème où les dimensions de l'espace d'état et d'action sont de un. L'objectif de l'apprentissage est de centrer un point lumineux dans l'image d'une des deux caméras. La récompense est inversement proportionnelle à la distance entre la tâche lumineuse et le centre de l'image. Le problème revient à traquer la tâche lumineuse. Nous avons choisi ce type de marqueurs (lumineux) afin de pouvoir s'affranchir des problèmes de traitement d'images.

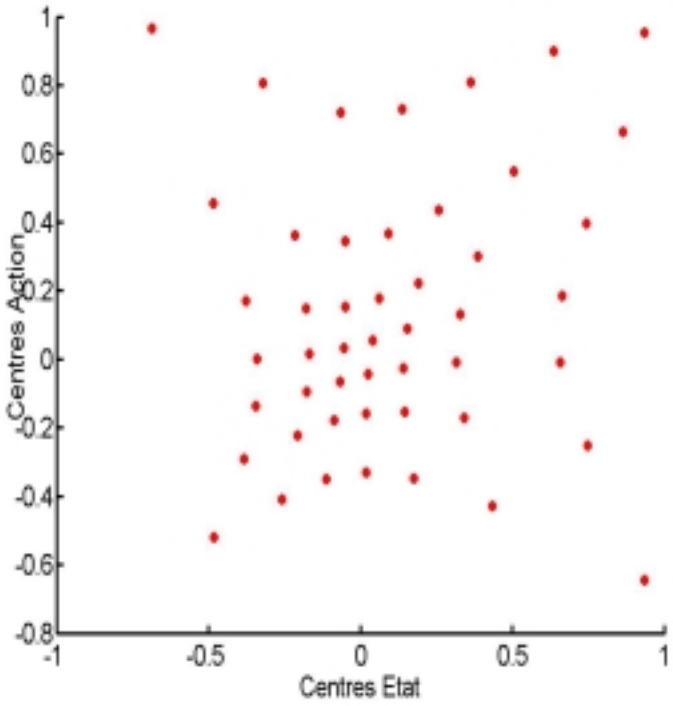


Figure 59 : Position des centres

La Figure 58 montre le résultat de la modélisation de la Q-fonction. L'état est donné par la position du point traqué dans l'image et l'action correspond à un incrément de consigne sur le servomoteur. Sur cette figure quelques points représentatifs peuvent être remarqués comme le centre de coordonnées : état=0 et action=0, qui possède la valeur maximale de la Q-fonction. Cela vérifie bien que lorsque la tâche est centrée dans l'image (Etat=0) et que le servomoteur est immobile (Action=0), la récompense est maximale. L'allure générale de la courbe vérifie bien les résultats escomptés : lorsque la tâche se trouve à gauche dans l'image (Etat < 0) il vaut mieux reculer le servomoteur (Action < 0) et inversement. Les meilleures récompenses se trouvent donc sur la diagonale. La méthode utilisée ici pour la modélisation est un réseau RBF avec une adaptation automatique de la position des centres. Cette dernière est montrée sur la Figure 59. Les centres se trouvent plus concentrés au centre, là où la modélisation est la plus importante.

6.3.4 Résultats expérimentaux 2

L'exemple précédent a permis de montrer que la modélisation de la Q-fonction est possible et que les résultats permettent d'appliquer une politique maximisant la récompense. Nous allons maintenant nous intéresser à l'apprentissage complet du problème. L'objectif est de centrer la tâche dans les deux dimensions sur une caméra. Les espaces d'états et d'actions comportent deux dimensions chacun.

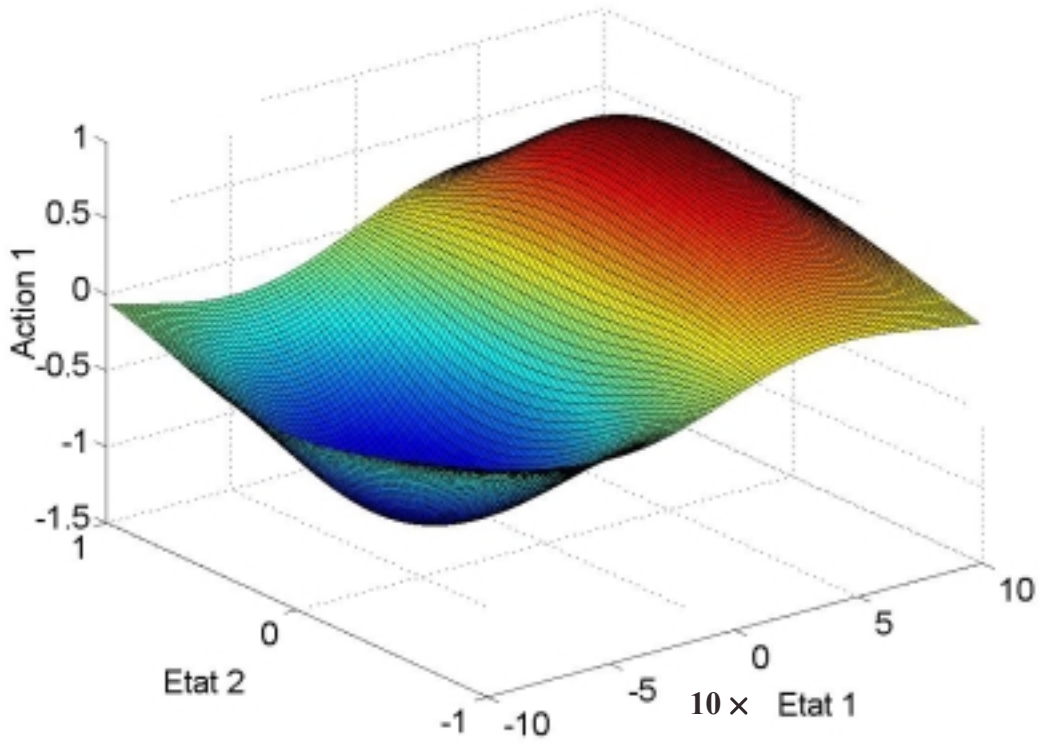


Figure 60 : Modélisation de l'Action 1 optimale en fonction de l'état de l'agent

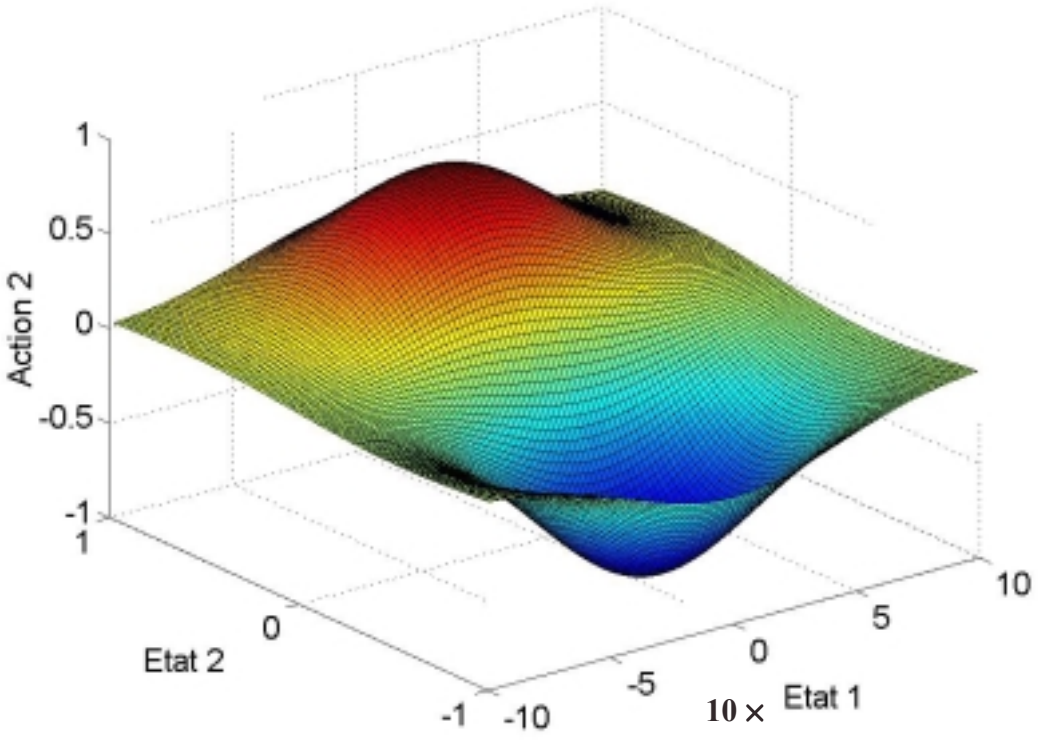


Figure 61 : Modélisation de l'Action 2 optimale en fonction de l'état de l'agent

La conclusion de B. Jacquot est la suivante : " Si les résultats obtenus sont corrects (Figure 60, Figure 61), il apparaît que la fonction Q-valeur est mal modélisée par le réseau lorsque le temps d'apprentissage est restreint à des valeurs raisonnables. Cependant et malgré cette mauvaise modélisation, le modèle obtenu pour les actions est très bon et notamment, le découplage entre les servomoteurs se fait très rapidement, même s'il n'est pas parfait. En fait et c'est à la fois un avantage et une difficulté d'interprétation, la fonction de Q-valeur n'est jamais entièrement explorée. De ce fait, la forme globale peut sembler non cohérente vue de l'extérieur, mais la fonction de Q-valeur peut localement être suffisamment bien modélisée pour permettre à l'algorithme d'apprentissage de trouver un chemin optimal. "

6.4 Conclusion

Ce chapitre traitait de l'apprentissage de comportements réactifs pour les systèmes hétérogènes. Nous avons pris comme hypothèse de départ qu'un apprentissage collectif n'était pas envisageable. L'apprentissage a donc été réalisé sur un agent isolé du reste du système.

La première méthode d'apprentissage était basée sur le recuit simulé. Nous avons dans un premier temps expérimenté la version traditionnelle du recuit simulé. L'objectif était de faire apprendre une tâche d'évitement d'obstacles à un robot mobile. La première méthode permet d'obtenir la convergence vers la solution optimale dans un environnement sain. Les résultats expérimentaux ont montré que la convergence n'était pas assurée dans des environnements perturbés. Ces résultats justifient d'ailleurs les expérimentations sur plate-forme expérimentale, qui permettent de mettre en évidence certains phénomènes non perceptibles en simulation.

Nous avons ensuite proposé une méthode d'apprentissage adaptative inspirée du recuit simulé. Cette méthode ne garantit pas la convergence vers la solution optimale, en revanche, elle permet d'accélérer les temps d'apprentissage et d'adapter le contrôleur en cas de panne ou de changement extérieur. Il est important de préciser qu'à l'issue de ces travaux, les méthodes inspirées du recuit simulé demande un réglage non trivial des paramètres. Ce réglage est crucial pour les temps d'apprentissage, les propriétés de convergence et les facultés d'adaptation du système.

Enfin, nous avons proposé une extension de l'apprentissage par renforcement aux systèmes continus. La méthode consiste à modéliser la Q-fonction en utilisant un réseau de neurones de type RBF. Ce type de réseau permet une modélisation locale de la Q-fonction. Il est donc possible d'accélérer les temps d'apprentissage en concentrant l'apprentissage sur des zones de l'espace d'états présentant des récompenses importantes. Les expérimentations menées sur notre système stéréoscopique ont montré une approximation suffisante de la Q-fonction, permettant de déterminer les meilleures actions à exécuter depuis un état donné.

De manière générale, les temps d'apprentissage évoluent exponentiellement avec la complexité de la tâche. L'apprentissage de tâches réactives pour les systèmes hétérogènes demande des temps d'apprentissage importants. Il nous apparaît essentiel de ne pas tenter d'apprendre directement une tâche complexe. Pour obtenir la convergence dans des temps raisonnables, la meilleure solution consiste à diviser la tâche en plusieurs tâches de bas niveaux. Ces résultats confirment donc les hypothèses communément acquises dans les systèmes multi-agents distribués.

CHAPITRE

7

7. Conclusion générale

Les travaux présentés dans ce mémoire avaient pour but l'étude de techniques d'apprentissage appliquées aux systèmes multi-agents. Arrivé au terme de ces travaux, nous nous proposons de faire un rapide bilan de chaque chapitre avant d'en évoquer les conclusions et les perspectives.

Le premier chapitre introductif, présente le contexte de ces travaux en insistant sur les futures tâches que ces recherches pourraient permettre d'accomplir. Ces travaux s'orientent vers des tâches d'exploration, de recherche, d'analyse et de sauvetage en milieu inconnu ou dangereux (milieu spatial, sous-marin ou encore nucléaire). Ces recherches s'inscrivent dans deux cadres principaux : les systèmes multi-agents et l'apprentissage. Les techniques d'apprentissage actuelles ne permettent pas de résoudre des problèmes complexes. C'est pourquoi ces travaux se focalisent sur l'apprentissage de comportements réactifs.

Le deuxième chapitre présente les systèmes multi-agents. Après une introduction et quelques définitions sur les systèmes multi-agents, les principales tâches génériques étudiées dans les travaux antérieurs sont présentées. Ces travaux antérieurs montrent clairement

l'intérêt de l'approche multi-agents, ils s'inscrivent notamment comme un complément aux techniques de programmation dite traditionnelle. En s'appuyant sur les hypothèses émises par Rodney Brooks, il serait donc possible de construire de nouvelles architectures logicielles pour les systèmes multi-robots, basées sur des bibliothèques de comportements réactifs. Enfin, les différents protocoles de communication employés dans les systèmes multi-agents sont présentés.

Dans le troisième chapitre, les principales techniques d'apprentissage sont décrites. Le parallèle avec le vivant est omniprésent et ces méthodes d'apprentissage sont généralement inspirées du vivant à commencer par les réseaux de neurones. Seules les techniques les plus courantes ont été décrites : les réseaux linéaires, la rétropropagation du gradient, et les réseaux à fonctions radiales. Ces techniques nécessitent de fournir aux réseaux des bibliothèques d'exemples, ce qui est mal indiqué pour nos problèmes d'auto-apprentissage. Une seconde technique basée sur les processus markoviens est présentée : l'apprentissage par renforcement. Là aussi, seules les méthodes les plus connues sont présentées : la programmation dynamique et le Q-learning. Ces méthodes présentent l'avantage d'être parfaitement appliquées aux problèmes d'auto-apprentissage, mais nécessitent de stocker des bases de données importantes et surtout sont mal indiquées pour les applications aux problèmes continus. Puis, est présentée une autre méthode d'apprentissage inspirée elle-aussi du vivant : les algorithmes évolutionnistes. Ces méthodes, directement inspirées de l'évolution darwinienne, semblent parfaitement indiquées à nos problématiques, à un détail près : ce sont des méthodes supervisées et ces travaux s'inscrivent dans le tout-distribué. Avant de clôturer ce chapitre, un dernier paragraphe est consacré à l'estimation de la performance, car la façon dont les agents sont récompensés peut influencer l'apprentissage et ce paragraphe synthétise les différentes façons d'estimer les performances d'un agent.

Afin de pouvoir expérimenter les méthodes et techniques proposées dans des conditions réelles, une plate-forme expérimentale a été construite. Cette plate-forme est décrite en détails dans le quatrième chapitre. Elle est principalement constituée de cinq robots mobiles Type 1. Ces robots ont été construits dans le but d'évoluer ensemble dans le même environnement. Ils peuvent se déplacer, détecter les obstacles, communiquer entre eux et sont totalement autonomes en terme d'énergie et de puissance de calcul. L'un de ces robots a été associé à un manipulateur miniature pour former M^3 (Manipulateur Mobile Miniature). Ce manipulateur à la conception originale présente une dynamique particulièrement intéressante. En effet, aucun moteur ne doit supporter le poids des autres actionneurs, puisque tous les moteurs sont fixés sur le châssis et que les mouvements sont transmis par des jeux de poulies et de courroies. Cette plate-forme expérimentale est également complétée d'un système de stéréovision BOP. Ce système, équipé de 2 caméras mobiles, peut acquérir plusieurs images d'un même objet en multipliant les points de vue et ainsi estimer sa position dans l'espace avec une grande précision.

Le cinquième chapitre est consacré à l'apprentissage de comportements réactifs pour les systèmes multi-robots homogènes. Ce chapitre est uniquement basé sur une approche évolutionniste, la première partie détaille une expérience menée à l'EPFL sur l'apprentissage d'un comportement d'évitement d'obstacles grâce aux algorithmes génétiques. Cette expérience montre notamment que de telles méthodes d'apprentissage permettent à une population de robots d'améliorer ses propres performances aux fil des générations. Cette expérience a été réalisée de manière supervisée. Afin de prouver qu'une telle méthode pouvait être appliquée sur un système entièrement distribué, nous avons présenté une version distribuée des algorithmes évolutionnistes. Les opérateurs de croisements et de mutations et

les communications ont été redéfinies afin de permettre aux agents de réaliser les croisements par couple. La méthode a également été testée sur une tâche d'évitement d'obstacles en simulation et sur les robots. L'étude des résultats a permis d'étudier l'influence des paramètres et on distingue lors de l'apprentissage des paliers correspondant chacun à une étape de l'évolution.

Le sixième chapitre est consacré à l'apprentissage de comportements réactifs appliqué aux systèmes hétérogènes. L'hypothèse de départ est ici que l'apprentissage collectif n'est pas envisageable et les méthodes sont appliquées sur un agent isolé de la population. La première méthode utilisée est basée sur l'application du recuit simulé, deux versions sont présentées : une première version traditionnelle et une seconde version améliorée permettant principalement d'accélérer les temps d'apprentissage et de s'adapter aux pannes et modifications de l'environnement. Les méthodes basées sur le recuit simulé, permettent d'obtenir une convergence rapide, mais ne garantissent pas la convergence vers la solution optimale. Les risques d'être happé dans un maxima local sont grands. La deuxième partie de ce chapitre est consacrée à l'apprentissage par renforcement. Afin de diminuer les durées d'apprentissage et de minimiser la taille des données mémorisées, la matrice Q est approximée par un réseau de neurones de type RBF. Les expérimentations menées sur le système de stéréovision BOP ont montré l'exactitude de l'approximation réalisée par le réseau ce qui permet au Q-learning de déterminer la meilleure action à réaliser dans chaque configuration.

Les méthodes étant assez éloignées les unes des autres, il était assez difficile de les comparer de façon quantitative. C'est pourquoi les résultats sont synthétisés selon cinq critères:

- La rapidité de convergence, relative aux durées d'apprentissage.
- L'application au domaine continu, la méthode est-elle plutôt indiquée pour les problèmes discrets ou continus ?
- Les capacité d'adaptation, le contrôleur appris est-il dédié à des circonstances précises ou peut-il évoluer en cas de changement ?
- L'optimalité de la solution, le résultat est-il un maxima absolu ou local ?
- La taille des données mémorisées.

La Figure 62 montre les performances des méthodes évolutionnistes. On distingue que les algorithmes génétiques proposent des performances moyennes sur l'ensemble des critères avec toutefois un point faible : la lenteur de convergence et un point fort: la faible taille des données mémorisées. La version distribuée proposée dans ce manuscrit ne modifie pas ces performances, elle évite de centraliser l'ensemble des données.

La Figure 63 montre les performances du recuit simulé dans deux versions : le recuit simulé traditionnel (en continu) et la version proposée (en pointillés). La version traditionnelle dispose de performance correctes sur l'ensemble des critères, à l'exception des capacités d'adaptation. En effet, lorsque la température est faible, le système ne peut plus évoluer. Il produit donc un contrôleur dédié. A l'inverse, la version proposée dispose d'une grande capacité d'adaptation, mais cette amélioration se fait au détriment de l'optimalité et le système se fait happer dans les minimums locaux.

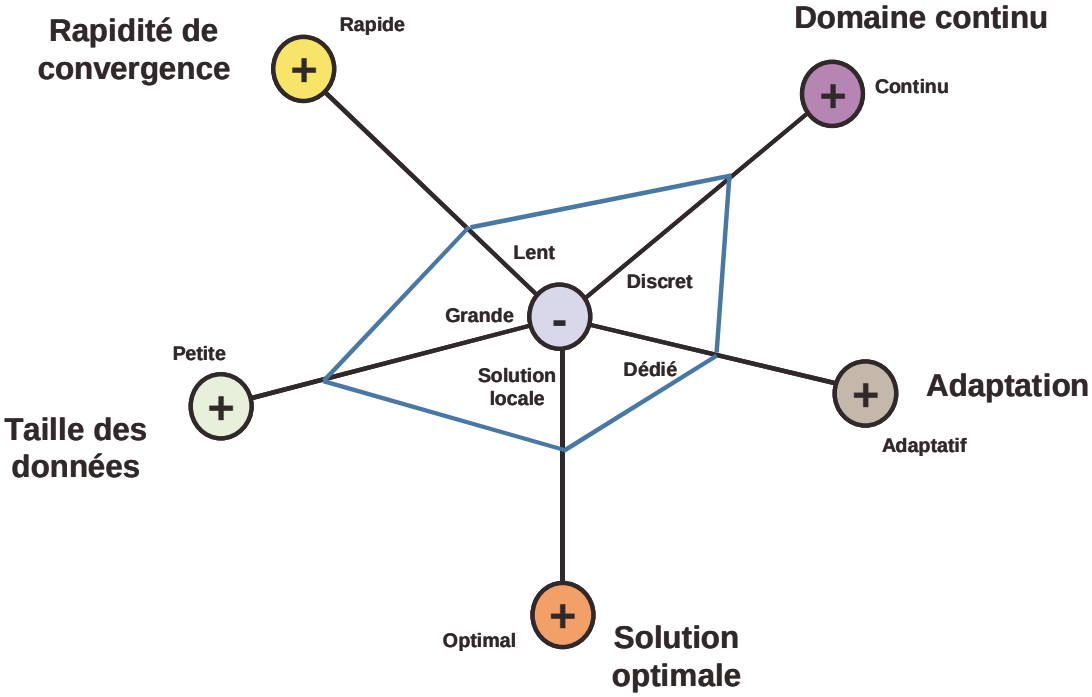


Figure 62 : Performance des méthodes évolutionnistes

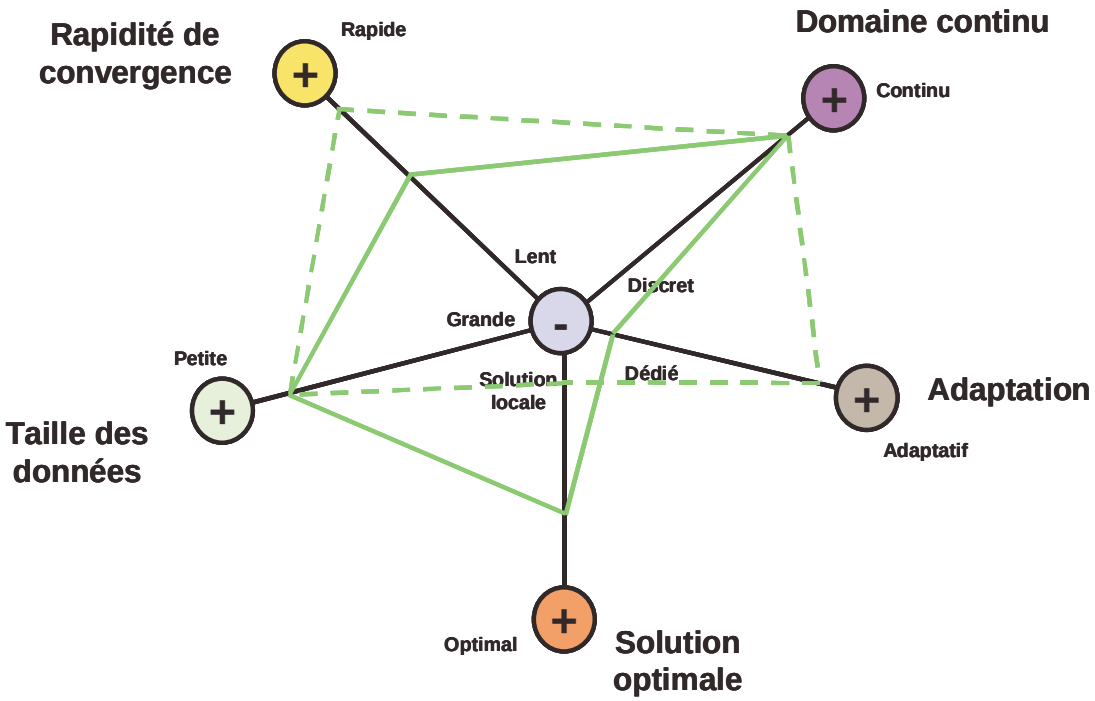


Figure 63 : Performance du recuit simulé

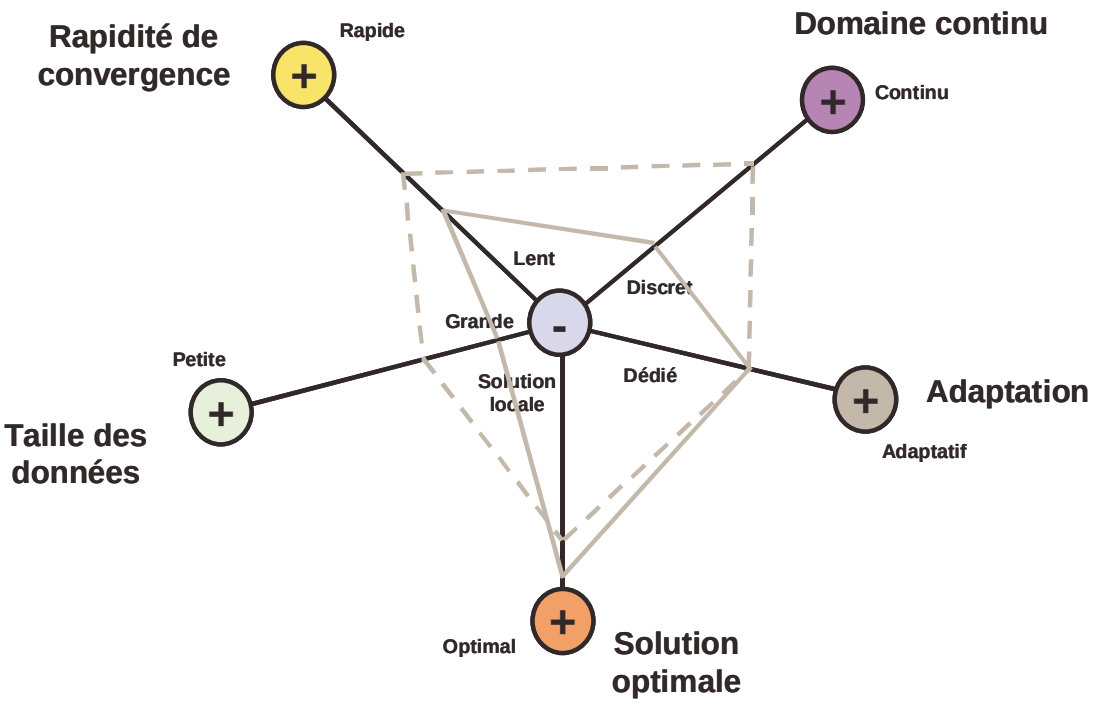


Figure 64 : Performance de l'apprentissage par renforcement

La Figure 64 compare les performances du Q-Learning à l'extension proposée dans ce manuscrit. La courbe continue montre les performances d'un Q-Learning classique, et la courbe en pointillés les performances modifiées grâce à l'apport d'un réseau de neurones modélisant la matrice Q. On constate que le Q-Learning dispose de caractéristiques relativement mauvaises sur l'ensemble des critères à l'exception l'optimalité. En effet, la convergence a été prouvée pour les systèmes déterministes. L'ajout d'un réseau de neurones augmente globalement les performances : la méthode s'applique dorénavant au domaine continu. La convergence est plus rapide car le réseau interpole les points non visités. La taille des données est diminuée car il ne reste que les poids synaptiques à mémoriser. Mais la modélisation n'est pas parfaite, et la méthode perd donc ses propriétés d'optimalité.

Les recherches à venir s'orienteront vers l'étude de l'apprentissage d'autres comportements réactifs, comme par exemple, du suivi et ralliement de cibles, de la saisie d'objet ou des tâches de navigation plus complexes. Ces expériences pourront être menées sur la plate-forme expérimentale et permettront de disposer de bibliothèques de comportements réactifs associés aux robots.

Les futures recherches pourront s'orienter vers l'étude de la combinaison de plusieurs méthodes d'apprentissage: par exemple, la combinaison du recuit simulé et des méthodes évolutionnistes pour des populations homogènes. Le recuit simulé permet une convergence rapide vers les maxima locaux ou globaux, alors que les méthodes évolutionnistes permettent une convergence vers l'optimum absolu. L'idée est donc de disposer d'une population d'individus capables d'atteindre rapidement des solutions locales et la sélection naturelle permettra de ne conserver que les meilleurs. De même, un agent isolé du groupe pourra

évoluer seul, mais les échanges avec le reste de la population permettra d'accélérer l'apprentissage.

Nous nous sommes concentrés dans ces travaux sur l'apprentissage de comportements réactifs. Les futures recherches devront s'orienter vers l'apprentissage de comportements et de stratégies plus complexes. Non pas en complexifiant les tâches à apprendre, mais en synthétisant les comportements réactifs au sein d'une architecture d'apprentissage de plus haut niveau permettant un séquencement ou un mixage de ces comportements. Des architectures du type Satisfaction/Altruisme ou ALLIANCE semblent être bien appropriées aux contraintes de robustesse et d'adaptation nécessaires à de telles tâches.

Bibliographie

[Alami 98] R. Alami, S. Fleury, M. Herrb, F. Ingrand and F. Robert "Multi-robot cooperation in the Martha project", IEEE Robotics and Automation Magazine, 5(1), p.36-47, 1998.

[Anderson 93] C. Anderson, "Q-Learning with Hidden-Unit Restarting" Advances in Neural Information Processing Systems, volume 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan Kaufmann Publishers, San Mateo, CA, p. 81–88, 1993.

[Arkin 92] R.C. Arkin, " Cooperation without Communication : Multiagent Schema-Based Robot Navigation ", Journal of Robotic Systems, Vol. 9 (3), p. 351-364, avril 1992.

[Balch 94] T. Balch and R.C. Arkin, " Communication in Reactive Multiagent Robotic Systems ", Autonomous Robots, 1, p 27-52, 1994.

[Brainteberg 84] V. Braitenberg, "Vehicule experiments in synthetic psychology", MIT Press, Cambridge, MA, 1984.

[Brooks 86] R. A. Brooks, "A robust layered control system for a mobile robot", IEEE Journal of Robotics and Automation, p. 14-23, 1986.

[Burdsall 97] B. Burdsall, C. Giraud-Carrier, "GA-RBF: A Self-optimising RBF Network", Third International Conference on Artificial Neural Networks and Genetic Algorithms, Norwich, UK, p. 346-349, 1997

[Cullogh 43] W.S. Mc Cullogh and W. Pitts , " A logical calculus of the ideas immanent in nervous activity ", Bulletin of Mathematical Biophysics 5, p. 115-133, 1943.

[Drogoul 92] A. Drogoul and J. Ferber, " From Tom Thumb to the Dockers : Some Experiments with Foraging Robots ", 2nd International Conference on Simulation of Adaptative Behaviour, Honolulu, p 451-459, 1992.

[Ferber 95] J. Ferber, "Les systèmes multi-agents, vers une intelligence collective", Livre InterEdition, 1995.

[Floreano 94] D. Floreano et F. Mondada, " Autonomic creation of an autonomous agent : Genetic evolution of a neural-network driven robot ", Simulation of Adaptive Behavior, from animals to animats 3, 1994.

[Golberg 89] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, 1989.

[Goldberg 99] D. Goldberg et M.J. Mataric, "Coordinating mobile robot group behavior using a model of interaction dynamics", Proceedings, Autonomous agents Seattle, WA, p. 100-107, 1999.

[Goldberg 00] D. Goldberg and M.J. Mataric, "Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks," USC Institute for Robotics and Intelligent Systems Technical Report IRIS-00-387, 2000.

[Hebb 49] D.O. Hebb, " The Organisation of Behaviour", Wiley, New York, 1949

[Hopfield 82] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Sciences, p. 460-464, 1982.

[Kane 86] T.R. Kane et D.A. Levinson, "The use of Kane's dynamical equations in robotics", The International Journal of Robotics Research 2, No. 3, 1986, pp. 3-21, 1986.

[Kretchmar 97] R. Kretchmar et C. Anderson, "Comparison of CMAC's and radial basis functions for local function approximators in reinforcement learning", In proceedings od the IEEE international Conference on Neural Networks, Houston, TX, p. 834-837, 1997.

[Lucidarme 02] P. Lucidarme, O. Simonin et A. Liégeois, "Implementation and Evaluation of a Satisfaction/Altruisme-Based Architecture for Multi-Robot Systems", proc. ICRA'02, Washington D.C., p. 1007-1012, 2002.

[Luh 80] J.Y.S. Luh, M.W. Walker et R.C.P. Paul, "On line computational scheme for mechanical manipulators" Trans. Of the ASME Journal of Dynamic Systems, Measurement and Control, Vol. 102, p69-76, 1980.

[Madani 02] T. Madani, A. Benallegue and N.K. M'SIRDI, " Apprentissage par renforcement pour la Navigation d'un Robot Mobile dans des Environnements Inconnus ", Journée des Jeunes Chercheurs en Robotique, 2002.

[Miller 90] Miller, W. T., F. H. Glanz et L. G. Kraft. "CMAC:An associative neural network alternative to backpropagation" Proceedings of IEEE, 78, p. 1561–1567, 1990.

[Minsky 69] M. Minsky and S. Papert, " Perceptron ", The MIT Press, Cambridge, 1969.

[Mitchell 97] T. M. Mitchell, "Machine learning", Livre, Mc Graw-Hill International Editions, 1997.

[Mitiguy 96] P.C. Mitiguy and T.R. Kane, "Motion Variables Leading to Efficient Equations of Motion", The International Journal of Robotics Research, Vol. 15, No. 15, pp. 522-532, October 1996.

[Mulgrew 96] B. Mulgrew, "Appling Radial Basis Functions," IEEE Signal Processing Magazine, vol. 13, p. 50–65, Mars 1996.

[Mondada 89] D. Mondada, E. fanzi et P. Ienne, " Mobile robot miniaturization: A tool for investigation in control algorithms " In proceedings of the third International Symposium on experimental Robotics, Kyoto, Japan, 1993.

[Parker 94] L. Parker, "An architecture for fault tolerant, cooperative control of heterogeneous mobile robots", Proc. Of the 1994 International Conference on Intelligent Robots and Systems, Munich, Germany, p. 776-783, 1994.

[Parker 99] L. Parker, "Adaptative heterogeneous multi-robot teams", Neurocomputing, special issue of NEURAP'98: Neural networks and their applications, 28, p.75-92, 1999.

[Rongier 01] P. Rongier et P. Lucidarme, "A Sizing Method for a Multi-Robot System", International Conference On Intelligent Robotic Systems, Maui, Hawaii, p 387-392, 2001.

[Rosenblatt 58] F. Rosenblatt, "The Perceptron : a Probabilistic Model for Information Storage and Organisation in the Brain", Psychological Review, p. 386-408, 1958.

[Rumelhart 86] D.E. Rumelhart and J.L. Mc Clelland, "Parallel Distributed Processing", The MIT Press, vol. 1 et 2, Cambridge, 1986.

[Simonin 00a] O. Simonin, A. Liégeois and P. Rongier, " An Architecture for Reactive Cooperation of Mobile Distributed Robots ", 5th International Symposium on Distributed Autonomous Robotic Systems 4 ,Knoxville, TN Springer, USA, p. 35-44, 2000.

[Simonin 00b] O. Simonin et J. Ferber " Modeling self satisfaction and altruism to handle action selection and reactive cooperation ", 6th international Conference on the Simulation of Adaptive Behavior, volume2, p. 314-323, 2000.

[Simonin 01] O. Simonin, "Le modèle satisfaction-altruisme : coopération et résolution de conflits entre agents situés réactifs, application à la robotique" , Thèse de Doctorat en Informatique , LIRMM UM2, soutenue le 20 décembre 2001.

[Steels 89] L. Steels, " Cooperation between distributed agents Through Self-organisation ", Journal on robotics and autonomous systems, North Holland, Amsterdam. 1989.

[Steels 91] L. Steels, " Towards a Theory of Emergent Functionnality ", From animal to animats, MIT Press, p. 451, 1991.

[Szepesvari 97] C. Szepsvari, "The assymptotic convergence-Rate of Q-learning", in Proc. Neural Information Processing Systems, 1997.

[Astrom 95] K J. Astrom et B. Wittenmark, "*Adaptive Control* ", Addison-Wesley, Deuxième Édition,1995.

[Tesauro 94] G.J. Tesauro, " TD-Gammon, a self-teaching backgammon program, achieves master-level play", Neural computation, 6(2), p.215-219.

[Watkins 89] C.J.C.H Watkins, "Learning from delayed rewards", thèse de doctorat, Cambridge University, 1989.

[Winfield 00] A. Winfield et O. Holland "The application of wireless local area network technology to the control of mobile robots", Microprocessors and Microsystems, 23/10, p 597-607, 2000.

[Zapata 92] R. Zapata, P. Lepinay, C. Novalés and P. Déplanques, "Reactive behaviors of fast mobile robots in unstructured environments : sensored based control and neural networks." SAB, December 1992.

Annexe A : Dynamique du manipulateur M3

A.1. Vitesses angulaire

Le vecteur vitesse angulaire $\boldsymbol{\omega}_i$ d'un corps i peut être déterminé simplement grâce à la structure plan du bras et des poulies :

$$(\boldsymbol{\omega}_1)_{\text{repère fixe}} = \begin{bmatrix} 0 \\ 0 \\ \dot{q}_1 \end{bmatrix} \quad (\boldsymbol{\omega}_i)_{\text{repère fixe}} = \begin{bmatrix} -\dot{q}_i S_1 \\ \dot{q}_i C_1 \\ \dot{q}_1 \end{bmatrix} \quad (\boldsymbol{\omega}_i)_{\text{repère}(i)} = \begin{bmatrix} 0 \\ \dot{q}_i \\ \dot{q}_1 \end{bmatrix} \; i = 2,3,4 \qquad \textbf{(A. 1)}$$

En utilisant la formule itérative du calcul du modèle dynamique direct on en déduit les vitesses:

$$\dot{\mathbf{O}}_{i+1} = \dot{\mathbf{O}}_i + \boldsymbol{\omega}_i \times \mathbf{l}_i \qquad \textbf{(A. 2)}$$

Ou $\times$ représente le produit en croix des vecteurs.

A.2. Equations de la dynamique

Les équations de la dynamique donne la relation entre les positions, les vitesses, les accélérations, les forces et les couples d'un mécanisme comportant plusieurs corps. D'un point de vue purement automatique, les équations de Lagrange ou de Kane serait suffisante pour déterminer le meilleur couple moteur à appliquer afin de réaliser un mouvement donné. Malgré cela, nous nous sommes également intéressé à la marge de stabilité du manipulateur et au condition de glissement, c'est-à-dire à l'ensemble des phénomènes qui mettent en jeu les forces interne et les réactions du sol sur le robot. Pour cette raison, nous nous sommes basés sur le formalisme de Newton-Euler. Contrairement à la cinématique, la dynamique demande d'utiliser la formule itérative du calcul dynamique inverse, en allant de l'organe terminal jusqu'à la base. Nous allons maintenant montrer mathématiquement ce qui peut être deviné intuitivement grâce aux particularités de ce mécanisme : Un corps i est uniquement chargé par les masses des corps qui lui succèdent, du point de vue O_{i+1}.

Les paramètres du corps *i* sont :

- Sa masse m_i
- Le vecteur $\mathbf{d}_i$ situé au centre de gravité G_i: $\mathbf{d}_i = \overrightarrow{O_i G_i}$
- La matrice d'inertie $\mathbf{I}_i$. Ses composantes dans le repère local sont notées $(I_i)_x$, $(I_i)_y$, $(I_i)_z$, $(I_i)_{xy}$ etc.

La gravité (**g**) sera considérée comme verticale (le long de l'axe z). Bien sur, les trois composantes doivent être considérées pour des application en terrain accidenté.

Les équations de la dynamiques sont obtenues de façon itératives, en commençant depuis l'organe terminal (body 4) jusqu'à la base (body 1), selon [Luh 80].

$$\mathbf{F}_i = m_i(\ddot{\mathbf{O}}_i + \dot{\boldsymbol{\omega}}_i \times \mathbf{d}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \mathbf{d}_i)) \qquad \textbf{(A. 3)}$$

$$\mathbf{M}_i = \mathbf{I}_i \bullet \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\mathbf{I}_i \bullet \boldsymbol{\omega}_i) + m_i \mathbf{d}_i \times \ddot{\mathbf{O}}_i \qquad \textbf{(A. 4)}$$

Le vecteur $\mathbf{F}_i$ est la somme des forces agissant sur le corps *i* comme s'il était isolé. $\mathbf{M}_i$ est le couple résultant sur le corps *i*, calculé en $\mathbf{O}_i$. Le point $\bullet$ représente le produit de matrice. La première équation calculée au centre de gravité G_i peut remplacer l'équation A.3 :

$$\mathbf{F}_{G_i} = m_i \ddot{\mathbf{G}}_i \qquad \textbf{(A. 5)}$$

Les équations détaillées seront développées afin de démontrer que ce robot M^3 facilite les calcul du modèle dynamique, l'asservissement et les tests de stabilité. Les équation A.3 et A.4 sont exprimées dans le repère local (O_i, x_i, y_i, z_i).

A.2.1. Corps 4 (l'organe terminal et la charge)

Considérons que l'organe terminal puisse être sujet à une force $\mathbf{F}_e$ et un couple extérieurs $\mathbf{M}_e$ au point de référence O_5. Tous les composants des vecteurs et matrices sont exprimés dans le repère local (O_4, x_4, y_4, z_4) car les actions externes sont généralement liées soit à la gravité, soit à une source externe solidaire du sol. La force $\mathbf{F}_{34}$ et le moment $\mathbf{M}_{34}$ exercés par le corps 3 sur le corps 4 sont schématisés sur la Figure 65. Le couple $\mathbf{M}_{14}$ agit directement sur le corps au niveau de l'axe de rotation O_4y_4.

$$\mathbf{F}_{34} = -\mathbf{F}_e + m_4((\ddot{\mathbf{O}}_4 - \mathbf{g}) + \dot{\boldsymbol{\omega}}_4 \times \mathbf{d}_4 + \boldsymbol{\omega}_4 \times (\boldsymbol{\omega}_4 \times \mathbf{d}_4)) \qquad \textbf{(A. 6)}$$

$$\mathbf{M}_{34} + \mathbf{M}_{14} = -\mathbf{M}_e - \mathbf{F}_e \times \mathbf{l}_4 + m_4 \mathbf{d}_4 \times (\ddot{\mathbf{O}}_4 - \mathbf{g}) + \mathbf{I}_4 \bullet \dot{\boldsymbol{\omega}}_4 + \boldsymbol{\omega}_4 \times (\mathbf{I}_4 \bullet \boldsymbol{\omega}_4) \qquad \textbf{(A. 7)}$$

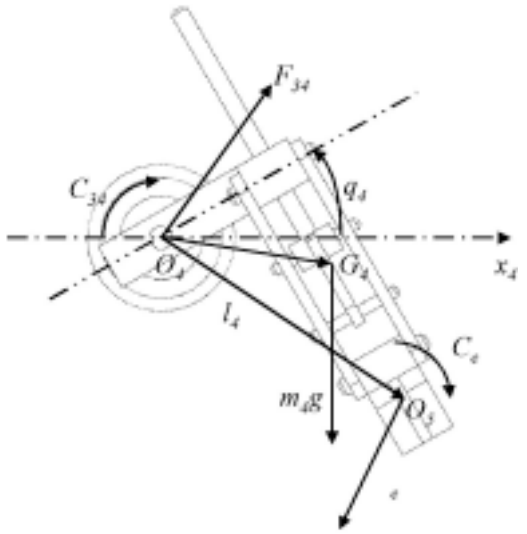


Figure 65 : Paramètres et forces externes relative au corps 4

L'équation du vecteur force (A.6) donne en premier lieu :

$$\begin{cases} (F_{34})_x = -(F_e)_x + m_4(\ddot{x}_{O_4} + \ddot{q}_4(-(d_4)_x S_4 + (d_4)_z C_4) - (\dot{q}_1^2 + \dot{q}_4^2)((d_4)_x C_4 + (d_4)_z S_4)) \\ (F_{34})_y = -(F_e)_y + m_4(\ddot{y}_{O_4} + \ddot{q}_1((d_4)_x C_4 + (d_4)_z S_4) + \dot{q}_1\dot{q}_4(-(d_4)_x S_4 + (d_4)_z C_4)) \\ (F_{34})_z = -(F_e)_z + m_4(\ddot{z}_{O_4} + g - \ddot{q}_4((d_4)_x C_4 + (d_4)_z S_4) - \dot{q}_4^2(-(d_4)_x S_4 + (d_4)_z C_4)) \end{cases} \quad \textbf{(A. 8)}$$

Remarquons que les mouvements des segments 2 et 3 n'induisent pas de force centrifuge sur le corps 4 grâce aux mécanisme de poulies remplaçant un double parallélogramme. Comme les équations sont écrites dans le repère de référence (O_4, x_4, y_4, z_4), la matrice d'inertie du corps 4 doit être calculée dans ce repère.
Considérons

$$(\mathbf{I}_4)_4 = \begin{bmatrix} I_{x4} & 0 & I_{xz4} \\ 0 & I_{y4} & 0 \\ I_{xz4} & 0 & I_{4z} \end{bmatrix} \quad \textbf{(A. 9)}$$

la matrice d'inertie quand q_4=0. Cela devient

$$\mathbf{I}_4 = \begin{bmatrix} C_4^2(I_{x4} + (I_{xload})_4) & 0 & C_4S_4(I_{z4} + (I_{zload})_4) - I_{x4} - (I_{xload})_4) \\ + S_4^2(I_{z4} + (I_{zload})_4) & & + (C_4^2 - S_4^2)(I_{xz4} + (I_{xzload})_4) \\ 0 & I_{y4} + (I_{yload})_4 & 0 \\ C_4S_4(I_{z4} + (I_{zload})_4) - I_{x4} - (I_{xload})_4) & 0 & C_4^2(I_{z4} + (I_{zload})_4) + S_4^2(I_{x4} + (I_{xload})_4) \\ + (C_4^2 - S_4^2)(I_{xz4} + (I_{xzload})_4) & & - 2C_4S_4(I_{xz4} + (I_{xzload})_4) \end{bmatrix}$$

quand le moment dû à la charge, le produit d'inertie et la rotation q_4 ont été ajoutées

Pour simplifier, nous noterons :

$$\mathbf{I}_4 = \begin{bmatrix} (I_4)_x & 0 & (I_4)_{xz} \\ 0 & (I_4)_y & 0 \\ (I_4)_{xz} & 0 & (I_4)_z \end{bmatrix} \qquad \textbf{(A. 10)}$$

L'équation A.6 donne :

$$\begin{cases} (M_{34})_x = -(M_e)_x - (-S_4(l_4)_z + C_4(l_4)_z)(F_e)_y - m_4\ddot{y}_{O_4}(-S_4(d_4)_z + C_4(d_4)_z) \\ + (I_4)_{xz}\ddot{q}_1 + ((I_4)_{xz} - (I_4)_x)\dot{q}_1\dot{q}_4 \\ (M_{34})_y = 0 \\ (M_{14})_y = -(M_e)_y + (-S_4(l_4)_z + C_4(l_4)_z)(F_e)_x - (S_4(l_4)_z + C_4(l_4)_z)(F_e)_z \\ + m_4(\ddot{x}_{O_4}(-(d_4)_x S_4 + (d_{4'})_z C_4) \\ -(\ddot{z}_{O_4} + g)((d_4)_x C_4 + (d_4)_z S_4)) + (I_4)_y\ddot{q}_4 + (I_4)_{xz}\dot{q}_1^2 \\ (M_{34})_z = -(M_e)_z - (C_4(l_4)_x + S_4(l_4)_z)(F_e)_y + m_4\ddot{y}_{O_4}((d_4)_x C_4 + (d_4)_z S_4) \\ + (I_4)_{zz}\ddot{q}_1 - (I_4)_{xz}\dot{q}_1\dot{q}_4 \end{cases} \qquad \textbf{(A. 11)}$$

Aucune composante du couple $(M_{34})_y$ n'est transmise du corps 3 vers le corps 4, mais le mécanisme de poulies et de courroies transmettent $(M_{14})_y$ depuis l'actionneur fixé sur la base.

A.2.2. Corps 3 (l'avant bras) et 2 (le bras)

A.2.2.1. L'avant bras

Les équations A.3 et A.4 donnent :

$$\mathbf{F}_{23} = m_3(\ddot{\mathbf{O}}_3 - \mathbf{g} + \dot{\boldsymbol{\omega}}_3 \times \mathbf{d}_3 + \boldsymbol{\omega}_3 \times (\boldsymbol{\omega}_3 \times \mathbf{d}_3)) + \mathbf{F}_{34} + \mathbf{F}_{spring(3)} \qquad \textbf{(A. 12)}$$

$$\mathbf{M}_{23} = \mathbf{I}_3 \bullet \dot{\boldsymbol{\omega}}_3 + \boldsymbol{\omega}_3 \times (\mathbf{I}_3 \bullet \boldsymbol{\omega}_3) + m_3\mathbf{d}_3 \times (\ddot{\mathbf{O}}_3 - \mathbf{g}) + \mathbf{l}_3 \times \mathbf{F}_{34} + \mathbf{M}_{34} + \mathbf{l}_{spring(3)} \times \mathbf{F}_{spring(3)} \qquad \textbf{(A. 13)}$$

On en déduit les composantes suivantes, qui peuvent être facilement calculées puisque tous les repères mobiles sont parallèles :

$$\begin{cases} (F_{23})_x = m_3(\ddot{x}_{O_3} - \ddot{q}_3 d_3 S_3 - (\dot{q}_1^2 + \dot{q}_3^2)d_3 S_3) + (F_{34})_x - (F_{spring(3)})_x \\ (F_{23})_y = m_3(\ddot{y}_{O_3} + \ddot{q}_1 d_3 C_3 - \dot{q}_1\dot{q}_3 d_3 S_3) + (F_{34})_y \\ (F_{23})_z = m_3(\ddot{z}_{O_3} + g - \ddot{q}_3 d_3 C_3 + \dot{q}_3^2 d_3 S_3) + (F_{34})_z - (F_{spring(3)})_z \end{cases} \qquad \textbf{(A. 14)}$$

$$\begin{cases} (M_{23})_x = (I_3)_{xz}\ddot{q}_1 + ((I_3)_z - (I_3)_x)\dot{q}_1\dot{q}_3 - m_3 d_3 S_3 \ddot{y}_{O_3} + l_3 S_3 (F_{34})_y + (M_{34})_x \\ (M_{23})_y = 0 \\ (M_{13})_y = (I_3)_y \ddot{q}_3 + (I_3)_{xz}\dot{q}_1^2 - m_3 d_3 (S_3 \ddot{x}_{O_3} + C_3(\ddot{z}_{O_3} + g)) \\ + l_3(-S_3(F_{34})_x + C_3(F_{34})_y) + (M_{34})_y + (-(l_{spring(3)})_x S_3 + (l_{spring(3)})_z C_3)(F_{spring(3)})_x \\ -((l_{spring(3)})_x C_3 + (l_{spring(3)})_z S_3)(F_{spring(3)})_z \\ (M_{23})_z = (I_3)_z \ddot{q}_1 - (I_3)_{xz}\dot{q}_1\dot{q}_3 + m_3 d_3 C_3 \ddot{y}_{O_3} + l_3 C_3 (F_{34})_y + (M_{34})_z \end{cases} \quad \textbf{(A. 15)}$$

où les inerties sont :

$$\mathbf{I}_3 = \begin{bmatrix} (I_3)_x & 0 & (I_3)_{xz} \\ 0 & (I_3)_y & 0 \\ (I_3)_{xz} & 0 & (I_3)_z \end{bmatrix} = \begin{bmatrix} C_3^2 I_{x3} + S_3^2 I_{z3} & 0 & S_3 C_3 (I_{z3} - I_{x3}) \\ 0 & I_{y3} & 0 \\ S_3 C_3 (I_{z3} - I_{x3}) & 0 & S_3^2 I_{x3} + C_3^2 I_{z3} \end{bmatrix} \quad \textbf{(A. 16)}$$

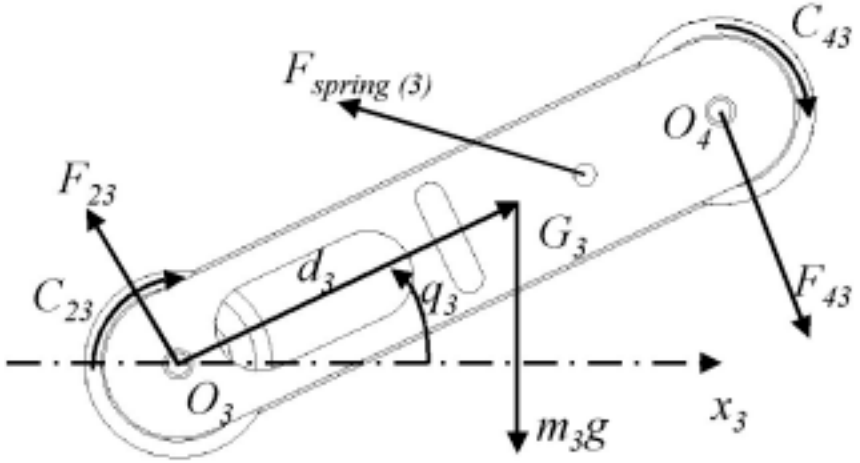


Figure 66 : Paramètres et forces relatifs au corps 3

La raideur et les points de fixations du ressort "spring(3)" ont été calculé de façon à compenser au maximum le poids du corps 3 (à G_3) et du corps 4 (à O_4). La Figure 67 montre les couples relatifs à q_3.

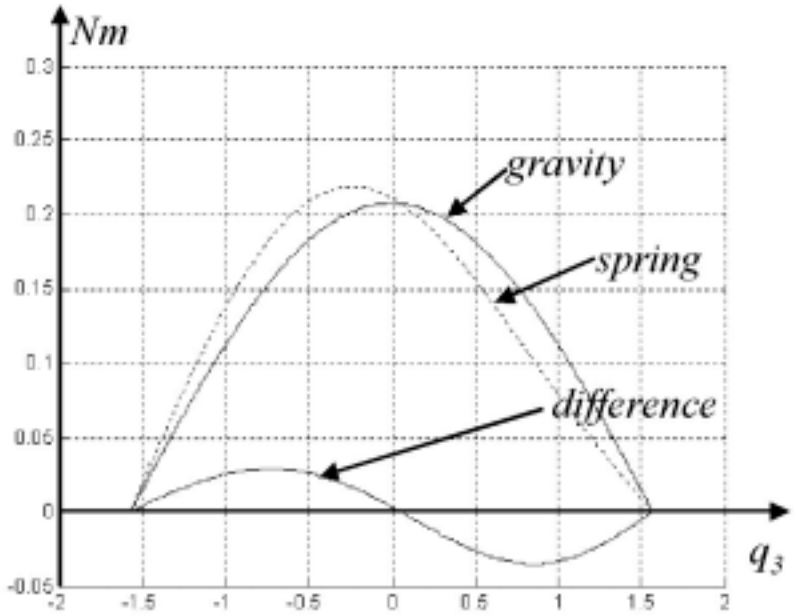


Figure 67 : Compensation du poids du corps 3

A.2.2.2. Le bras

Les forces et les couples appliqués sur le corps 2 sont définis sur la Figure 68.

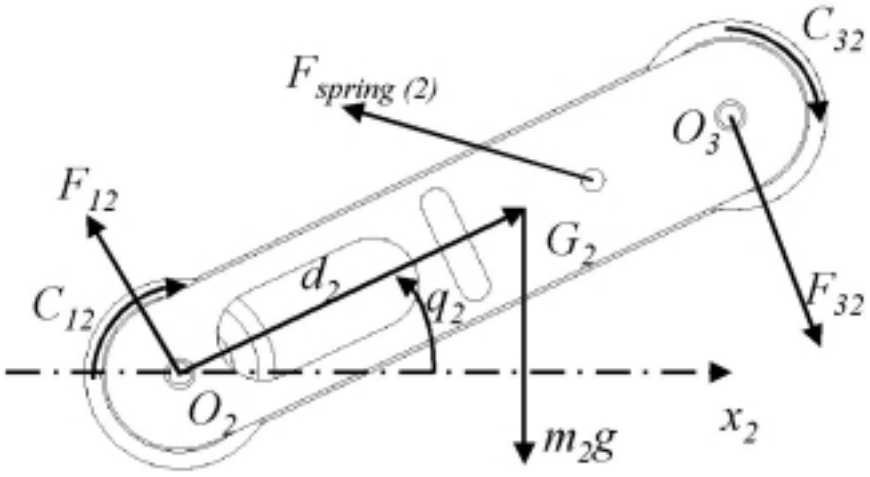


Figure 68 : Paramètres et forces relatifs au corps 2

Les équations A.3 et A.4 donnent :

$$\mathbf{F}_{12} = m_2(\ddot{\mathbf{O}}_2 - \mathbf{g} + \dot{\boldsymbol{\omega}}_2 \times \mathbf{d}_2 + \boldsymbol{\omega}_2 \times (\boldsymbol{\omega}_2 \times \mathbf{d}_2)) + \mathbf{F}_{23} + \mathbf{F}_{spring(2)} \quad \textbf{(A. 17)}$$

$$\mathbf{M}_{12} = \mathbf{I}_2 \bullet \dot{\boldsymbol{\omega}}_2 + \boldsymbol{\omega}_2 \times (\mathbf{I}_2 \bullet \boldsymbol{\omega}_2) + m_2 \mathbf{d}_2 \times (\ddot{\mathbf{O}}_2 - \mathbf{g}) + \mathbf{l}_2 \times \mathbf{F}_{23} + \mathbf{M}_{23} + \mathbf{l}_{spring(2)} \times \mathbf{F}_{spring(2)} \quad \textbf{(A. 18)}$$

On en déduit les composantes suivantes :

$$\begin{cases} (F_{12})_x = m_2(\ddot{x}_{O_3} - \ddot{q}_2 d_2 S_2 - (\dot{q}_1^2 + \dot{q}_2^2) d_2 S_2) + (F_{23})_x - (F_{spring(2)})_x \\ (F_{12})_y = m_2(\ddot{y}_{O_3} + \ddot{q}_1 d_2 C_2 - \dot{q}_1 \dot{q}_2 d_2 S_2) + (F_{23})_y \\ (F_{12})_z = m_2(\ddot{z}_{O_3} + g - \ddot{q}_2 d_2 C_2 + \dot{q}_2^2 d_2 S_2) + (F_{23})_z - (F_{spring(2)})_z \end{cases} \quad \textbf{(A. 19)}$$

$$\begin{cases} (M_{12})_x = (I_2)_{xz} \ddot{q}_1 + ((I_2)_z - (I_2)_x) \dot{q}_1 \dot{q}_2 - m_2 d_2 S_2 \ddot{y}_{O_3} + l_2 S_2 (F_{23})_y + (M_{23})_x \\ (M_{23})_y = 0 \\ (M_{12})_y = (I_2)_y \ddot{q}_2 + (I_2)_{xz} \dot{q}_1^2 - m_2 d_2 (S_2 \ddot{x}_{O_3} + C_2(\ddot{z}_{O_3} + g)) \\ + l_2(-S_2(F_{23})_x + C_2(F_{23})_y) + (M_{23})_y + (-(l_{spring(2)})_x S_2 + (l_{spring(2)})_z C_2)(F_{spring(2)})_x \\ -((l_{spring(2)})_x C_2 + (l_{spring(2)})_z S_2)(F_{spring(2)})_z \\ (M_{12})_z = (I_2)_z \ddot{q}_1 - (I_2)_{xz} \dot{q}_1 \dot{q}_2 + m_2 d_2 C_2 \ddot{y}_{O_3} + l_2 C_2 (F_{23})_y + (M_{23})_z \end{cases} \quad \textbf{(A. 20)}$$

où les inertie sont :

$$\mathbf{I}_2 = \begin{bmatrix} (I_2)_x & 0 & (I_2)_{xz} \\ 0 & (I_2)_y & 0 \\ (I_2)_{xz} & 0 & (I_2)_z \end{bmatrix} = \begin{bmatrix} C_2^2 I_{x2} + S_2^2 I_{z2} & 0 & S_2 C_3 (I_{z2} - I_{x2}) \\ 0 & I_{y2} & 0 \\ S_2 C_2 (I_{z2} - I_{x2}) & 0 & S_2^2 I_{x2} + C_2^2 I_{z2} \end{bmatrix} \quad \textbf{(A. 21)}$$

Comme pour le corps 3, la raideur et la fixation du ressort "spring(2)" ont été calculer de façon à compenser le poids du corps 2 (à G_2) et des corps 3 et 4 (à O_3). Les résultats sont proches de ceux de l'avant bras.

A.3.3. Corps 1 (la base mobile)

A.3.3.1 les équations du déplacement

Les différentes forces qui sont appliquées sur la base sont isolées : force inertiel, moment d'inertie, poids, interactions depuis le manipulateur, force de traction et réaction du sol. L'influence gyroscopique des moteurs est négligée. La géométrie et les forces sont illustrées sur la Figure 69.

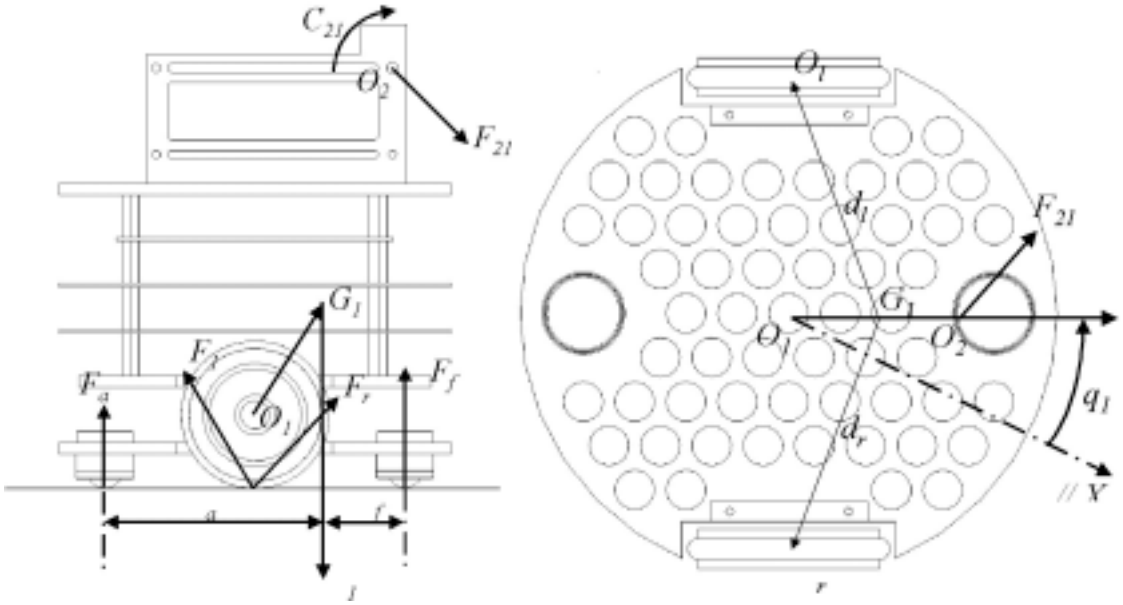
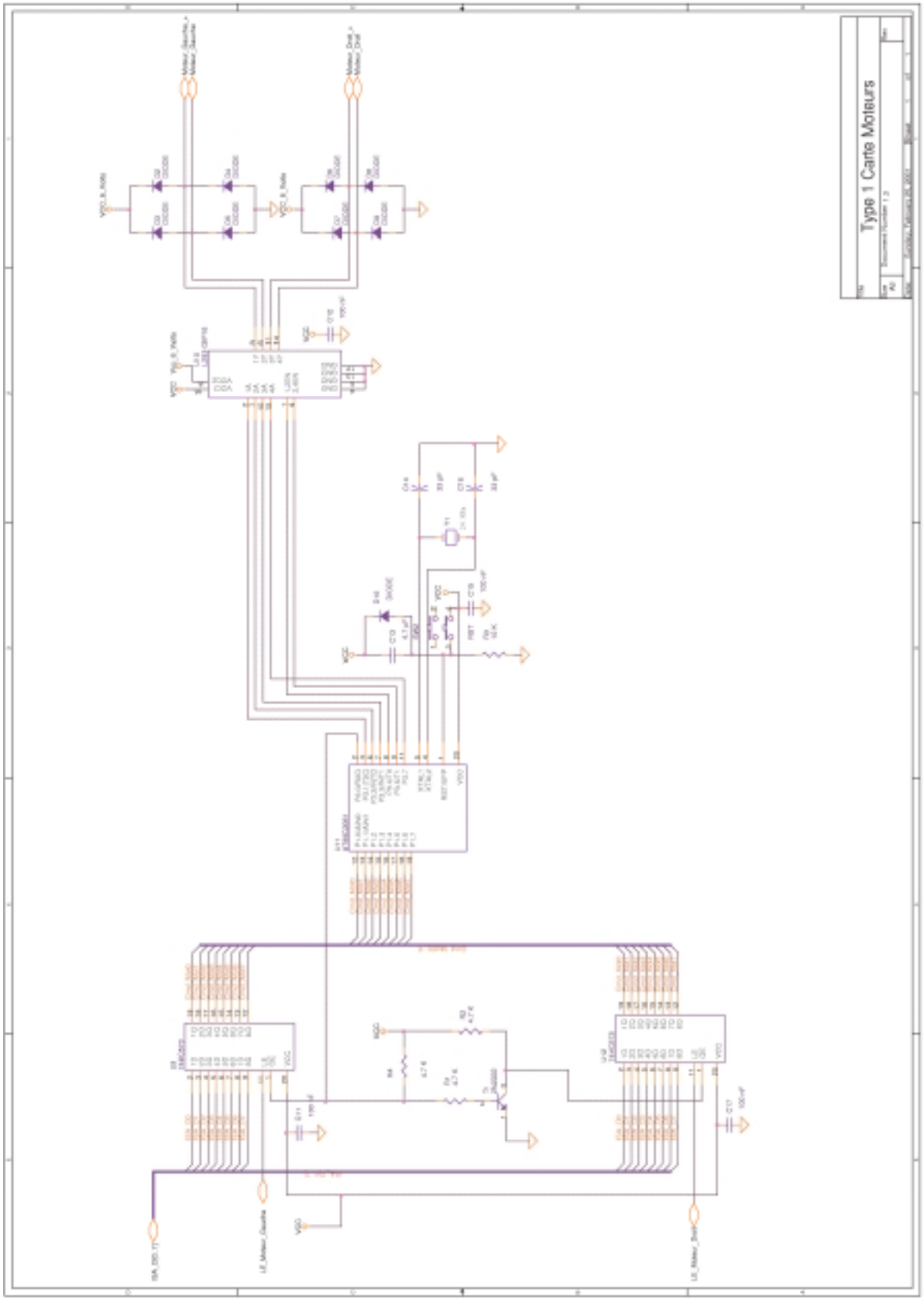


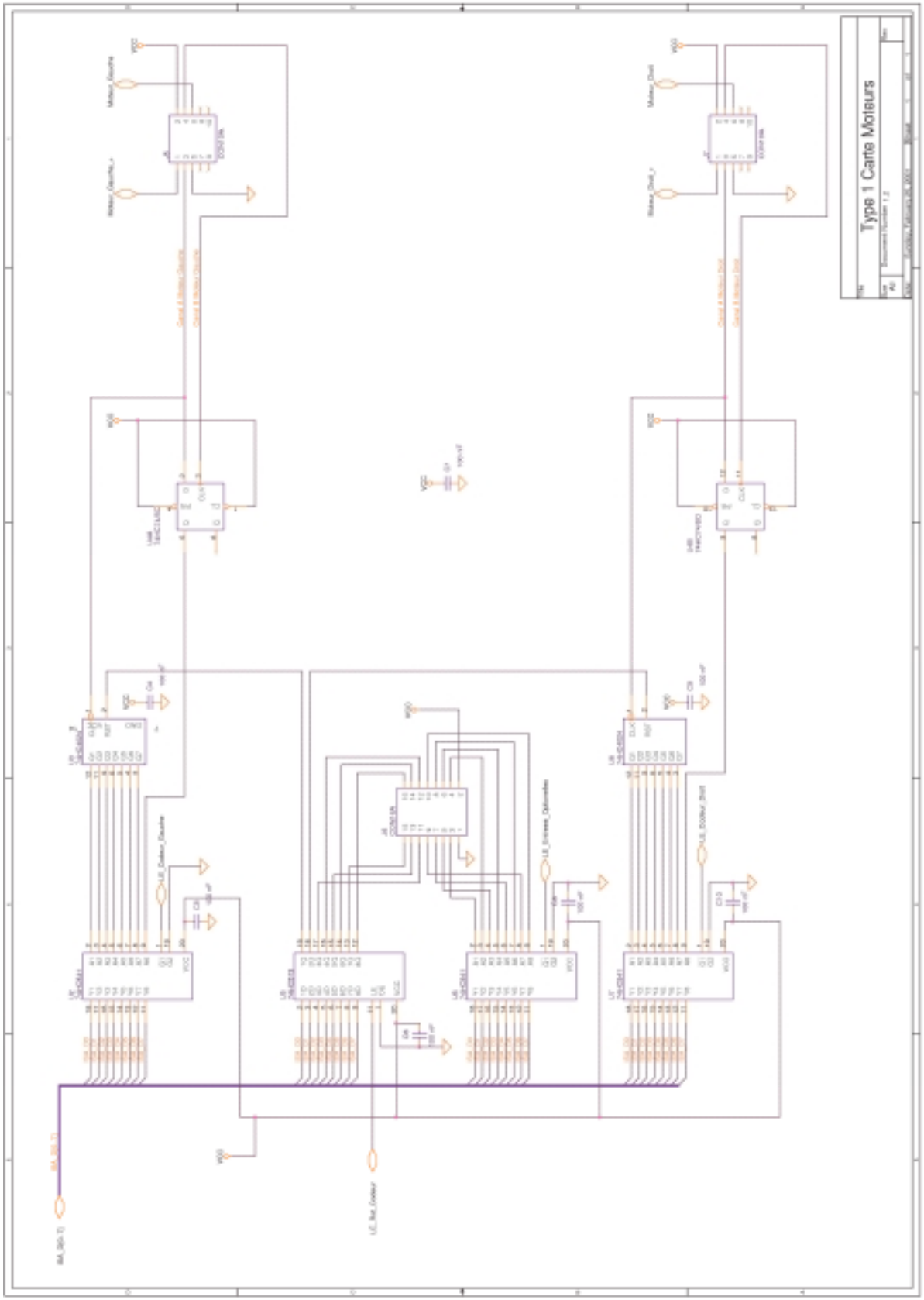
Figure 69 : Repères et paramètre de la base mobile.

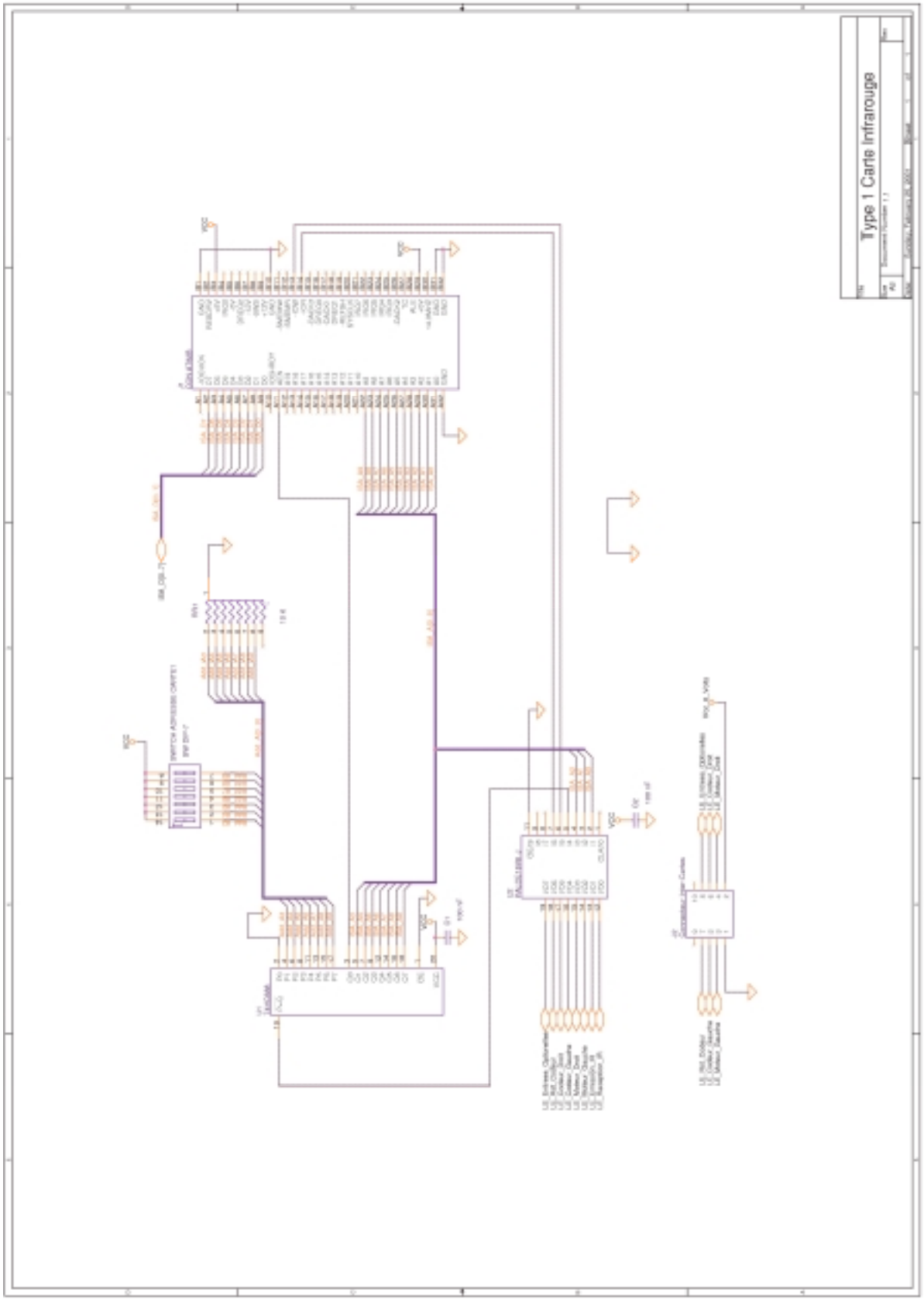
Les équations sont données par :

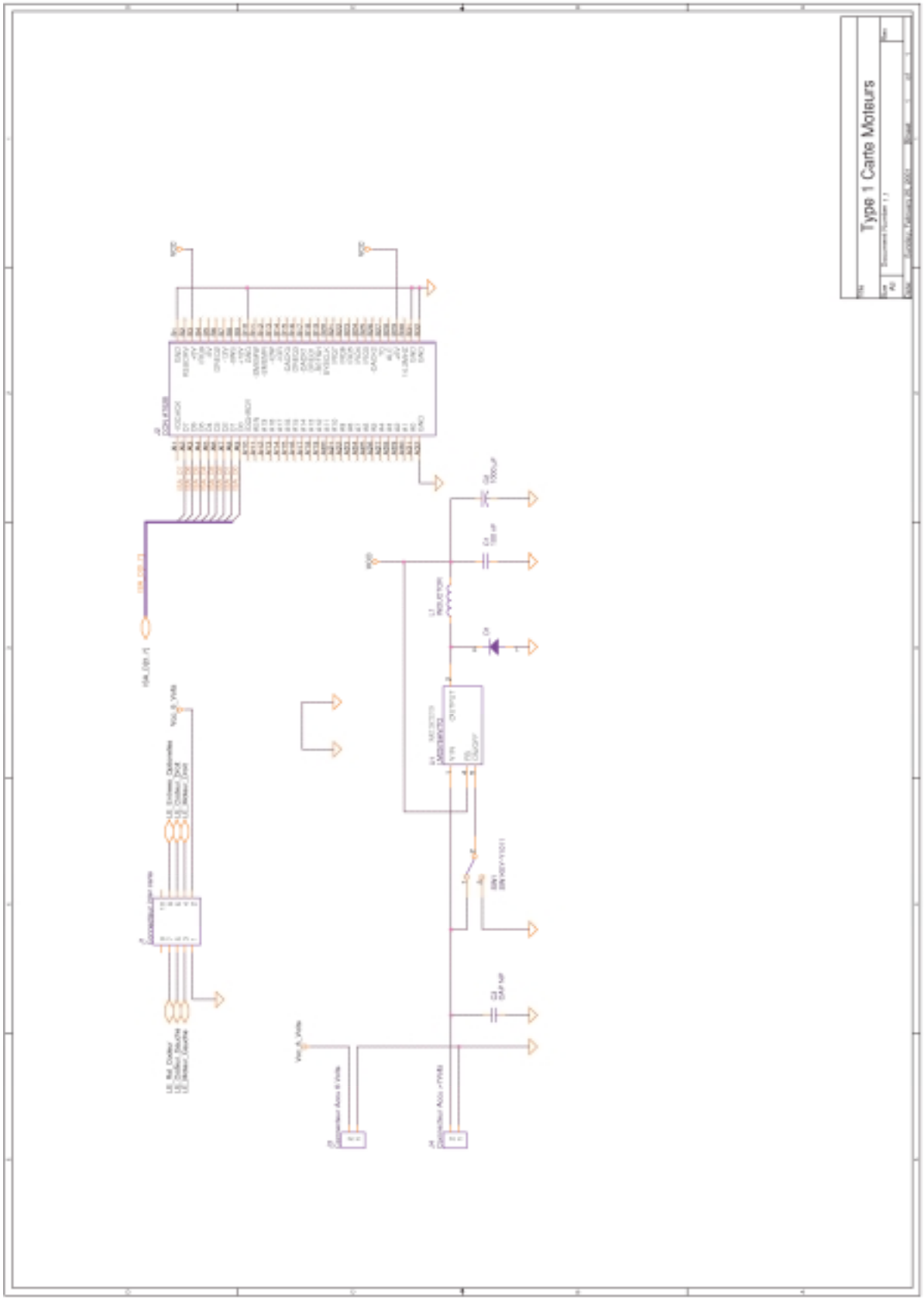
$$\begin{cases} m_1 \ddot{X}_{G_1} = ((F_{21})_x + \dfrac{1}{R}(M_r + M_l))C_1 \\ m_1 \ddot{Y}_{G_1} = ((F_{21})_y + \dfrac{1}{R}(M_r + M_l))S_1 \\ (I\ )_z \ddot{q}_1 = R\dfrac{\dot{\omega}_r - \dot{\omega}_l}{2d} = (M_{21})_z + \dfrac{d}{R}(M_r - M_l) \\ + (F_{21})_x((l_1)_y - (d_1)_y) - (F_{21})_y((l_1)_x - (d_1)_x) \end{cases} \quad \textbf{(A. 22)}$$

Annexe B : Schéma électronique du robot Type 1


Type 1 Carte Moteurs


Type 1 Carte Moteurs


Type 1 Carte Infrarouge


Type 1 Carte Moteurs

Philippe LUCIDARME

APPRENTISSAGE ET ADAPTATION POUR DES ENSEMBLES DE ROBOTS REACTIFS COOPERANTS

Résumé : Ces travaux de thèse se placent dans le contexte des systèmes multi-agents distribués. L'objectif est l'étude de méthodes d'auto-apprentissage appliquées à des ensembles de robots réactifs. Ces travaux se focalisent sur l'apprentissage de comportements sensori-moteurs de bas niveaux.

Il nous semble important que les méthodes proposées puissent être appliquées sur des systèmes réels, dont les contraintes sont parfois loin de celles de la simulation. C'est pour cette raison que nous avons imaginé et conçu une plate-forme expérimentale composée de 4 robots mobiles, un manipulateur mobile miniature et un système de vision stéréoscopique.

Cette étude se décompose en deux parties. La première, appliquée aux systèmes homogènes, présente l'étude de méthodes évolutionnistes appliquées aux systèmes multi-robots. La seconde, appliquée aux systèmes hétérogènes, s'intéresse à la possibilité d'utiliser la technique du recuit simulé pour optimiser les poids d'un contrôleur neuronal. Toujours dans ce contexte d'hétérogénéité, une seconde méthode basée sur l'apprentissage par renforcement est expérimentée.

Mots-clefs : Systèmes multi-robots, apprentissage, réseaux de neurones artificielles, algorithmes évolutionnistes, apprentissage par renforcement, recuit simulé, manipulateur mobile miniature.

LEARNING AND ADAPTATION FOR GROUPS OF REACTIVE AND COOPERANT ROBOTS

Abstract : The aim of this work is to build fault tolerant cooperative multi-robots systems. Our approach uses self-learning techniques to control groups of reactive robots. This work focuses on learning low level sensory-motor behaviors.

It seems important that the proposed methods may be implemented on real robots. The constraint of such real systems is sometime far from simulated worlds. This is why we imagined, designed and build an experimental platform composed of four mobile robots, one miniature mobile manipulator and one stereoscopic vision system.

This study is composed of two parts. The first one is applied to homogeneous systems. Evolutionist techniques are studied.

The second one, applied to heterogeneous systems, focuses on using simulated annealing procedure to optimize the synaptic weights of a neuro-controller. Another method is also experimented, based on reinforcement learning.

Keywords : Multi-robot systems, learning, artificial neural networks, evolutionist algorithms, reinforcement learning, simulated annealing, miniature mobile manipulator.

Université Montpellier II