Apprentissage de nouveaux comportements: vers le développement épigénétique d'un robot autonome.

Matthieu Lagarde, Philippe Gaussier, Pierre Andry

Laboratoire ETIS

CNRS UMR8051, ENSEA, Université de Cergy-Pontoise

THÈSE

présentée pour obtenir le titre de Docteur en Sciences et technologies de l'information et de la communication.

Apprentissage de nouveaux comportements: vers le développement épigénétique d'un robot autonome.

Matthieu Lagarde

E-mail : matthieu.lagarde@ensea.fr

Soutenue le 13 juillet 2010 devant le jury composé de :

Dr Agnès Guillot (Université Paris Ouest, Paris 10),	Rapporteur
Pr Peter Dominey (INSERM U846, Bron (69)),	Rapporteur
Pr Aude Billard (Ecole Polytechnique Fédérale de Lausanne),	Examinatrice
Dr Nicolas Bredeche (Université Paris-Sud, Paris 11),	Examinateur
Dr Jean-Christophe Baillie (Gostai, Paris),	Examinateur
Pr Philippe Gaussier (Université de Cergy-Pontoise),	Directeur de thèse
Dr Pierre Andry (Université de Cergy-Pontoise),	Encadrant de thèse

Remerciements

Je tiens tout d'abord à exprimer mes sincères remerciements à mon directeur de thèse Philippe Gaussier et mon encadrant de thèse Pierre Andry pour leur soutien qu'ils m'ont accordé, tout ce qu'ils m'ont apporté à travers les nombreuses discussions passionnantes et pour leur amitié durant ces merveilleuses années de thèse.

Je remercie vivement Peter Dominey d'avoir accepté de présider mon jury et, avec Agnès Guillot, d'avoir accepté de rapporter ce travail malgrè les faiblesses dont il souffre. Je souhaite également remercier les autres membres du jury : Aude Billard, Nicolas Bredèche et Jean-Christophe Baillie, pour le temps consacré à évaluer mes travaux.

J'aimerai également remercier tous les membres de l'équipe de Neurocybernétique qui ont chacun contribué à l'aboutissement de ce travail. J'ai trouvé au sein de cette équipe un environnement de travail stimulant qui m'a permis de découvrir le domaine des neurosciences computationnelles et de librement mener des expériences sur de nombreux matériels robotiques.

Je tiens à remercier Inbar Fijalkow de m'avoir accueilli en tant que directrice du laboratoire au laboratoire ETIS, dans lequel j'ai trouvé une ambiance particulièrement favorable.

Je souhaite également exprimer mes sincères remerciements à mes amis et collègues avec qui se fut un plaisir de travailler et qui m'ont supporté : Mickael M, Nicolas C, Christophe G, Sofiane B, Cyril H, Julien H, Frederic D.M, Benoit M, Adrian, Arnaud B, Philippe P, Mathias Q, Jean-Paul B, LaurenceH, Patrick H, Laurent P, Michel J, . . .ainsi que tous ceux que j'ai oublié.

Enfin, je remercie également ma famille, famille de ma compagne et ma compagne Chloé pour leur soutien dans les finitions du manuscrit.

Résumé

La problématique de l'apprentissage de comportements sur un robot autonome soulève de nombreuses questions liées au contrôle moteur, à l'encodage du comportement, aux stratégies comportementales et à la sélection de l'action. Utiliser une approche développementale présente un intérêt tout particulier dans le cadre de la robotique autonome. Le comportement du robot repose sur des mécanismes de bas niveau dont les interactions permettent de faire émerger des comportements plus complexes. Le robot ne possède pas d'informations a priori sur ses caractéristiques physiques ou sur l'environnement, il doit apprendre sa propre dynamique sensori-motrice. J'ai débuté ma thèse par l'étude d'un modèle d'imitation bas niveau. Du point de vue du développement, l'imitation est présente dès la naissance et accompagne, sous de multiples formes, le développement du jeune enfant. Elle présente une fonction d'apprentissage et se révèle alors être un atout en terme de temps d'acquisition de comportements, ainsi qu'une fonction de communication participant à l'amorce et au maintien d'interactions non verbales et naturelles. De plus, même s'il n'y a pas de réelle intention d'imiter, l'observation d'un autre agent permet d'extraire suffisamment d'informations pour être capable de reproduire la tâche.
Mon travail a donc dans un premier temps consisté à appliquer et tester un modèle développemental qui permet l'émergence de comportements d'imitation de bas niveau sur un robot autonome. Ce modèle est construit comme un homéostat qui tend à équilibrer par l'action ses informations perceptives frustres (détection du mouvement, détection de couleur, informations sur les angles des articulations d'un bras de robot). Ainsi, lorsqu'un humain bouge sa main dans le champ visuel du robot, l'ambiguïté de la perception de ce dernier lui fait confondre la main de l'humain avec l'extrémité de son bras. De l'erreur qui en résulte émerge un comportement d'imitation immédiate des gestes de l'humain par action de l'homéostat. Bien sûr, un tel modèle implique que le robot soit capable d'associer au préalable les positions visuelles de son effecteur avec les informations proprioceptives de ses moteurs. Grâce au comportement d'imitation, le robot réalise des mouvements qu'il peut ensuite apprendre pour construire des comportements plus complexes. Comment alors passer d'un simple mouvement à un geste plus complexe pouvant impliquer un objet ou un lieu ?
Je propose une architecture qui permet à un robot d'apprendre un comportement sous forme de séquences temporelles complexes (avec répétition d'éléments) de mouvements. Deux modèles différents permettant l'apprentissage de séquences ont été développés et testés. Le premier apprend en ligne le timing de séquences temporelles simples. Ce modèle ne permettant pas d'apprendre des séquences complexes, le second modèle testé repose sur les propriétés d'un réservoir de dynamiques, il apprend en ligne des séquences complexes. A l'issue de ces travaux, une architecture apprenant le timing d'une séquence complexe a été proposée. Les tests en simulation et sur robot ont montré la nécessité d'ajouter un mécanisme de resynchronisation permettant de retrouver les bons états cachés pour permettre d'amorcer une séquence complexe par un état intermédiaire.
Dans un troisième temps, mes travaux ont consisté à étudier comment deux stratégies sensori-motrices peuvent cohabiter dans le cadre d'une tâche de navigation. La première stratégie encode le comportement à partir d'informations spatiales alors que la seconde utilise des informations temporelles. Les deux architectures ont été testées indépendamment sur une même tâche. Ces deux stratégies ont ensuite été fusionnées et exécutées en parallèle. La fusion des réponses délivrées par les deux stratégies a été réalisée avec l'utilisation de champs de neurones dynamiques. Un mécanisme de "chunking" représentant l'état instantané du robot (le lieu courant avec l'action courante) permet de resynchroniser les dynamiques des séquences temporelles.
En parallèle, un certain nombre de problème de programmation et de conception des réseaux

de neurones sont apparus. En effet, nos réseaux peuvent compter plusieurs centaines de milliers de neurones. Il devient alors difficile de les exécuter sur une seule unité de calcul. Comment concevoir des architectures neuronales avec des contraintes de répartition de calcul, de communications réseau et de temps réel ? Une autre partie de mon travail a consisté à apporter des outils permettant la modélisation, la communication et l'exécution en temps réel d'architecture distribuées. Pour finir, dans le cadre du projet européen Feelix Growing, j'ai également participé à l'intégration de mes travaux avec ceux du laboratoire LASA de l'EPFL pour l'apprentissage de comportements complexes mêlant la navigation, le geste et l'objet.
En conclusion, cette thèse m'a permis de développer à la fois de nouveaux modèles pour l'apprentissage de comportements - dans le temps et dans l'espace, de nouveaux outils pour maîtriser des réseaux de neurones de très grande taille et de discuter à travers les limitations du système actuel, les éléments importants pour un système de sélection de l'action.

Abstract

The problem of learning behaviors on an autonomous robot raises many issues related to motor control, behavior encoding, behavioral strategies and action selection. Using a developmental approach is of particular interest in the context of autonomous robotics. The behavior of the robot is based on low level mechanisms that together can make more complex behaviors emerge. Moreover, the robot has no *a priori* information about its own physical characteristics or on its environment, it must learn its own sensori-motor dynamic. For instance, I started my thesis by studying a model of low level imitation. From a developmental point of view, imitation is present from birth and accompanies the development of young children under multiple forms. It has a learning function and shows up as an asset in term of performance in time of behaviors acquisition, as well as a communication function playing a role in the bootstrap and the maintenance of nonverbal and natural interactions. Moreover, even if there is not a real intention to imitate, the observation of another agent allows to extract enough information to be able to reproduce the task.

Initially, my work consisted in applying and testing a developmental model allowing emergence of low level imitation behaviors on an autonomous robot. This model is built like a homeostatic system which tends to balance its rough perceptive information (movement detection, color detection, angular information from motors of a robotic arm) by its action. Thus, when a human moves his hand in the robot visual field, the perception ambiguity of the robot makes it consider the human hand as its own arm extremity. From the resulting error a immediate imitation behavior emerges. Of course, such a model implies that the robot is initially able to associate the visual positions of its effector with the proprioceptive informations of its motors. Thanks to imitation behavior, the robot makes movements from which it can learn to build more complex behaviors. Then, how to go from a simple movement to a more complex gesture which can imply an object or a place ?

I then proposed an architecture allowing a robot to learn a behavior as a complex temporal sequences (with repetition of elements) of movements. Two models allowing to learn sequences have been developed and tested. The first, based on a model of the hippocampus, learns on-line the timing of simple temporal sequences. The second, based on the properties of a dynamic reservoir, learns on-line complex temporal sequences. Based on these works, an architecture learning the timing of a complex temporal sequence has been proposed. The tests in simulation and on actual robot have shown the necessity to add a resynchronization mechanism that allows to find the correct hidden states for starting a complex sequence by an intermediate state.

In a third time, my work consisted in studying how two sensori-motor strategies can cohabit in the context of a navigation task. The first strategy codes the behavior from spatial informations, then the second uses temporal informations. Both architectures have been independently tested on the same task. Then, both strategies were merged and executed in parallel. Responses of both strategies were merged with the use of dynamical neural filed. A mechanism of "chunking" which represents the instantaneous state of the robot (current place with current action) allows to resynchronize the temporal sequences dynamics.

In parallel, a number of programming and design problems about neural networks have appeared. In fact, our networks can be made of many hundreds of thousands of neurons. It becomes hard to execute them on one computational unit. How to design neural architectures with parallel computation, network communication and real time constraints ? Another part of my work consisted in providing tools allowing the design, communication and real time execution of distributed architectures. Finally, in the context of the Feelix Growing European project, I

contribute to integrate my work with those of the LASA laboratory of EPFL for the learning of complex behaviors mixing navigation, gesture and object.
To conclude, this thesis allowed me to develop new models for learning behaviors - in time and in space, new tools to handle very large neural networks, and to discuss, beyond limitations of the current system, the important elements for an action selection system.

Table des matières

Chapitre 1

Introduction

Cette thèse a pour objectif d'illustrer comment un robot autonome peut apprendre des comportements complexes. Mes travaux cherchent à montrer comment à partir des interactions de mécanismes de bas niveau, des comportements plus complexes peuvent émerger. Ainsi je montrerai comment une architecture neuronale inspirée des structures du cerveau des mammifères, peut permettre à un robot d'apprendre à partir de sa propre dynamique une tâche combinant la navigation, l'objet et le geste. Nous verrons comment l'ajout de dynamiques internes peut permettre à un robot Aibo d'apprendre le timing de séquences complexes de gestes. Nous étudierons à travers une tâche de navigation sur un robot mobile équipé d'une caméra et d'une boussole électronique, comment cette première boucle sensori-motrice peut cohabiter avec une seconde apprenant des associations lieux-actions. J'introduirai un mécanisme de chunks pour resynchroniser les dynamiques temporelles à partir des informations spatiales. Cette thèse soulève de nombreuses questions liées, d'une part au contrôle moteur, à l'encodage des comportements, aux stratégies comportementales et à la sélection de l'action et d'autre part, aux outils permettant la conception, la communication et l'exécution en temps réel d'architectures parallèles.
Au cours de la seconde moitié du XX^{e} siècle, l'informatique à connu un essor fulgurant. La puissance de calcul et la quantité de mémoire disponible ne cessent de croître permettant d'exécuter des algorithmes de plus en plus complexes. De plus, les avancées en miniaturisation des circuits électroniques permettent de facilement embarquer de plus en plus de puissance de calcul. Néanmoins, cette miniaturisation se heurte à des limites physiques, c'est pourquoi ces dix dernières années le principal facteur de la croissance de la puissance de calcul est le nombre d'unités de calcul (processeur multi coeurs, réseau d'ordinateurs). Néanmoins, cette multiplication des unités de calcul impose de nouvelles méthodes de programmation pour réaliser du calcul parallèle tout en gardant l'intégrité des données traitées. En effet, la programmation parallèle soulève des problèmes liés à la concurence des accés aux données, mais également à la communication des différentes unités de calcul pour réaliser un traitement cohérent.
L'augmentation de la puissance de calcul permet l'exécution d'applications de plus en plus complexes. L'intelligence artificielle bénéficie donc de cette puissance pour réaliser plus rapidement des traitements sur de nombreux paramètres. Néanmoins, selon l'approche adoptée, les besoins en terme de performance ne sont pas nécessairement la puissance de calcul disponible en tant que telle, mais comment les données sont traitées. Dans le cadre de la robotique, la conception d'un contrôleur peut être réalisé suivant différentes approches.
La première, est la décomposition du contrôleur en modules successifs. Cette approches consiste en une succession de modules traitant les flux de données provenant des capteurs du robot pour finalement appliquer les commandes aux moteurs. Par exemple, les senseurs percoivent certaines informations de l'environnement. Ces données sont transmises à un module de modélisation qui les tranforme en une représentation interne de l'environnement. Cette représentation est ensuite transmise à un module de planification qui fournit les actions souhaitées. Finallement, ces actions sont transmises à un module de contrôle des moteurs pour les appliquer. Une telle approche demande alors une importante puissance de calcul pour que ce traitement en série prenne un temps suffisament faible pour permettre un contrôle cohérent et sûr du robot (le contrôleur prend une minute de calcul pour éviter un obstacle proche, il y aura collision avant même que les actions appropriées soient effectivement appliquées.).
Une seconde approche est apportée par les architectures de subsomption [Brooks, 1986]. Chaque module de cette architecture est un comportement. Chacun des modules reçoit en entrée les informations provenant des capteurs et fournit en sortie les actions souhaitées. Pour sélectionner l'action finale et appliquer les commandes motrices correspondantes, chaque comportement possède un niveau de priorité. Supposons un contrôleur composé de deux modules exécutés en

parallèle : un premier de basse priorité qui lorsque le robot ne perçoit rien de particulier qui fournit l'action d'avancer tout droit, et un second module de plus haute priorité dont le comportement est de se diriger vers de la nourriture lorsqu'il en détecte. Alors le comportement du robot sera par défaut d'avancer tout droit. Lorsqu'il détectera de la nourriture, il se dirigera vers celle-ci. Un avantage notable de la parallèlisation des comportements est que si l'un est en défaut, le robot peut continuer dans une moindre mesure, à évoluer dans l'environnement, piloté par un autre. Une telle architecture nécessite plutôt des capacités de calculs en parallèle pour efficacement réaliser les traitements de chacun des modules.
Chacune des deux approches ont leur philosophie. Dans la première, un module réalise une partie d'un traitement, alors que dans la seconde, un module est un comportement parmi d'autres dans un répertoire hierarchisé. Dans le cadre de la robotique épigénétique, nos architectures de contrôle s'inspirent du système nerveux animal. Une question qui se pose est : comment concevoir correctement une architecture de contrôle d'un robot lui permettant d'apprendre de nouveaux comportements ?
Lorsque l'on s'intéresse à la robotique sociale, on peut dégager deux principaux objectifs : l'assistance et le divertissement. Même si les applications peuvent être très différentes (assistance aux personnes, assistance au combat, robot jouet), l'idée est de concevoir un robot intelligent capable d'évoluer dans l'environnement physique, et de s'adapter à l'environnement social des humains qui l'entourent. Nous pouvons alors dégager deux approches possibles pour le développement de tels robots. Une première consiste à pré-programmer ou "scripter" les comportements. Cette approche descendante de l'intelligence artificielle trouve de nombreuses applications aussi bien dans des jeux vidéos que pour certains robots industriels. Néanmoins, une telle approche sous entend que le modèle mécanique du robot est connu et que l'environnement ne change pas ou peut alors être modélisé sans erreur, car ces programmes sont peu adaptatif. Or, il est très difficile de parfaitement décrire un environnement dans sa globalité sans tenir compte de sa dynamicité. Généralement, les applications adoptant cette approche visent non seulement à développer le contrôleur du robot, mais également à formater son environnement pour assurer au mieux le bon fonctionnement du robot. La seconde approche, adoptée dans cette thèse, dite ascendante, vise à permettre à des robots à apprendre et s'adapter dans un environnement a priori inconnu. L'enjeu est fondamentalement différent, car il ne s'agit plus de pré-programmer les comportements du robots et de décrire l'environnement. Il s'agit de donner les capacités suffisantes au robot à s'adapter non seulement à l'environnement dans lequel il évolue, mais également à sa propre mécanique. C'est alors l'ensemble des ces capacités qui permet de faire émerger des comportements plus complexes. Une question qui se pose est : quels sont les mécanismes minimaux et génériques permettant à un robot d'apprendre des comportements dans un environnement inconnu ? Les animaux et l'homme ont la capacité d'apprendre de nouveaux comportements dans un environnement a priori inconnu (l'idée d'une decription de l'environnement "pré-cablée" dans le système nerveux paraît peu probable.). Les études menées aussi bien sur l'homme en psychologie que sur les animaux en éthologie, ainsi qu'en neurobiologie, apportent de nombreuses données sur l'organisation du système nerveux et leurs implications dans différents comportements observés. Dans ce cadre, les réseaux de neurones artificiels sont un outil idéal pour le développement d'un contrôleur de robot biologiquement inspiré. En effet, ces réseaux permettent de reproduire plus ou moins fidèlement le fonctionnement de différentes structures neuronales. Grâce à la plasticité des synapses, les réseaux de neurones peuvent permettre à un robot de s'adapter à un environnement inconnu et apprendre de nouveaux comportements.
Bien que les dispositifs et les outils actuels permettent le développement de contrôleurs de robots, il reste à résoudre la question de l'adaptation et de l'autonomie comportementale. Com-

ment s'affranchir des situations supervisées pour pouvoir interagir de manière naturelle ? Dans ce contexte un modèle développemental qui permet l'émergence de comportements d'imitation de bas niveau sur un robot *Aibo* est appliqué et testé dans le chapitre 2. Nous verrons ensuite sous quelle forme un comportement peut être mémorisé. Le chapitre 3 propose alors d'encoder un comportement comme une succession d'événements sensori-moteur, sous la forme d'une séquence temporelle complexe de gestes sur un robot *Aibo*. Il faut aussi lier ces séquences avec les états ou catégories appris dans l'environnement. Le chapitre 4 présente alors deux stratégies possibles permettant à un robot mobile équipé d'une caméra et d'une boussole électronique de naviguer. La première permet à un robot de se déplacer dans l'environnement en utilisant des informations spatiales. La seconde utilise des informations temporelles permettant à un robot d'apprendre la succession de ses déplacements. Le chapitre 5 présente un modèle dans lequel les stratégies spatiale et temporelle sont fusionnées et exécutées en parallèle. Une étude est alors réalisée sur la coopération et la compétition de ces stratégies afin de discuter les éléments importants d'un système de sélection de l'action. Enfin, le chapitre 6 présente les outils développés et utilisés pour concevoir et exéctuter en temps réel de très grandes architectures neuronales distribuées sur plusieurs unités de calculs. Ces outils ont permis d'intégrer mes travaux avec ceux du laboratoire LASA (Learning Algorithms and Systems) de l'EPFL (Ecole Polytechnique Fédérale de Lausanne) permettant ainsi à un robot mobile équipé d'un bras robotique d'apprendre des comportements combinant la navigation, le geste et l'objet.

Chapitre 2

Du développement sensori-moteur à l'imitation : une approche épigénétique

Dans le contexte de l'apprentissage de comportements sur un robot, la question qui se pose est : comment un robot peut-il apprendre un comportement ? Une première possibilité est de laisser le robot apprendre seul, essayant de découvrir la solution en explorant l'ensemble des associations possibles, de lier par exemple les lieux, les objets, les actions. Bien sur, cette méthode demande beaucoup de temp avant que la solution soit trouvée, le temps de l'exploration et de la découverte étant directement dépendant du nombre d'associations possibles, de la séquence à apprendre et des catégories à constituer(reconnaître un lieu, un objet, généraliser des actions) pour "agréger" les différents éléments qui constituent le comportement. Si une telle solution est acceptable dans le cadre d'environnements bien définis (simulations, lieux ou objets sont segmentés a priori, actions déjà codées sous forme de symboles), elle devient inapropriée dans un environnement réel et riche.

Que ce soit en robotique ou en psychologie, la capacité d'imitation est apparue très tôt comme l'élément qui permet d'accélérer l'apprentissage d'un agent grâce à l'observation de l'autre, cette observation réduisant fortement l'espace d'exploration pour découvrir et "assembler" les éléments du comportement [Bandura, 1971]. Dans ce cadre, l'imitation se révèle être alors un atout certain en terme de temps d'acquisition de nouveaux comportements.

L'imitation permet d'envisager l'autre comme un "outil social" qui permet la découverte rapide de propiétés intéressantes de l'environnement physique [Dautenhahn, 1995]. Dautenhahn illustre ce principe d'"outil social" par un robot dont la seule capacité de suivi (sans notion évoluée de ce qu'est l'autre, sans notion évoluée de "soi") amorce la découverte de régions de "récompense" dans un environnement vallonné parcouru par des robots mobiles. L'imitation permet par l'observation d'accéder à des comportements de plus haut niveau [Kuniyoshi, 1994a, Billard *et al.*, 1998, Hayes et Demiris, 1994]. L'imitation est enfin une interface homme machine intuitive et accessible par quiconque [Cheng et Kuniyoshi, 2000, Mataric, 2000, K. Dautenhahn, 2002].

Néanmoins, un des grands défis de l'imitation reste de comprendre les mécanismes mis en jeu qui permettent de lier "ce que fait l'autre" avec "ce que je peux faire". Cette problématique trouve le nom de problème de correspondance (*corresponding problem*) et a été évoqué par Nehaniv [Nehaniv et Dautenhahn, 2002].

Ce problème des correspondances dissimule pour moi un second problème en robotique autonome, celui du lien entre le contrôle moteur et l'apprentissage d'un comportement aussi simple soit il (par exemple une séquence de déplacements ou de gestes, même sans signification particulière). En d'autre terme, comment passer du mouvement à l'action (et vice versa), de la simple commande motrice au comportement ?

La littérature nous fourni nombre de modèles pour le contrôle moteur. J'évoquerai plus particulièrement dans ce chapitre ceux qui permettent d'effectuer le contrôle d'un bras de robot dans l'espace visuel. Beaucoup de travaux de recherche se concentrent sur l'apprentissage de comportement mais possédent en pré-requis une simplification des espaces de travail (exosquelette pour l'adaptation des données motrices, dispositif externe de vision pour le calcul du modèle inverse des commandes du robot, etc).

Existe-t-il une hiérarchie si forte entre un "étage" où serait encodé le comportement et un "étage" d'execution de la commande et quelle est la place des transformations nécessaires (espace visuel, espace de travail, espace de contrôle) ?

J'aborderai ainsi ce chapitre par ce qui est pour moi le révélateur de cette question fondamentale : la différence en psychologie et en robotique de deux niveaux d'imitation. Selon certains [Schaal, 1999, Heyes, 2001], le premier niveau concerne une imitation accessible de l'être humain seul, centré sur la définition de l'imitation véritable. Le second niveau regroupe un ensemble plus large de comportements partagés par l'homme (notamment le très jeune enfant) et certains animaux.

Ce bas niveau fait intervenir des mécanismes perception-action comme l'apprentissage sensori-moteur, la valence du stimulus, l'attention de l'observateur se porte sur l'objet manipulé par l'imité [Spence, 1937, Thorpe, 1963] ou sur le lieux dans lequel l'imité réalise l'action [Roberts, 1941]), l'émulation [Whiten et Ham, 1992,Byrne et Russon, 1998] qui concerne l'augmentation de la saillance de certains buts après l'observation d'un congénère atteignant ce but et la facilitation de la réponse [Bandura, 1971] où l'attention de l'observateur porte sur les actions réalisées par un congénère, alors la probabilité de faire les mêmes actions augmente.
Par la suite, j'emprunterai une démarche développementale en reprenant les principes de la proto-imitation, destinée à montrer comment un homéostat, muni de l'ambiguïté de la perception est capable de faire émerger un comportement d'imitation immédiate de gestes sans notion de soit, de l'autre ni d'informations a priori sur les transformations entre espace de travail, espace visuel et commande motrice. Enfin, je discuterai les limitations de cette architecture et notamment le codage visuo-moteur appris par le robot, pour proposer un principe de codage plus adapté à l'apprentissage de comportement. Un des principaux avantage de ce nouveau codage est d'être similaire pour le contrôle d'un bras de robot à plusieurs degrés de liberté ou pour l'utilisation d'un robot mobile (chapitre 4 et 5).

2.1 Apprentissage par imitation en robotique et en psychologie

L'imitation a reçu différentes définitions qui peuvent être distinguées selon différentes modalités spatio-temporelles. On parle alors de l'imitation différée qui implique des processus cognitifs complexes et de l'imitation immédiate comme un comportement plus bas niveau [Bandura, 1969].
L'étude de comportements d'imitation en robotique a souvent été réalisée à travers l'apprentissage par l'observation ou l'apprentissage par démonstration pour des solutions moins contraintes. En effet, ces approches dites *descendantes* se heurtent aux difficultés liées à la complexité du traitement de la vision afin de reconnaître l'action de l'autre et de la correspondance avec les actions motrices du robot.
Néanmoins, une hypothèse prend à contre-pied l'imitation différée et propose de d'abord travailler sur l'imitation immédiate à l'image de l'imitation spontannée des jeunes enfants comme un comportement émergent des principes développementaux simples. Dès lors, on peut se poser la question : sur la base d'une imitation simple ou proto-imitation (imitation spontannée de gestes), peut-on "prolonger" la séquence développementales et apprendre des tâches plus complexes ?

2.1.1 Imitation différée et apprentissage par l'observarion

L'imitation différée est définie par le fait qu'un agent imitant reproduit ce qu'il a observé dans un cadre spatio-temporel éloigné et sans la présence de l'imité. En 1945, Piaget étudie le développement de ses trois enfants en procédant par mise en situations de problèmes [Piaget, 1945]. Il décrit notamment le développement d'imitations chez les enfants âgés de 9 à 12 mois environ, qu'il considère comme le "*(...) processus assurant la transition entre l'intelligence sensorimotrice et la représentation imagée*". Piaget se concentre ainsi sur l'imitation différée : l'enfant reproduit une action en la comparant à l'image interne qu'il s'est construite en observant auparavant l'adulte. Avant le 2ème mois, Piaget n'observe que des actions sensorimotrices spontanées qu'il qualifie de *"préparations réflexes à l'imitation"*. Avant le 9ème mois, il parle d'actions motrices d'accommodation pure, sans considérer aucun de ces phénomènes comme des imitations. Dès lors, l'observation de l'imitation différée devient le témoin du passage de l'enfant a une activité

mentale plus élaborée. Ca serait à partir du 18ème mois que le jeune enfant atteindrait le stade de l'imitation représentative - dimension centrale du processus symbolique [Piaget, 1970].

De manière similaire en robotique, une grande partie des travaux se sont concentrés sur l'apprentissage par observation en utilisant essentiellement la vision et des algorithmes d'apprentissage non supervisé [Kuniyoshi, 1994b, Kuniyoshi, 1994a, Bakker et Kuniyoshi, 1996, Gaussier *et al.*, 1998]. Cette méthode d'apprentissage se déroule en trois temps. Dans un premier temps, le robot observe le démonstrateur réalisant la tâche. Durant cette phase, le robot enregistre les informations nécessaires qui lui permettront de restituer le comportement. Ensuite, le robot traite les données afin de les transposer à ses propres effecteurs. Une fois ce traitement réalisé, le robot restitue le comportement préalablement observé sans intervention extérieure. Une approche souvent adoptée de l'apprentissage par l'observation repose sur un ensemble de symboles [Pardowitz, 2007, Pardowitz *et al.*, 2007]. Ces symboles représentent un répertoire d'actions reconnues par le robot qui sont ensuite appariés avec les actions motrices (figure 2.1).

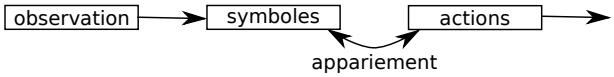


Fig. 2.1: Modèle d'apprentissage par imitation. A partir de l'observation, le robot reconnait des symboles correspondant à des actions. Ces symboles sont ensuite couplés avec des actions motrices à réaliser.

Néanmois, l'apprentissage par l'observation soulève un certain nombre de problèmes difficiles à résoudre comme la construction des symboles (*symbol grounding problem*) et la correspondance des informations observées sur l'imité afin de les transposer pour la mécanique du robot (*corresponding problem* [Nehaniv et Dautenhahn, 2002]). Cela implique souvent que le robot possède un modèle de sa mécanique et un modèle de l'humain qu'il observe. De plus, une telle méthode d'apprentissage implique l'utilisation de symboles réagissant à des actions particulières reconnues lors de l'observation. Des solutions moins contraintes ont été proposées utilisant des moyens techniques plus direct que la vision comme des exosquelettes ou des capteurs placés sur le corps de l'humain, mais aussi en manipulant directement le robot. On parle alors d'apprentissage par démonstration. Ces travaux permettent alors de se concentrer sur la généralisation et l'adaptation du mouvement de l'humain à celui du robot.

2.1.2 Apprentissage par démonstration

Reprenant le principe d'apprentissage par imitation, l'apprentissage par observation ne se limite plus à l'utilisation de la seule modalité visuelle pour acquérir des informations sur la tâche. Les informations relatives au comportement à apprendre peuvent être recueillies à travers un équipement plus direct comme un exosquelette [Ijspeert *et al.*, 2002b], des accéléromètres placés sur l'humain ou lors d'une interaction plus naturelle en manipulant directement le robot [Calinon, 2007]. Ces dispositifs permettent alors de s'abstraire des difficultés à traiter les informations provenant de la vision et de se concentrer sur les caractéristiques motrices qui constituent le comportement (que ce soit des informations sur ce que fait le démonstrateur, ou des informations sur ce que fait le robot). En réalisant plusieurs fois l'apprentissage d'une même tâche, le robot peut alors généraliser pour restituer un comportement plus robuste. Pour que le robot puisse correctement généraliser, il faut une certaine variance entre les différentes démonstrations. Cette méthode d'apprentissage implique donc l'intervention d'un expert qui connait la tâche et comment la réaliser [Münch *et al.*, 1994].

Dans [Schaal *et al.*, 2001,Ijspeert *et al.*, 2002a,Schaal *et al.*, 2007] les auteurs apprennent un geste à un robot humanoïde. Pour effectuer cet apprentissage, le démonstrateur est équipé de capteurs au niveau des articulations qui permettent de mesurer l'angle de chacune des articulations. Ce dispositif permet alors de faire la correspondance de dynamiques sensori-motrices de systèmes physiques différents (l'homme vs le robot). Lors de la phase d'apprentissage, l'expert réalise le mouvement qu'il souhaite faire apprendre au robot. Pendant qu'il réalise le mouvement, toutes les mesures d'angles au niveau des articulations sont enregistrées. A partir de ces données le robot apprend le geste puis le reproduit.
Dans les travaux de [Calinon et Billard, 2007], une expérience similaire a été réalisée sur un robot humanoïde HOAP3[1]. Dans ces travaux, les auteurs traitent également du problème de correspondance. En effet, lors de l'apprentissage, les données enregistrées le sont à partir du corps humain du démonstrateur, elles sont ensuite traitées pour que le robot reproduise le geste avec ses propres effecteurs. Par conséquent, le geste reproduit peut être plus ou moins différent de celui de la démonstration. Pour traiter ce problème, les auteurs ajoutent au robot la capacité à être corrigé en ligne par l'expert qui le manipule durant la reproduction. Le robot apprend plus rapidement la tâche à réaliser grâce à cette interaction avec l'humain. On peut alors parler d'apprentissage coopératif. Lors de cette phase de correction, l'expert manipule le robot qui est alors en mode passif, c'est-à-dire que le robot se laisse manipuler. Grâce à cette coopération, le robot devient plus performant dans la tâche qu'il doit réaliser.
Néanmoins, l'utilisation d'équipements plus direct que la vision ne permet pas de s'abstraire du problème de la correspondance entre ce que le robot perçoit et ses propres effecteurs. De plus, l'apprentissage se déroule hors ligne et nécessite donc que le robot ait toutes les informations relatives au geste avant de réaliser l'apprentissage et de pouvoir le reproduire.

2.1.3 Imitation immédiate

En 1986, J. Nadel définit, à partir de différents travaux, l'imitation comme un processus développemental [Nadel, 1986] ; avec un rôle d'acquisition de connaissances, l'enfant commence par imiter des actions simples et progresse vers des imitations d'actions de plus en plus complexes [Nadel et Potier, 2002], mais aussi avec une fonction de communication gestuelle chez les enfants pré-verbaux. Cette fonction est principalement défendue depuis 20 ans par J.Nadel [Nadel, 1986] et par [Andry, 2002a, Ito et Tani, 2004] en robotique. De manière précoce, l'enfant pré-verbal est capable d'interagir et de communiquer via, par exemple, des jeux d'imitation.
Dès la naissance, le nouveau né est capable d'imiter ou mimer certains mouvements faciaux [Zazzo, 1957,Meltzoff et Moore, 1977,Meltzoff et Decety, 2003]. Les auteurs, Meltzoff et Moore, spéculent que la principale fonction de cette imitation correspondrait à un "like me mechanism", mécanisme dont disposerait l'enfant à la naissance pour détecter ses semblables : "si je peux t'imiter et si tu peux m'imiter, alors nous sommes de la même espèce." De plus, l'imitation néo-natale intrigue, puisque le bébé, qui n'a jamais vu son visage, est capable d'associer des gestes de parties du corps qu'il ne voit pas, mais qu'il ressent, avec des gestes (ceux de l'autre) de parties du corps qu'il voit mais ne ressent pas. Ainsi, l'imitation néo-natale pose la question de la représentation et la formation des associations des différentes modalités (vision, propriocepion). Dans les mois suivants la naissance, les capacités d'imitation vont se complexifier de pair avec le développement sensori-moteur de l'enfant [Nadel et Potier, 2002].
Des travaux de Rizzolatti mettent en évidence la présence de neurones mirroirs dans le cortex du macaque [Rizzolatti *et al.*, 1996]. D'après ces travaux, les neurones miroirs désignent une

[1]Robot humanoïde Fujitsu

certaine catégorie de neurones du cerveau qui présentent une activité à la fois lorsqu'un individu exécute une action spécifique sur un objet et lorsqu'il observe un autre individu (en particulier de son espèce) exécuter cette même action, d'où le terme miroir. En neurosciences cognitives, ces neurones miroirs sont supposés jouer un rôle dans des capacités cognitives liées à la vie sociale comme la capacité à imiter ou l'empathie. Cependant, rien ne permet de dire si la réponse de ces neurones possède un caractère "inné", ou concerne le résultat d'un apprentissage sensorimoteur qui aurait lieu pendant les premiers mois de vie. Mais si les neurones mirroirs nécessitent un apprentissage préalable alors un problème se pose, car ils ne peuvent pas être à la fois la cause et la conséquence d'un même apprentissage. Enfin, il est à noter que les neurones miroirs ne concernent pas les gestes faciaux envoyés lors d'imitation néo-natale.
D'un point de vue développemental, l'imitation immédiate peut alors servir de point de départ pour permettre à un robot d'apprendre des comportements simples à partir desquels des comportements plus complexes pourront être appris. La question qui se pose alors est : comment un comportement d'imitation immédiate peut-il émerger d'un robot ?

2.1.4 Proto-imitation

Supposons un contrôleur capable de commander de manière cohérente un système oeil-bras, c'est à dire qu'il est capable d'associer la vision de l'extrêmité avec la propriocetion correspondant aux positions de son bras pour atteindre un quelconque point dans l'espace de travail. On fait également l'hypothèse que le robot ait un système visuel simple, pouvant par exemple seulement détecter les zones de mouvement. Le robot pourra suivre une main, l'extrêmité de son bras ou un objet qui se déplace dans son champ visuel, mais il ne sera pas capable (sans autres informations) de les différencier (pas de notion de soi ou de l'autre). Enfin, supposons que le contrôleur du robot est un homéostat qui tend à tout moment à équilibrer par l'action ses perceptions, c'est à dire la vision et la proprioception. Si un humain bouge sa main dans le champ de vision du robot, ce dernier détectera une zone en mouvement, comprise comme la position de l'extrêmité de son bras, qui induira une erreur avec la position de son bras (figure 2.2).

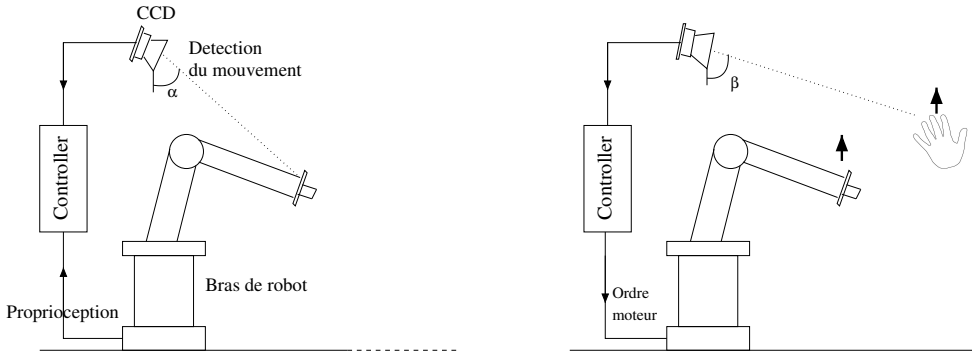


Fig. 2.2: Illustration d'un mécanisme simple de proto-imitation. A gauche : un contrôleur neuronal apprend les associations visuo-motrices de l'espace de travail. L'apprentissage se déroule pendant une phase d'exploration aléatoire dans l'espace sensori-moteur. A droite : une fois les associations sensori-motrices apprises, un simple décalage mécanique de la caméra induit une confusion entre la main agitée par un utilisateur et la position du bras robotique. Cette confusion génère une erreur que le contrôleur corrige en déplaçant le bras de robot vers la main agitée. Le système peut ainsi imiter la trajectoire de la main de l'utilisateur et un observateur dire que le robot "imite" les gestes de l'utilisateur.

Le robot tentera alors de minimiser cette erreur en déplaçant son bras vers la zone de mouvement. Si l'expérimentateur continue de bouger sa main, alors le robot réalisera les mêmes mouvements

et un observateur externe concluera à une imitation de gestes simples.
Ainsi, en jouant sur l'ambiguïté de la perception du robot, un comportement d'imitation émerge d'un simple homéostat cherchant à équilibrer par l'action ses informations perceptives. Ce principe à été utilisé avec succés pour permettre à un robot mobile d'imiter et apprendre la séquence de déplacements d'un professeur dans une salle [Gaussier *et al.*, 1997, Gaussier *et al.*, 1998, Moga et Gaussier, 1999, Moga, 2001]. Le mécanisme homéostatique est alors un contrôleur visant à constamment corriger l'erreur entre la vision et la position de son bras (figure 2.3).

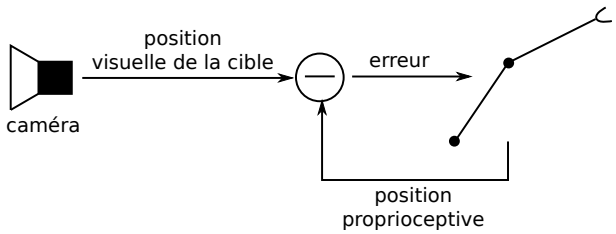


Fig. 2.3: Mécanisme d'homéostat visant à minimiser l'erreur entre les informations visuelles et les informations motrices du bras robotique.

Néanmoins, pour permettre l'émergence d'un comportement d'imitation, le robot doit au préalable apprendre à associer les positions visuelles avec les informations proprioceptives de son bras de manière à contrôler de manière cohérente son bras. Ce problème implique de s'intéresser à la construction des primitives motrices qui composent le répertoire d'actions et de leur adaptation aux changements. Dans ce cadre, il est alors indispensable de s'intéresser au niveau inférieur du contrôle moteur : comment apprend-on à contrôler son bras ?

2.2 Contrôle moteur

Comme nous venons de le voir, un préalable à la proto-imitation est que le robot soit capable de contrôler sont bras en fonction de stimulus visuels. Ce type de contrôleur fait alors référence aux travaux portant sur l'asservissement visuel. Le terme de "visual servoing" (asservissement visuel) a été introduit par [Hill et Park, 1979] pour différencier leur travaux avec les approches visant à séparer la prise d'une image et le contrôle du robot. Les tâches les plus fréquemment réalisées avec ce type de mécanisme est le suivi d'une cible particulière, le suivi d'une trajectoire particulière et la prise d'un objet particulier. Dans tous les cas, l'objectif est de minimiser l'erreur entre la position du robot manipulateur et la position désirée. Il existe un très grand nombre de travaux portant sur ce sujet ainsi qu'énormément d'articles faisant la synthèse des différents travaux existant sur l'asservissement visuel [Corke, 1994, Hutchinson *et al.*, 1996, Kragic et Christensen, 2002, Chaumette et Hutchinson, 2006, Chaumette et Hutchinson, 2007]. Ces travaux mettent en évidence les difficultés liées à la correspondance entre les espace visuel et proprioceptif et du contrôle du bras robotique.

2.2.1 Correspondance des informations visuelles et motrices

Pour facilement réaliser le calcul de l'erreur entre ses perceptions, il est alors indispensable que les informations perceptives soient dans le même espace. Il est donc nécessaire que le robot soit capable de localiser la position visuelle de son extrêmité dans l'espace moteur (figure 2.4) ou de localiser son extrêmité dans l'espace visuel (figure 2.5).

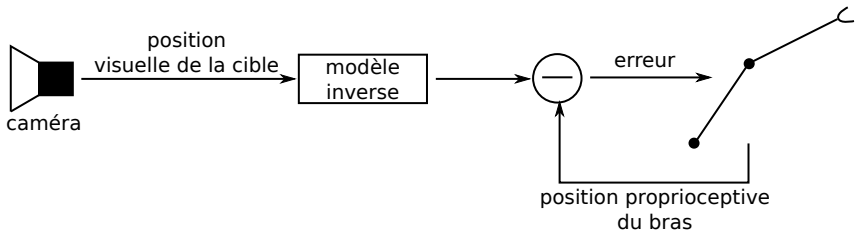


Fig. 2.4: Contrôleur réalisant le calcul de l'erreur entre la position visuelle de la cible et la position du bras dans l'espace moteur. La position de la cible est transformée dans l'espace moteur avec l'utilisation d'un modèle inverse

Pour réaliser une transformation de l'espace visuel vers l'espace moteur du robot, une méthode fréquemment utilisée est le modèle inverse. Cette méthode impose alors d'avoir a priori un modèle de la mécanique du bras, de définir une matrice de transformation de la position visuelle vers l'espace moteur du bras (généralement une matrice Jacobienne du bras robotique). Même si cette méthode donne une réponse, il n'y a pas de gestion des ambiguïtés ; c'est à dire lorsque plusieurs positions angulaires du bras sont possibles pour une même position de l'espace visuel. Pour palier ce problème, il est alors nécessaire d'avoir une ou plusieurs caméras extérieures.

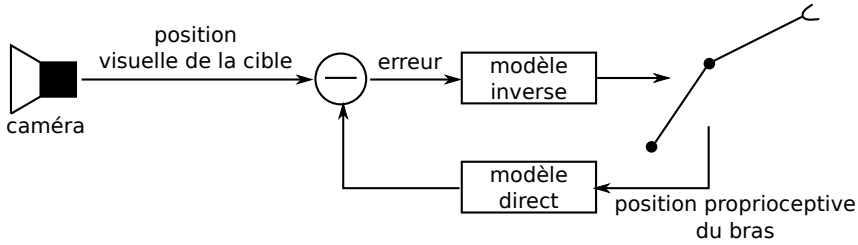


Fig. 2.5: Contrôleur réalisant le calcul de l'erreur entre la position visuelle de la cible et la position du bras dans l'espace visuel. La position du bras est transformée dans l'espace visuel avec l'utilisation d'un modèle direct. L'erreur à appliquer sur les moteurs est alors transformée dans l'espace moteur.

Pour réaliser la transformation de l'espace moteur vers l'espace visuel, il faut déterminer la position de tous les points du bras à partir des informations motrices. Le modèle direct est alors généralement utilisé, mais cette méthode implique également de connaître a priori le modèle mécanique du bras. De plus, après que l'erreur ait été calculée dans l'espace visuel, les commandes motrices résultantes doivent généralement être retransformées vers l'espace moteur. Néanmoins, cette méthode peut se révéler plus simple pour le calcul de l'erreur, car dans ce cas il est réalisé dans l'espace visuel à deux ou trois dimensions, alors que dans l'espace moteur il serait fait dans un espace à six dimensions pour un bras robotique à six degrés de liberté (en supposant qu'il y ait une dimension par degré de liberté).
Dans le cadre d'apprentissage par imitation sur un robot autonome, je ne souhaite pas faire un tel a priori, car le robot doit pouvoir s'adapter à sa mécanique ou, d'une autre manière, le modèle doit pouvoir être le même quelque soit la mécanique du bras utilisé.
Dans [Albus, 1975], l'auteur présente un modèle baptisé *Cerebellar Model Articulation Controller* (*CMAC*) permettant le contrôle d'un bras robotique de manière adaptative. Le modèle est composé d'un vecteur d'entrée pour chaque articulation à contrôler dont les composantes sont les informations sensorielles, motrices, ainsi que l'identifiant de la commande désirée. Ce vecteur représente alors une adresse mémoire dans laquelle sont stockées les poids synaptiques. En sortie, un neurone fait la somme des poids pour délivrer la commande motrice au moteur. Il y a un neurone de sortie par articulation. L'adaptation se fait par modification des poids synaptiques

stockés en mémoire à partir de l'erreur angulaire entre la position effective de l'articulation et la position désirée. Ici, le modèle n'a pas besoin de modèle de la mécanique du bras à priori, mais seulement le nombre d'articulations à contrôler. Néanmoins, l'apprentissage nécessite une supervision fournissant l'information angulaire attendue pour calculer l'erreur faite par le robot et adapter les poids du réseau. La supervision est alors incompatible avec l'objectif de permettre à un robot d'apprendre par lui même sa propre coordination visuo-motrice.
Dans [Kuperstein, 1991], l'auteur présente un modèle neuronal, baptisé *INFANT*, qui permet à un robot d'apprendre sa coordination visuo-motrice. Le modèle est composé d'une carte neuronale de l'espace visuel sur laquelle est projetée la position visuelle de la cible à atteindre. Dans le cas d'un système visuel motorisé, la direction du centre de la vision est projetée sur une carte neuronale motrice. En fusionnant les informations des ces différentes cartes, il en résulte une carte "cible". Les informations motrices du bras robotique à contrôler sont ensuite associées aux unités (neurones) actives de la carte "cible". De cette manière, lors de l'utilisation d'une telle carte, une position courante dans l'espace visuel est déterminée à partir des informations motrices. Le robot utilisé est composé d'un montage de stéréo vision avec trois degrés de liberté. Le robot apprend l'erreur entre la position du centre de la vision et la position de la cible. Le modèle n'a que très peu d'information pour réaliser cet apprentissage : la vision, le nombre de moteur et leurs débattement. Le modèle *INFANT* a l'avantage de ne pas avoir besoin du modèle mécanique du bras a priori. De plus, l'apprentissage est complètement autonome (apprentissage purement associatif). Il reste néanmoins nécessaire que le robot ait un contrôleur lui permettant de calculer l'erreur entre ce qu'il perçoit visuellement et la position de l'extrêmité de son bras dans l'espace visuel et appliquer cette erreur sur ces moteurs.

2.2.2 Contrôle d'un bras robotique

Dans [Bullock et Grossberg, 1989], les auteurs présentent un modèle baptisé *Vector Integration To Endpoint* (*VITE*) permettant le contrôle d'un bras robotique (figure 2.6.A).

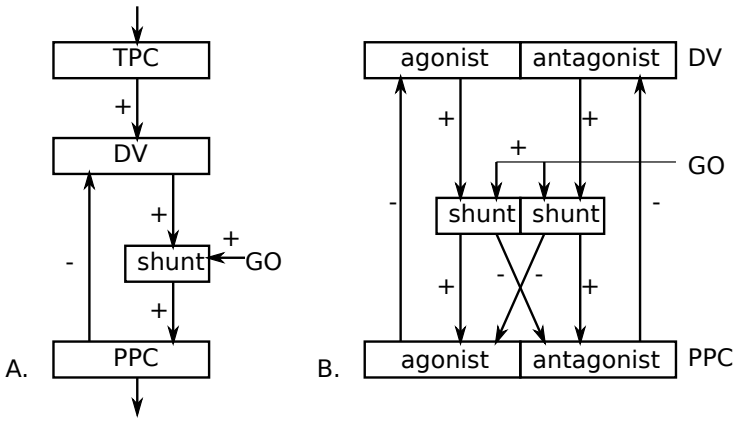


Fig. 2.6: A. Modèle VITE. *TPC* est un vecteur sur lequel est projeté la cible. *PPC* est un vecteur sur lequel est projeté la position courante de l'articulation. *DV* est un vecteur différentiel entre *TPC* et *PPC*. *GO* est un signal qui permet de moduler la vitesse de l'articulation. B. Modèle de deux circuit VITE permettant un contrôle semblable au contrôle musculaire.

Ce modèle se compose de deux vecteurs dans lesquels chaque composante représente une position d'une articulation et d'un vecteur différentiel. Sur le premier vecteur est projetée la position cible (*TPC*) à atteindre, sur le second est projetée la position courante de l'articulation (*PPC*). Le vecteur différentiel (*DV*) est la différence entre le vecteur *TPC* et *PPC* à chaque instant. Cette

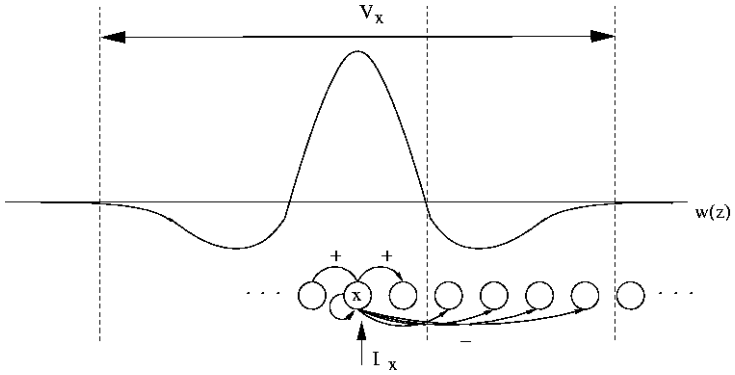


Fig. 2.7: Représentation neuronale du modèle d'Amari. Le champ neuronal se présente comme une carte 1D de neurone ayant des connexions récurrentes et des connexions locales. Les connexions locales du neurone x sur ses voisins sont ici représentées par la Différence de Gaussiennes (DOG) w.

différence est multipliée par un signal *GO* qui permet de moduler la vitesse du mouvement de l'articulation sans en perturber la forme. Une fois cette opération réalisée, le résultat est alors intégré dans le vecteur *PPC* qui ensuite génère la commande motrice à appliquer. Dans [Hersch et Billard, 2006], les auteurs utilisent un couple de contrôleurs *VITE* en parallèle pour réaliser un contrôle plus robuste et stable d'un bras robotique. Un premier contrôleur permet alors de calculer la commande dans l'espace angulaire du bras, alors que le second permet de calculer la commande dans l'espace cartésien. Ces deux commandes sont ensuite soumises à un mécanisme de contrainte de cohérence. Ce mécanisme permet d'assurer que la configuration angulaire désirée du bras fournie par le premier contrôleur *VITE* corresponde effectivement à la position désirée fournie par le second contrôleur *VITE*.
Néanmoins, le modèle *VITE* ne tient pas compte des informations proprioceptives, mais se contente d'intégrer la nouvelle position courante dans le calcul de l'erreur (dans le vecteur DV). Par conséquent, le contrôle du bras peut ne plus correspondre au mouvement désiré.

Les propriétés dynamiques du champ neuronal en font un outil idéal pour le contrôle moteur d'un robot autonome. La propriété de mémoire ainsi que les capacités de filtrage associées permettent l'utilisation d'informations relativement bruitées en entrée ou incomplètes dans le temps. C'est précisément cette propriété qui permet de conserver des comportements robustes à des entrées bruitées. Cet effet mémoire permet de garder l'activité sur une certaine durée même s'il n'y a pas d'informations en entrée.
Le calcul de l'activité de chaque neurone x se fait selon l'équation d'Amari [Amari, 1977] :

$$\tau \cdot \frac{f(x,t)}{dt} = -f(x,t) + I(x,t) + h \int_{z \in V_x} w(z) \cdot g\left(f(x-z,t)\right) dz \tag{2.1}$$

La fonction $f(x,t)$ représente l'activité du neurone x à l'instant t. $I(x,t)$ représente les entrées du système, c'est à dire la stimulation qui peut être appliquée en différents points du champ. La constante de temps τ représente le taux de relaxation du système. V_x est le voisinage du neurone x dans lequel les interactions excitatrices et inhibitrices modélisées par la fonction w (Différence de Gaussiennes (DOG) représentée figure 2.7) sont prises en compte. Ces interactions latérales modélisent les connexions fortes locales qui définissent une topologie sur le champ neuronal. Chaque neurone x possède des voisins z par lesquels il est susceptible d'être coactivé de façon excitatrice - créant ainsi une zone d'attraction - ou inhibitrice - créant une zone de répulsion -

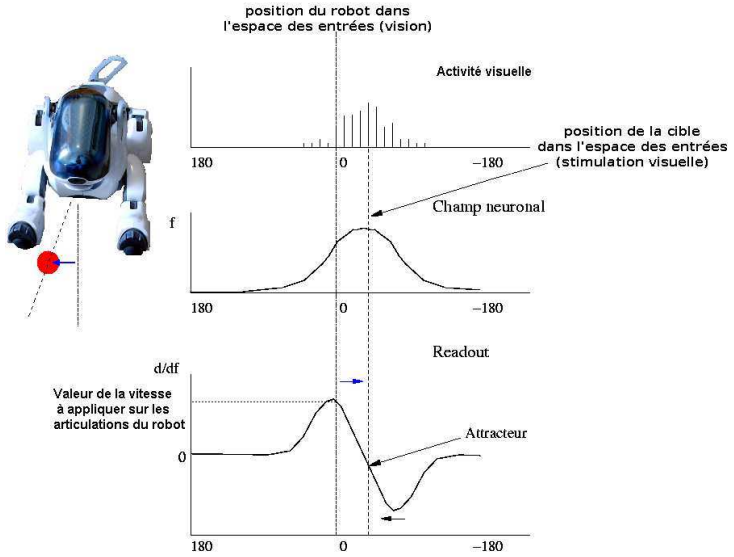


Fig. 2.8: Illustration du mécanisme d'extraction en présence d'une stimulation (en rouge). L'entrée visuelle stimule le champ neuronal et engendre un attracteur dont la dérivée par rapport à l'orientation est utilisée pour calculer la commande en vitesse du robot en direction de l'attracteur.

(selon w) en fonction de leur activité ($g()$ étant la fonction de transfert non linéaire -généralement une fonction continue par morceaux à seuil- de chaque neurone), et de leur distance.
Pour être correctement utilisées, les informations en sortie du champ neuronal sont traitées par un mécanisme d'extraction (readout, figure 2.8). Ce mécanisme permet d'exploiter les attracteurs et répulseurs formés par le champ neuronal pour le contrôle moteur. Le champ neuronal conserve la topologie des entrées. Le calcul de la dérivée spatiale de l'activité du champ neuronal exprime les valeurs des vitesses à appliquer aux articulations du robot pour aller vers la cible ou au contraire s'en éloigner. Cette dérivée est exprimée avec la même topologie que celles des entrées du champ neuronal. Les vitesses sont ensuite directement envoyées sur les moteurs du robot.
Ce mécanisme permet à chaque moteur placé dans l'"axe" des entrées de contribuer au déplacement de l'effecteur (tête, bras) vers la cible (activité de la stimulation visuelle). Néanmoins, si l'extraction de la commande est aisée (même valeur de vitesse envoyée à tous les moteurs d'un même axe), il est nécessaire d'être capable de connaître la position de l'effecteur à contrôler (orientation du robot, position de l'extrêmité) dans l'espace de calcul du champ de neurones (dans cet exemple l'espace visuel). Le parti pris est d'effectuer les calculs dans l'espace des entrées pour obtenir une commande indépendante du nombre de degrés de liberté de l'effecteur.

2.3 De l'apprentissage visuo-moteur à l'imitation bas niveau

Le robot doit se construire une représentation interne associant la vision d'un point de l'espace en trois dimensions et la position de l'extrêmité de son bras. Le système sur lequel porte cette étude est constitué des éléments suivant :
– La caméra d'un robot Aibo[2] (figure 2.9) ainsi que le cou du robot à trois degrés de liberté dont deux redondant, en configuration *Pan-Tilt*, c'est à dire un moteur *Pan* permettant un déplacement horizontal et deux moteurs *Tilt* permettant un déplacement vertical de la tête.

[2]Robot chien de Sony

– La patte de ce même robot possède trois degrés de liberté.

Fig. 2.9: Robot Aibo de Sony.

Pour se construire une représentation interne, le robot doit apprendre sa coordination visuo-motrice. Ce type d'apprentissage permet d'effectuer un changement d'espace en passant de l'espace proprioceptif (informations motrices) à un espace visuel. De cette manière, le robot est capable de pouvoir situer l'extrémité de sa patte dans son espace visuel uniquement à partir de ses informations motrices [Andry, 2002b]. Le résultat de ce mécanisme peut être assimilé à l'apprentissage du modèle direct par le robot mais également au modèle inverse par l'utilisation de champs neuronaux et de mécanismes d'extractions de la commande motrice.
L'espace moteur du bras étant plus important que l'espace visuel, le robot doit être également capable de suivre grâce aux moteurs de la tête du robot, l'extrémité de son bras de manière à l'avoir constamment dans son champ visuel.

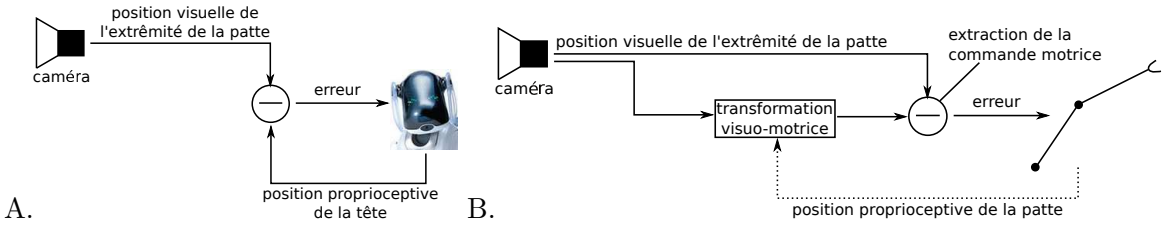


Fig. 2.10: Les deux homéostats composant l'architecture. A. Les informations motrices de la tête sont exprimées dans le même système de coordonnées que la vision, leur comparaison est possible directement. B. Les informations motrices du bras sont exprimées dans un système de coordonnées différent de la vision. Un apprentissage des associations visuo-motrices est alors nécessaire pour réaliser la transformation qui permet la régulation des informations de l'homéostat.

L'architecture intègre alors deux types d'homéostats représentés figure 2.10. Le premier est nécessaire au contrôle de la tête (suivi de l'extrêmité de la patte) et le second pour la coordination visuo-motrice).

2.3.1 La coordination sensori-motrice

L'apprentissage de la coordination sensori-motrice consiste à permettre au robot d'associer des stimuli avec des informations motrices. De cette manière, lorsque le robot utilise ce mécanisme,

il peut déterminer à partir des informations motrices où se trouve le stimulus dans l'espace de ces senseurs. Dans ce travail, le robot traite des stimuli visuels qui sont ensuite associés aux informations motrices du bras. Le modèle se compose d'une carte visuelle en deux dimensions (macro carte) sur laquelle est projetée la position visuelle de l'extrémité de la patte et pour chaque position visuelle, une carte proprioceptive en une dimension (micro carte) catégorisant les informations proprioceptives de la patte du robot. (figure 2.11).

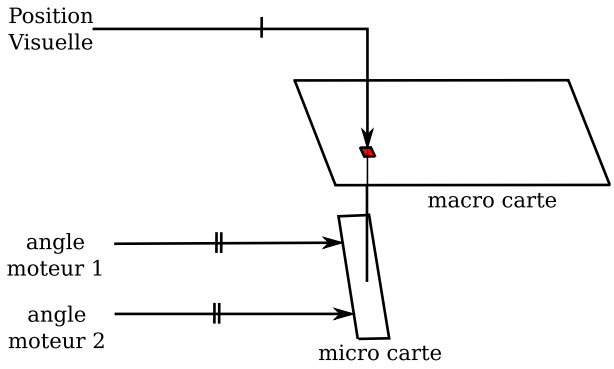


Fig. 2.11: Modèle d'apprentissage de coordination visuo-motrice

Le système étant équipé d'une seule caméra, il ne peut donc pas traiter d'informations en trois dimensions. Par contre, n'ayant aucune contrainte sur les mouvements du bras (hors contraintes physiques), l'extrémité peut se trouver n'importe où dans l'espace. Par conséquent le système doit être capable de catégoriser plusieurs postures du bras pour une même position visuelle de l'extrémité. Cette catégorisation est alors réalisée sous forme de carte auto organisatrice sur les micro cartes (figure 2.12).

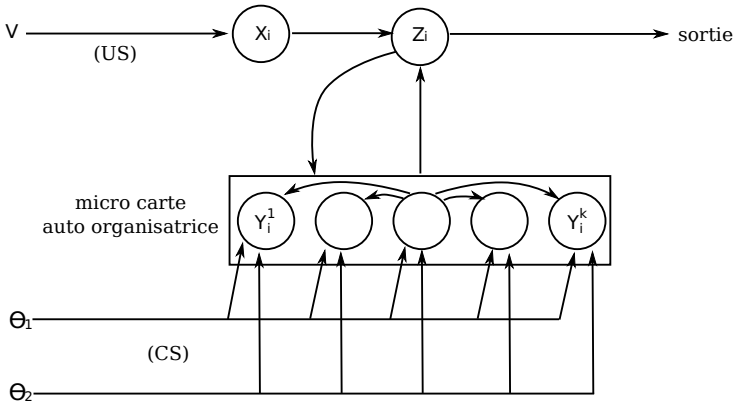


Fig. 2.12: Modèle d'une "colonne". Lors de l'apprentissage, la colonne reçoit en entrée un stimulus visuel (V) sur un lien inconditionnel (US). Ce stimulus permet alors l'apprentissage des informations motrices (θ_1, θ_2) par modification des poids sur les liens conditionnels (CS). Les neurones (Y_i^1, ..., Y_i^k) de la micro carte correspondant à cette colonne catégorise alors la posture du bras.

L'activité des neurones Y est proportionnelle au calcul de distance entre le vecteur courant des informations motrices θ et les valeurs des poids associés W :

$$Y_i^k = \frac{1}{1 + |\theta_j - W_{jk}|} \tag{2.2}$$

avec Y_i^k le k^{ime} neurone de la micro carte associé à la i^{ime} colonne. Chaque micro carte calcul un gagnant :

$$gagnant_i = argmax_{k \in n}(Y_i^k) \tag{2.3}$$

L'apprentissage se fait en fonction de la topologie des connexions locales des micro cartes.

$$W_{jk} = W_{jk} + \varepsilon.Y_i^k.\delta(d(gagnant_i, k), \theta_n, N_n).Z_i \tag{2.4}$$

La fonction d calcule la distance entre le k^{ime} micro neurone et le micro neurone gagnant de la micro carte. La fonction δ est une différence de gaussiennes qui modélise les interactions locales entre les micro neurones. ε est une constante d'apprentissage qui détermine la force de mise à jour des poids. Le neurone X_i déclenche l'apprentissage selon la règle :

$$X_i = \{^{si\ V_j.W_{ji} > \theta}_{0\ sinon} \tag{2.5}$$

Lorsqu'aucune entrée visuelle n'est active en entrée de la carte sensori-motrice, seule les informations motrices peuvent engendrer une réponse sur la colonne associée. Elles entraînent alors une activité sur la micro carte associée, dont le maximum est propagé sur le macro-neurone Z_i.

$$Z_i' = argmax(X_i, gagnant_i) \tag{2.6}$$

Chaque neurones Z_i est ensuite mit en compétition avec ceux de la carte sensori-motrice :

$$Z_i = \{^{1\ si\ Z_i' = max_{j \in n}(Z_j')}_{0\ sinon} \tag{2.7}$$

Le neurone Z_i représente donc la réponse de la carte sensori-motrice exprimée dans l'espace visuel. Cet apprentissage nécessite donc que le robot soit capable de détecter dans son champ de vision local l'extrêmité de sa patte.

2.3.2 Le traitement visuel

Le traitement visuel a pour objectif d'extraire de la vision les informations suffisantes pour permettre au robot de localiser l'extrémité de son effecteur.

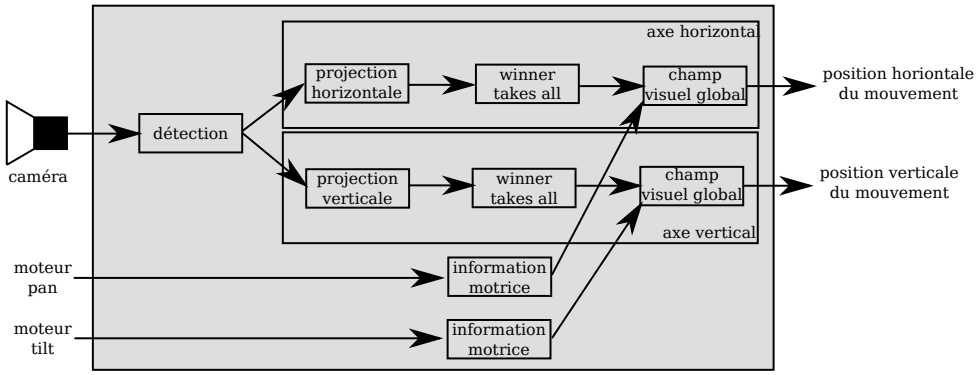


Fig. 2.13: Modèle de la détection de la position visuelle de l'extrémité de la patte du robot.

Comme le montre le figure 2.13, les images capturées par une caméra, sont traitées par un groupe de neurones qui extrait les informations visuelles. Ces informations sont alors projetées horizontalement et verticalement sur deux vecteurs (figure 2.14) dont la taille correspond au

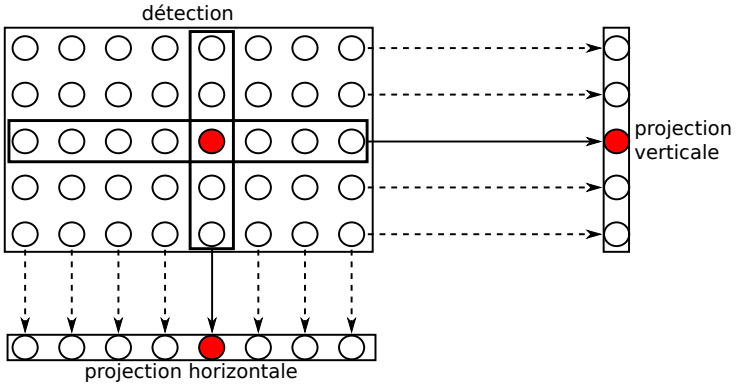


Fig. 2.14: Schéma des connexions entre le groupe de détection (mouvement ou de couleur selon l'expérience) et les groupes de projections horizontales et verticales. Les poids des connexions ont été choisis de manière à toujours rester dans l'intervalle [0,1] pour une activité en entrée bornée.

champ visuel local du robot ; c'est à dire à une capture. Pour chaque vecteur seule l'activité maximale est récupérée suivant l'équation :

$$gagnant_i = \{^{potentiel_i \ si \ potentiel_i > Seuil}_{0 \ sinon} \tag{2.8}$$

avec $potentiel_i$ l'activité en entrée du $i$ème neurone du groupe. Ensuite, les neurones gagnants sont mis en compétition pour choisir les N neurones les plus activés.
Ces activités maximales sur chacune des deux dimensions définissent pour le robot la position de l'extrémité de son effecteur. Si nous prenons l'exemple de mouvements humains du bras, la zone où le mouvement est maximum est justement la main, c'est à dire l'extrêmité du membre. Le mouvement de la main bénéficie de la sommation des vitesses de mouvements du poignet, du coude, de l'épaule, voire du corps. De plus, lorsque le robot bouge la tête, alors il perçoit son propre mouvement. Il est nécessaire de ne pas traiter ce mouvement, car il ne correspond pas à celui d'un geste. J'ai utilisé la solution trés simple : quand le robot perçoit son propre mouvement, l'activité est présente sur la majorité de l'image. J'ai alors ajouté deux groupes (un pour chaque axe) dont le rôle est d'inhiber l'activité lorsqu'elle est trop importante. De cette manière, il n'y aura pas d'autre traitement sur cette activité indésirable. En d'autres termes, seul un mouvement localisé et inférieur à une intensité donnée est détectée par le capteur de vision (mouvement de l'extrêmité de sa patte ou du bras d'un autre). Les activités gagnantes sont alors repositionnées sur deux vecteurs qui représentent non plus le champ visuel local, mais le champ visuel global du robot ; c'est à dire partout où le robot peut voir (figure 2.15).
Ce repositionnement est réalisé à partir des informations motrices, du débattement des moteurs et des angles d'ouvertures horizontales et verticales de la caméra. Les informations du moteur *Pan* sont donc utilisées pour repositionner horizontalement la position et les informations motrices du moteur *Tilt* pour le repositionnement vertical. Ce traitement permet alors de déterminer où se trouve le centre de la vision locale du robot. Ensuite, les activités des neurones gagnants sur les vecteurs du champ visuel local sont reportées sur les vecteurs du champ visuel global.
Finalement, les informations en sortie de ce système permettent de localiser une activité particulière dans l'espace visuel global du robot. Ces informations vont stimuler des champs de neurones dynamiques pour permettre au robot d'amener son effecteur vers l'activité perçue.

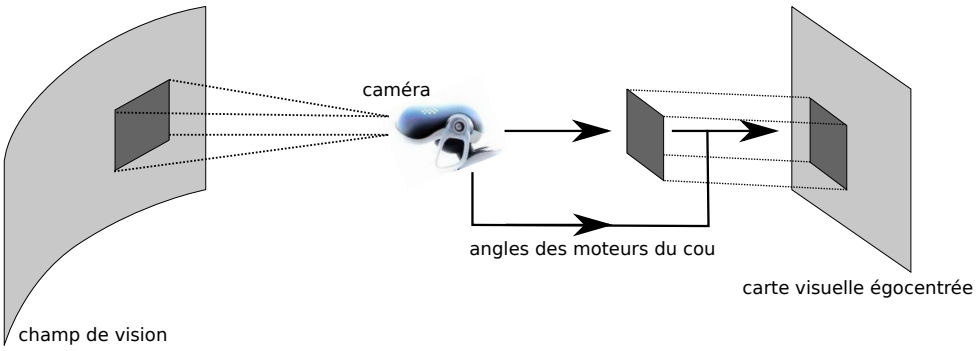


Fig. 2.15: Espace visuel du robot. Il s'agit d'une carte en deux dimensions représentant le champ visuel global accessible à la caméra du robot montée sur les moteurs du cou.

2.3.3 Dynamique du contrôle moteur

Pour réaliser le contrôle de la tête en fonction des informations visuelles, le contrôleur tire parti des propriétés de filtrage et de mémoire des champs de neurones dynamiques.

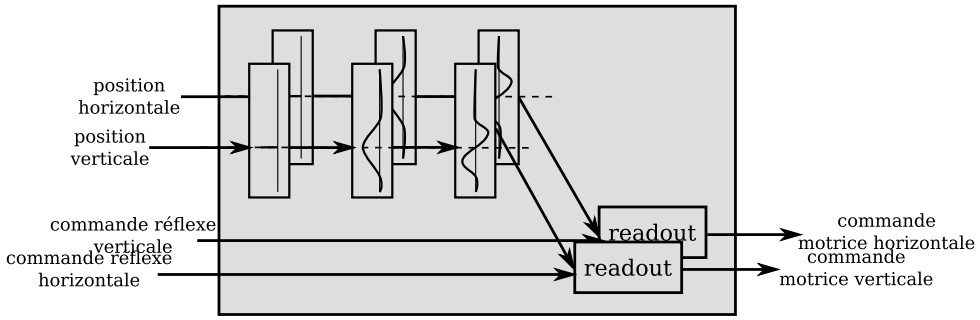


Fig. 2.16: Model d'extraction de la commande motrice de la tête du robot permettant le suivi de l'extrémité de sa patte.

Dans mes travaux, j'utilise deux champs de neurones dynamiques sous forme de deux cartes de neurones 1D correspondant à l'axe vertical et horizontal (axes des informations visuelles projetées comme vu dans la section précédente)(figure 2.16). Ces cartes sont stimulées par des cartes de même dimension (connexions de 1 vers 1) dont chaque neurone représente l'intensité de la stimulation.
L'utilisation du mécanisme "readout" permet d'extraire la commande motrice en vitesse à appliquer sur les moteurs en fonction de la position désirée et de la position courante de l'effecteur. Lors de l'apprentissage de la coordination visuo-motrice, l'effecteur contrôlé est la tête du robot pour qu'il puisse garder sa patte au centre de la vision.
La position des moteurs à déplacer (tête dans notre configuration) étant connu dans l'espace d'entrée, alors la valeur de la dérivée à cette position correspond à la vitesse à appliquer sur les moteurs pour se rapprocher de l'attracteur (mécanisme d'extraction vu figure 2.8). En appliquant cette commande, la position du de la tête change, et donc une nouvelle vitesse sera extraite de la dérivée. De cette manière, la commande motrice va permettre au robot de se rapprocher progressivement de l'attracteur en alignant sa position sur celle de la cible. Le signe de la dérivée défini le sens dans lequel il faut que la tête du robot aille pour se rapprocher de l'attracteur. De cette manière, si la tête va trop loin, il repartira dans l'autre sens pour toujours se rapprocher de la cible.

Ce mécanisme de suivi visuel constitue un comportement réflexe servant de base à l'apprentissage de coordinations visuo-motrices.

2.3.4 Tests de la coordination visuo-motrice

En raison de la taille trop importante de la patte du robot dans son champ visuel local, la détection du mouvement comme information visuelle n'a pu être utilisée. Dans les expériences précédentes de l'équipe, la détection de mouvement convenait bien avec un bras robotique *Katana*[3], car le bras était relativement éloigné de la caméra. Avec un robot de type *Aibo*, sa patte peut prendre un tiers du champ de vision, la détection du mouvement est alors difficile à utiliser. En conséquence, plutot que de détecter des zones de mouvement, le robot détecte des zones de couleur rouge. Des pastilles rouges ont alors été placées sur l'extrémité de la patte de manière à ce que le robot soit capable de la détecter visuellement Par conséquent, le traitement bas niveau de la vision consiste ici à extraire la couleur rouge des images capturées sur un groupe de neurone. De la même manière que l'hypothèse faite était que le mouvement extrait de sa vision correspondait à sa patte, ici la même hypothèse est faite avec la couleur rouge extraite qui correspond donc à l'extrêmité de sa patte.
Dans ces tests, l'image capturée a une taille de 208 pixels de largeur et de 160 pixels de hauteur. Elle est ensuite échantillonnée selon le nombre de neurones présents dans le groupe. Ici, le groupe est composé d'une matrice de 28 neurones de largeur et 22 neurones de hauteur. Par conséquent, un neurone correspond non pas à un pixel mais à une zone de l'image.

Test avec 3 degrés de liberté pour la patte et la tête

Dans un premier temps, trois degrés de liberté ont été utilisé pour la tête, ainsi que pour la patte (figure 2.17). La tête utilise uniquement la partie horizontale de gauche (de 0° à 90°), car la patte gauche ne pourra être visible uniquement dans cet espace visuel. Ceci permet de diminuer la taille du champ visuel global, et donc de gagner en temps de mise à jour et d'apprentissage de la carte visuo-motrice[4].

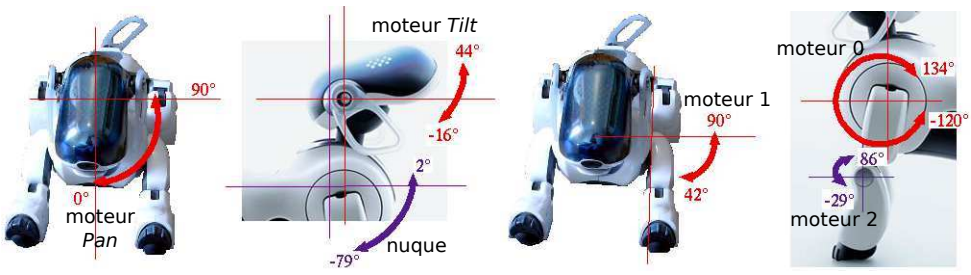


Fig. 2.17: A gauche : Illustration des articulations utilisées avec les valeurs minimum et maximum en degrés. A gauche est représenté l'espace horizontal (le "Pan"). A droite est représenté l'espace vertical couvert par la tête (le "tilt") et la nuque. A droite : Illustration des articulations utilisées avec les valeurs minimum et maximum de chacune. A gauche, le moteur 1 de l'épaule représente l'espace couvert horizontalement par la patte du robot. A droite est représenté l'espace couvert verticalement au niveau du moteur 0 "l'épaule" et du "coude" du robot.

[3]Bras robotique de Neuronics

[4]Le champ visuel global vertical a un maximum équivalent à la somme des deux articulations permettant un mouvement vertical ; c'est à dire $44 + 2 = 46^\circ$ maximum. Avec un champ de vision de 28° horizontalement et de 22° verticalement, on obtient donc un champ de vision global de $90^\circ + 28^\circ = 118^\circ$ de largeur par $(44^\circ - (-16^\circ)) + (2^\circ - (-79^\circ)) + 22^\circ = 163^\circ$ de hauteur.

Le débattement des articulations de la patte du robot (figure 2.17) ont été volontairement réduits, pour éviter les problèmes de casse mécanique. En effet, dans certaines positions le robot peut venir à toucher sa tête avec sa patte et forcer pour essayer d'aller plus loin. Pour cet apprentissage, la patte va être déplacée par un programme extérieur à l'application. Ce programme a pour but d'amener la patte du robot à différentes positions couvrant un maximum d'espace dans le champ visuel global du robot. Pour obtenir cette couverture de l'espace, chacun des degrés de liberté de la patte se déplacera d'un dixième de son amplitude. Par exemple, pour l'articulation générant un mouvement horizontal :

$$\frac{pas = (angle_max - angle_min)}{10} \tag{2.9}$$

En faisant ce calcul pour les 3 articulations de la patte, on obtient ainsi : $10 * 10 * 10 = 1000$ positions par lesquelles la patte se déplacera. A chaque position, la patte s'arrête 5 secondes le temps que l'apprentissage se fasse correctement. Le robot va donc apprendre les informations motrices de la patte pour chacune des positions où elle se trouve ; pas nécessairement l'une des 1000 positions, mais également des positions intermédiaires.

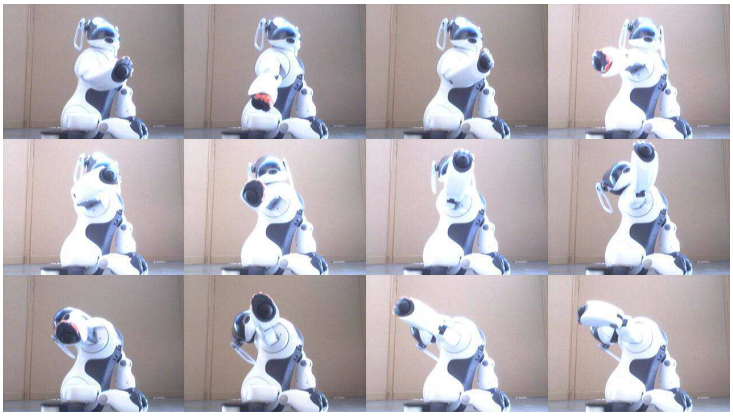


Fig. 2.18: Photos du robot en cours d'apprentissage sensori-moteur. La patte est guidée par un programme extérieur bougeant la patte sur 1000 positions dans l'espace global. Notre architecture permet au robot de garder sa patte dans son champ visuel afin d'associer la position visuelle de sa patte avec ses informations proprioceptives.

Pour cet apprentissage, les micro cartes associées à chaque position de la carte sensori-motrice du macro-colonne, sont composées de 6 micro neurones. Le modèle va donc faire l'apprentissage sur $118 * 163 * 3 * 6 = 346212$ liens.
Une fois l'apprentissage terminé, les tests consistent à placer le centre de vision du robot à certaines positions, de manière à ce qu'il positionne sa patte dans son champ de vision. Aprés plusieurs observations, j'ai pu constater que le robot ne répondait pas correctement ; il n'amenait pas sa patte dans son champ de vision, mais il la bougeait sur des positions éloignées. Il est également arrivé que le robot se retrouve dans une situation de blocage comme le montre la figure 2.19. Ces comportements résultaient de l'utilisation de 3 degrés de liberté pour la tête et la patte du robot.
La première cause vient directement de la résolution de l'espace global. Durant l'apprentissage, le robot n'a pas appris sur toutes les positions où sa patte est passée. Ceci est dû à la faible résolution angulaire des positions dans l'espace global, alors que le robot n'est pas aussi précis dans les valeurs appliquée aux articulations (de 2° à 3° d'erreur sur les angles ordonnés). Le très grand nombre de positions possibles dans l'espace global implique aussi un temps conséquent de

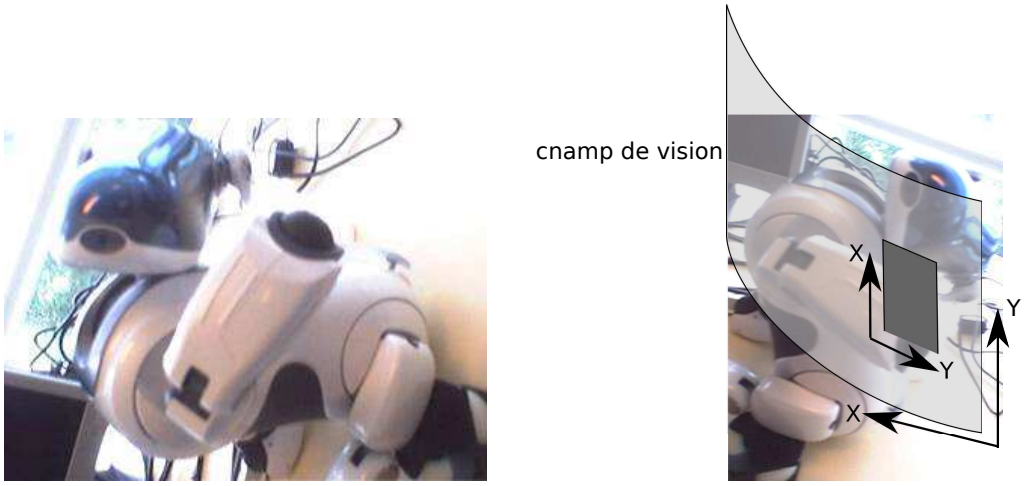


Fig. 2.19: Situation dans laquelle le robot est bloqué due à une rotation supplémentaire du champ visuel de la caméra (foncé) qui ne permet plus l'extraction de commandes motrices cohérentes

calcul qui créé une latence ; le robot n'apprend donc pas sur un certains nombre de positions intermédiaire. Après plusieurs jours, l'apprentissage de la coordination visuo-motrice n'était donc pas complet laissant ainsi de nombreux "trous" dans la carte visuo-motrice. Par conséquent, lorsque le robot exploite la carte visuo-motrice, des positions visuelles éloignées répondent et engendrent des commandes motrices ne permettant pas de rejoindre la position désirée. Il faudrait donc continuer l'apprentissage pour que le robot puisse complètement terminer sa coordination visuo-motrice.
La solution est qu'il est possible d'échantillonner l'espace global en divisant par 2 le nombre de positions. Cet échantillonnage paraît suffisant vis-à-vis des erreurs des valeurs sur les articulations sans pour autant empêcher la détection de la couleur à l'extrêmité de sa patte. La deuxième cause vient de l'articulation de la nuque (pour la tête) du robot. En effet, même si ce moteur permet un déplacement vertical de la tête, il ne peut finalement pas être tout simplement sommé avec le moteur *Tilt* permettant un déplacement suivant le même axe. Le moteur de la nuque portant les moteurs *Pan* et *Tilt*, il induit une rotation du plan correspondant au champ visuel du robot lorsque celui-ci regarde sur le coté. Cette rotation ne permet alors plus de garder la correspondance des repères de la tête avec celui de la patte. Par conséquent, les commandes motrices extraites par l'architecture ne réalisent pas le mouvement désiré. Une solution serait de tenir compte de cette rotation, par exemple de l'apprendre, pour que le robot puisse extraire des commandes motrices cohérentes, mais ce problème sort du cadre de mes travaux. J'ai donc décidé de passer sur deux degrés de liberté pour la tête et la patte du robot en ne considérant plus que les moteurs "pan" et "tilt" de la tête et les moteurs 1 et 2 de la patte.

Test avec 2 degrés de liberté pour la patte et la tête

Le passage de trois à deux degrés de liberté par périphériques (figure 2.20) implique de refaire l'apprentissage sensori-moteur. De plus, comme dit précédemment, la résolution du champ visuel global est divisée par deux. Ce passage n'est pas sans conséquence. La première est la taille de l'espace visuel qui est donc diminuée[5]. De ce changement de résolution résulte une forte

[5]la résolution de l'espace visuel passe à $(90^\circ/2^\circ) + 28^\circ = 73^\circ$ horizontalement et $((20^\circ - (-16^\circ))/2) + 22^\circ = 40^\circ$ verticalement.

diminution du temps de calcul sur notamment le groupe qui permet d'apprendre les associations visuo-motrices. La diminution du temps de calcul est expliquée par une forte diminution du nombre de poids à apprendre[6]. Une fois les modifications effectuées dans le modèle, l'apprentissage sensori-moteur a été entièrement refait[7].

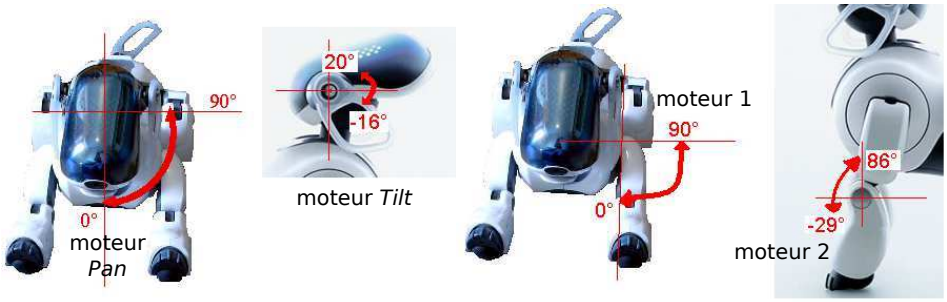


Fig. 2.20: Illustration des degrés de liberté du robot : 2 pour la tête et 2 pour la patte.

A nouveau, j'ai testé ce nouvel apprentissage en faisant centrer la patte du robot dans son champ visuel (figure 2.21).

Fig. 2.21: Illustration du robot ayant sa patte dans son champ visuel local

Avec la diminution de la résolution du champ visuel, le robot a pu apprendre suffisament d'associations entre sa vision et ses informations motrices pour correctement exploiter la carte visuo-motrice. Durant ce test, différentes positions ont été données à la tête du robot de manière à ce qu'il déplace le centre de son champ de vision à différentes positions de l'espace global. Le robot à alors pu correctement centrer l'extrêmité de sa patte au centre de son champ visuel rendant donc compte d'une coordination visuo-motrice exploitable pour tester l'émergence d'un comportement de proto-imitation.

2.3.5 Test d'une imitation

Pour tester le comportement d'imitation, la tête du robot est tournée de 90° de manière à ce qu'il ne voit plus sa propre patte. De manière réflexe le robot suit de la tête le déplacement de la main de l'humain de manière à ce qu'elle soit toujours au centre de sa vision. L'humain

[6]Le nombre de poids sur lesquels est réalisé l'apprentissage passe à $73 * 40 * 2 * 5 = 29200$.

[7]La nuque est positionnée à $-75°$, le moteur 0 de l'épaule à $+10°$. De cette manière, les plans de l'espace visuel global et de l'espace global de la patte sont correctement superposés.

Fig. 2.22: En jouant sur l'ambigüité de la perception, le robot voit la main de l'humain comme si c'était sa patte. Le robot agissant comme un homéostat, il cherche alors à corriger l'erreur entre la position de ce qu'il voit et la position fournie à partir de ces informations motrices. Alors on observe que le robot imite le geste de l'humain avec sa patte.

peut alors déplacer sa main dans le champ visuel du robot. La vision du robot étant ambigüe, il ne fait pas la différence entre la main de l'humain et sa propre patte. L'architecture étant construit comme un homéostat, il va alors chercher à garder l'équilibre entre ce qu'il voit et ses informations motrices. A partir de la perception de la main de l'humain, le robot corrige alors l'erreur entre la position qu'il voit et la position fournie à partir des informations motrices. Finalement, on observe le robot imiter avec sa patte le geste que l'humain est en train de réaliser devant lui (figure 2.22).

2.4 Discussion

Pour faire émerger un comportement d'imitation, le robot doit au préalable apprendre sa coordination visuo-motrice. Dans la pratique, un tel apprentissage prend plusieurs heures, plusieurs jours, voir plusieurs semaines suivant la complexité du robot (nombre de degrés de liberté du bras, échantillonnage de l'espace visuel global) avant que cet apprentissage soit exploitable. Ce temps d'apprentissage reste acceptable comparé au processus équivalent chez le jeune enfant pour qui il dure plusieurs années. Néanmoins, malgré le temps pris par ce processus, l'apprentissage n'est pas toujours complet. En effet, des tests réalisés en simulation (simulation d'une caméra montée sur deux moteurs *Pan* et *Tilt* et d'un bras à six degrés de liberté) ont montré que certaines zones de l'espace visuel global du système simulé n'étaient pas suffisamment apprises (figure 2.23). Par conséquent, lorsque le robot simulé devra exploiter les informations relatives à cette zone, elles seront erronées et entraînera un comportement inapproprié du bras simulé.
Des tests sur la méthode d'apprentissage ont été alors réalisés avec un stagiaire [Bailly, 2007] afin d'essayer de limiter ce problème. Ce test consiste à faire diminuer la vitesse d'apprentissage par paliers et à modifier le voisinage de chaque neurone des micro cartes impactées par l'apprentissage. De cette manière, le premier apprentissage permet de placer en un coup (vitesse d'apprentissage à 1 avec un voisinage impactant tous les neurones d'une micro carte) chaque micro carte sur les zones du plan sphérique visuel qu'elles doivent coder (figure 2.24.A) correspondant à un apprentissage "le bras tendu". Puis, dans sur les paliers suivant, la vitesse d'apprentissage et le voisinage sont diminués progressivement de manière à répartir de plus en plus finement, les neurones de chaque micro cartes dans l'espace (figure 2.24.B, C et D).
Ce test met en évidence que l'apprentissage s'effectue dans un premier temps, d'une manière grossière puis s'affine. L'apprentissage converge alors de manière plus efficace en répartissant mieux les neurones dans l'espace global. Ce processus pourrait être comme une sorte de maturation de structures cérébrales telle qu'elle a pu être mise en évidence pour les aires visuelles primaires [Hubel et Wiesel, 1965] ou alors aires motrices. Des travaux en cours au laboratoire visent à chercher de nouveaux algorithmes pour permettre au robot d'estimer la position de

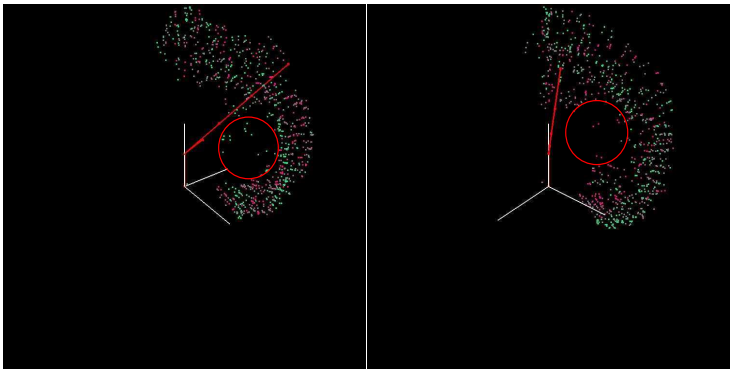


Fig. 2.23: Coupe verticale de l'espace visuel global mettant en évidence une zone de l'espace non suffisamment apprise (zone encerclée). Le squelette rouge est le bras simulé (trois degrés de liberté). Chaque point dans l'espace est un neurone codant pour sa position dans l'espace.

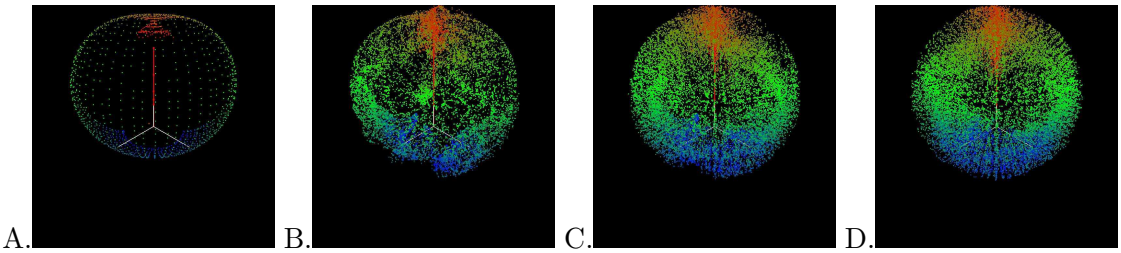


Fig. 2.24: Evolution de l'apprentissage en quatre paliers avec des micro cartes de 10 neurones A. premier palier : $eps = 1$ et $voisinage = 5$ B. second palier : $eps = 0.25$ et $voisinage = 4$ C. troisième palier : $eps = 0.05$ et $voisinage = 3$ D. quatrième palier : $eps = 0.01$ et $voisinage = 2$

son effecteur dans l'espace global sans pour autant avoir appris au préalable les associations sur toutes les positions visuelles. Ces travaux reposent sur un nombre limité d'attracteurs (une dizaine) définis dans l'espace moteur. Ces attracteurs sont associés à des positions visuelles particulières sur une carte visuo-motrice. Alors, lorsque le bras doit rejoindre une position visuelle, le stimulus visuel va activer plus ou moins fortement les différents attracteurs voisins en fonction de leur distance au stimulus. Grâce à ce mécanisme, le robot peut découvrir de nouvelles positions visuelles non apprises auparavant. Ces travaux permettraient d'accélerer énormément le temps d'apprentissage. Une seconde étape d'amélioration serait d'apprendre les associations attracteurs-mouvements du bras. Ce modèle se rapproche alors des travaux réalisés en navigation permettant à un robot de se déplacer dans l'environnement en apprenant des associations lieux-actions qui seront utilisées dans les chapitres 4 et 5.

2.5 Conclusion

Dans ce chapitre, j'ai montré comment l'imitation pouvait émerger de mécanismes plus bas niveau en me concentrant sur les modèles d'asservissement visuels. J'ai ensuite appliqué et testé un modèle qui permet l'émergence de comportement d'imitation bas niveau. Ce modèle est construit comme un homéostat qui tend à équilibrer par l'action ses informations perceptives frustres (détection du mouvement ou de la couleur). Ce modèle implique que le robot ait au préalable associé les positions visuelles de son effecteur avec les informations proprioceptives de

ses moteurs.
Les tests ont été réalisés sur un robot Aibo mécaniquement différent d'autres robots utilisés avec le même modèle [Andry *et al.*, 2002]. Ceci a permis de montrer que le modèle est indépendant des propriétés mécaniques du robot. J'ai alors pu tester les limites du modèle (espace de travail trop petit, patte trop grosse dans le champ de vision). Les résultats en simulation avec un bras à six degrés de liberté ont mis en évidence un temps de convergence de l'apprentissage trop important (plusieurs années).
Cependant, grâce au comportement d'imitation immédiate, le robot est capable de réaliser des mouvements. Ces mouvements peuvent être appris sous forme de séquences pour permettre au robot d'apprendre des comportements plus complexes.

Chapitre 3

Apprentissage de séquences

Dans le cadre de l'apprentissage de comportements en robotique, un comportement peut être codé comme une succession d'événements sensori-moteurs. Le robot apprend alors à prédire le prochain événement à partir des précédents. Un système ayant la capacité d'apprendre une séquence doit donc être capable d'associer un événement avec ceux du passé. Par conséquent un tel système doit comporter une mémoire du passé et être capable d'associer le présent avec tout ou partie du passé (figure 3.1).

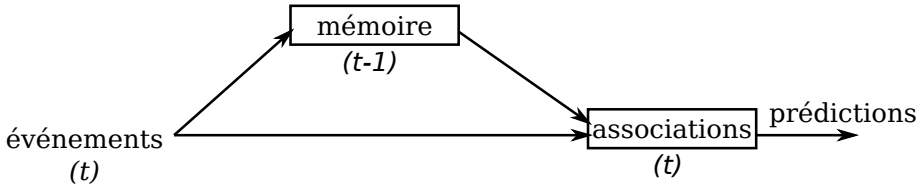


Fig. 3.1: Illustration d'un modèle d'apprentissage de séquence associant un événement aux événements antérieurs mémorisés.

Dans une version simple, il est possible d'apprendre une séquence en associant un événement à l'événement précédent ; c'est-à-dire avec une mémoire d'un seul événement. Dans ce cas l'association apprise entre deux événements est une transition. De cette manière, en présentant un événement, le système prédira le suivant.
Ce type de modèle permet d'apprendre des séquences simples, c'est à dire des séquences composées d'événements présents une seule et unique fois. En effet, dans une séquence comportant plusieurs fois un même élément, la reconnaissance seule de cet événement ne permet pas de choisir ses différents successeurs. L'événement n'est qu'un observable ambigu. Pour lever l'ambiguïté, il est alors nécessaire d'ajouter de l'information.
Dans la suite, deux modèles seront proposés. Un premier modèle apprenant des séquences temporelles simples et un second reposant sur un réservoir de dynamiques. Ces deux modèles permettent d'aborder deux grandes familles d'apprentissage de séquences. L'une basée sur l'association rapide d'événements et l'autre basée sur un réservoir de dynamiques. Pour finir un modèle tirant partie de la richesse de dynamiques pour l'apprentissage rapide de séquences temporelles complexes sera proposé.

3.1 Modèle neuro mimétique pour la prédiction du timing

En tant qu'être humain, nous sommes tous capable d'apprendre des séquences, qu'il s'agisse de séquences de nombres, de notes musicales, de séquences d'actions, etc. La question qui se pose est comment notre cerveau apprend-il ces séquences ? Quelles sont les structures cognitives impliquées ? Quels sont les mécanismes sous-jacent ?
Il existe de nombreux travaux portant sur l'étude des mécanismes cognitifs entrant en jeu lors de l'apprentissage de séquences. Parmi les structures revenant le plus souvent dans la littérature, on trouve l'implication du cervelet, du cortex ainsi que de l'hippocampe. Ces différentes structures représentent plusieurs types de mémoires stockant différents types d'informations plus ou moins longtemps dans la vie de l'être humain.

3.1.1 Les mémoires du cerveau

Il est possible de distinguer ces mémoires à partir de critères différents. On catégorise ces mémoires selon la nature des informations et leur durée de mémorisation. On dégage souvent deux

catégories : la mémoire à long terme et la mémoire à court terme. La mémoire à court terme est aussi appelée mémoire de travail, car nous l'utilisons en permanence. Les éléments stockés dans ce type de mémoire ont une persistence très courte (quelques secondes) et le nombre d'éléments mémorisés est limité. A contrario, la mémoire à long terme permet de stocker beaucoup plus d'éléments et sur de bien plus grandes périodes, voire de manière différente pendant toute une existence.
On trouve également plusieurs distinctions faites selon le type d'information stockée. On différencie la mémoire épisodique de la mémoire sémantique et la mémoire déclarative de la mémoire procédurale. Comme son nom l'indique, la mémoire épisodique permet de se souvenir d'épisodes particuliers de notre vie. La mémoire sémantique permet de stocker des informations beaucoup plus abstraites comme des concepts, des symboles. La mémoire procédurale nous permet d'acquérir des savoir-faire. Elle est beaucoup plus liée aux actions motrices que nous réalisons. Quant à la mémoire déclarative, elle nous permet d'exprimer nos souvenirs.
Ici, j'ai volontairement mis à plat différentes mémoires, mais elles ne sont pas indépendantes, elles sont chacune liées à d'autres structures cognitives, d'autres mémoires. Une question faisant encore débat aujourd'hui est : quelles structures cognitives jouent un rôle dans quel type de mémoire ? Mes travaux ne portant pas sur l'étude des modèles neurobiologiques de la mémoire, je ne vais pas détailler ici les différents travaux animant les débats sur cette question. Néanmoins, je ferai une rapide présentation du cervelet et de l'hippocampe structures inspirant certaines propriétés des modèles que j'utiliserai dans ce chapitre.

3.1.2 Le cervelet

Le cervelet est situé en dessous des hémisphères cérébraux et en arrière du tronc cérébral, en particulier, en arrière du bulbe rachidien et de la protubérance (figure 3.2.A).

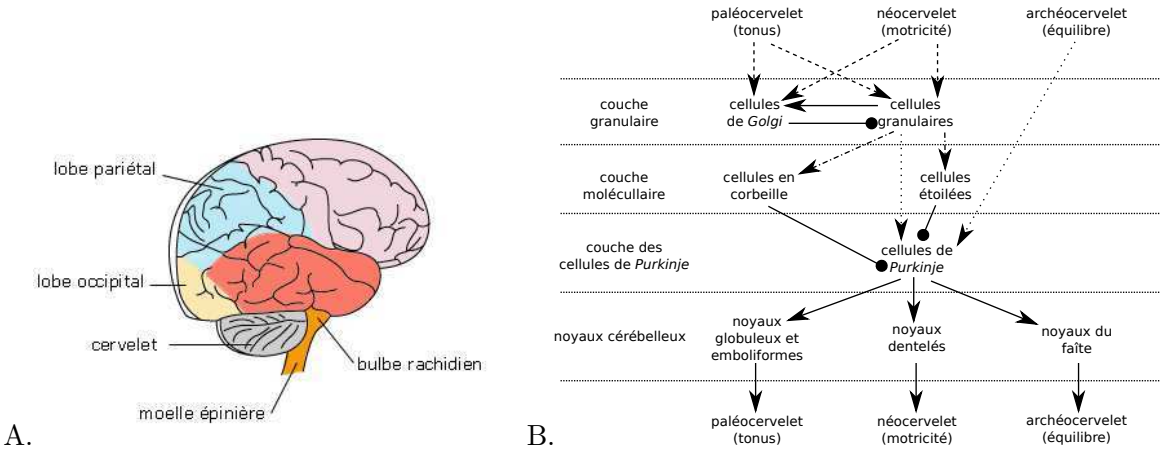


Fig. 3.2: A) Localisation du cervelet dans le cerveau. B) Schémas des connexions du cervelet.

Le cervelet est subdivisé en trois régions : l'archéocervelet, le paléocervelet et le néocervelet. Chacune de ces régions est connectée à une structure cérébrale spécifique, et est impliquée ainsi dans une fonction précise. L'archéocervelet est connecté au vestibule et est impliqué dans l'équilibre. Le paléocervelet est connecté à la moelle et est impliqué dans le tonus musculaire. Le néocervelet est connecté au cortex et est impliqué dans la motricité. A l'intérieur, le cervelet est organisé en cortex cérébelleux en surface et des noyaux en profondeur. Le cortex cérébelleux est découpé en trois couches : la couche granulaire, la couche molléculaire et la couche des cellules de

Purkinje (figure 3.2.B). Les cellules de *Golgi* et les cellules granulaires reçoivent les informations de la moelle (tonus) et du cortex (motricité) par les fibres moussues. Ces deux types de cellules sont inteconnectées via des connexions exitatrices partant des cellules granulaires vers les cellules de *Golgi* et des connexion inhibitrices dans le sens inverse. Puis, les cellules granulaires projettent leurs axones, appelés fibres parallèles, vers la couche molléculaire. La couche molléculaire est composée d'inter-neurones inhibiteurs nommés cellules étoilées ou en corbeille qui transmettent les activités aux cellules de *Purkinje*. La couche des cellules de *Purkinje* reçoit les informations de la couche molléculaire et du bulbe rachidien (équilibre) via les fibres grimpantes. Les cellules de *Purkinje* envoient les activités vers les noyaux cérébelleux pour les transmettre aux différentes structures impliquées dans l'équilibre (bulbe), la motricité (cortex via thalamus) et le tonus musculaire (moelle). Le cervelet reçoit les informations de l'intention d'un mouvement du cortex moteur et retourne à ce dernier les caractéristiques nécessaires à l'execution de ce mouvement. Le cervelet joue alors un rôle important dans la mémorisation de comportements moteurs [Doya, 2000] et dans le timing de ces mouvements [Ivry *et al.*, 2002]. Le micro circuit entre les cellules granulaires et les cellules de *Golgi* permettraient la gestion du timing dans le cervelet [Buonomano et Mauk, 1994].

3.1.3 La boucle hippocampique

L'hippocampe est une structure corticale faisant parti du système limbique. Elle est présente dans les deux hémisphères du cerveau. Cette structure se compose principalement de la Corne d'Ammon (CA) et du Gyrus Dentelé (DG). La Corne d'Ammon est elle même subdivisée en trois sous-structures CA1, CA2, CA3. Le Cortex Enthorinal (EC), ainsi que le subiculum sont deux structures para-hippocampique qui jouent le rôle respectivement d'entrée et de sortie de l'hippocampe (figure 3.3).

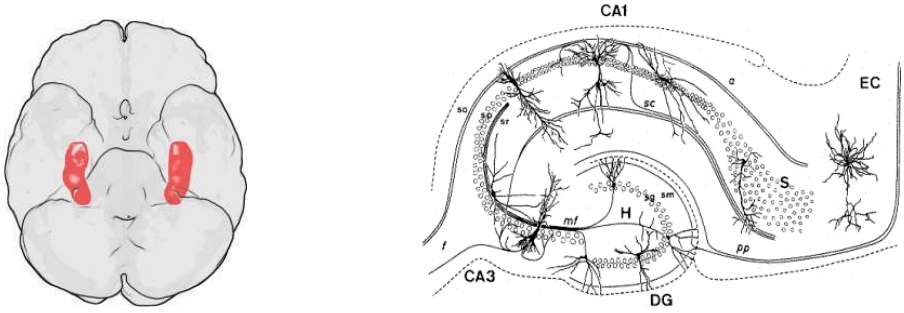


Fig. 3.3: A gauche : localisation de l'hippocampe (en rouge) dans le cerveau. A droite : structures composant l'hippocampe

La structure DG est composée de cellules granulaires massivement connectées aux cellules de CA3. Les cellules de CA3 se projettent vers les cellules de CA1 (collatérale de Schaeffer) [Schultz *et al.*, 2000], elles mêmes se projetant vers le subiculum. Le subiculum est ensuite connecté aux couches profondes de EC. EC faisant figure de porte d'entrée de l'hippocampe est divisé en plusieurs couches : les couches superficielles et les couches profondes. En plus des connexions des couches profondes entre elles d'une part, et des couches profondes vers les couches superficielles d'autre part, des couches superficielles 2 et 3 partent vers les cellules pyramidales de CA3 ainsi que celles de CA1. Cette double connectivité est encore aujourd'hui mal comprise.
Les multiples connexions qui proviennent de différentes zones du cortex vers l'hippocampe, ainsi que les connexions partant de l'hippocampe vers les différents cortex, soulèvent la question de

la nature des informations que l'hippocampe mémorise.
On distingue principalement deux réponses bien différentes sur la nature des informations encodées dans l'hippocampe. La première est que l'hippocampe a la capacité de coder des informations spatiales. Cette réponse est principalement apportée par des études effectuées sur le rat. La deuxième réponse, bien que différente, mais non exclusive de la première, est que l'hippocampe est une mémoire épisodique. Cette réponse provient d'études réalisées chez l'homme.
Des travaux montrent que l'hippocampe ne garde que très peu de temps les nouvelles informations apprises puis, durant certaines phases de sommeil, le transfert vers d'autres structures cognitives [Siapas et Wilson, 1998]. Pour compléter, de récentes études montrent que l'hippocampe émettrait des signaux permettant de prédire le niveau de récompense qui serait obtenu [Vanni-Mercier *et al.*, 2009]. Plus le signal est fort, moins la personne est certaine d'obtenir une récompense. Le signal coderait alors pour un niveau d'incertitude, ce qui en fait un signal d'alerte.

3.1.4 Model computationnel de l'hippocampe

Dans [Banquet *et al.*, 1997], les auteurs proposent un modèle computationnel de l'hippocampe. Dans ce modèle on retrouve les structures principales de l'hippocampe : EC, DG, CA1, CA3 ainsi que le subiculum (figure 3.4).

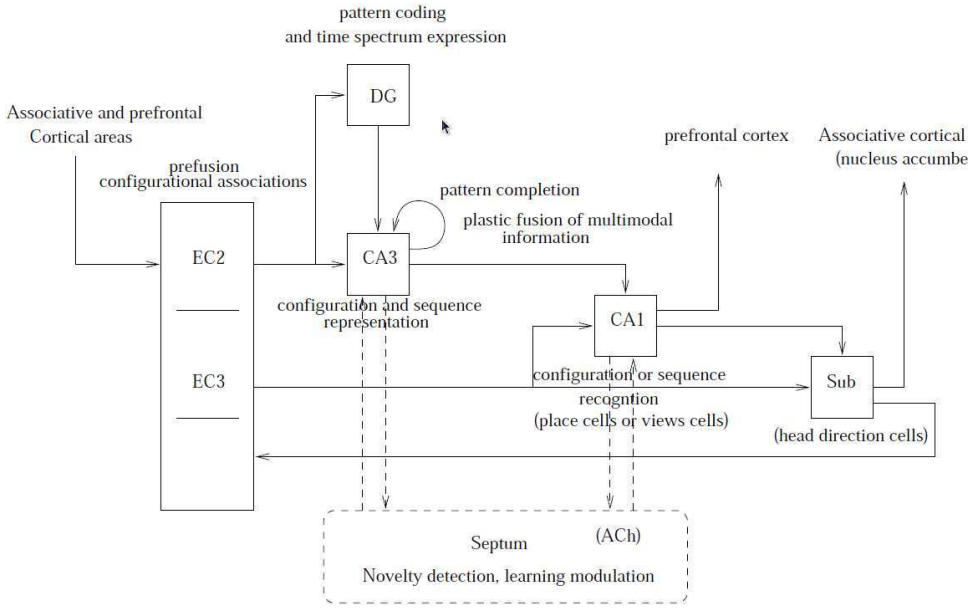


Fig. 3.4: Représentation schématique du modèle de l'hippocampe extrait de [Banquet *et al.*, 1997]

Ici, EC reçoit des informations des aires corticales et les fusionne. Ces informations sont ensuite transmises à DG ainsi qu'aux cellules pyramidales de CA3. Dans ce modèle, on suppose que les cellules granulaires de DG constituent une base de temps qui permettent de maintenir une activité temporelle. Les liens récurrents de CA3 permettent la reconstruction de formes et l'association d'une forme arrivant de EC avec une précédente forme. CA3 est donc capable d'apprendre et restituer des séquences. La forme arrivant de EC est maintenue dans DG. La forme reconnue par CA3 est ensuite intégrée par CA1 permettant la reconnaissance de configurations ou de séquences. Puis l'information intégrée par CA1 est traitée par le subiculum.

3.1.4.1 Modèle d'apprentissage de séquences temporelles simples

Dans ce chapitre, je nomme les différents groupes de ce modèle par les noms des structures de la boucle hippocampique, mais il est important de garder à l'esprit que ce modèle est une abstraction correspondant aussi bien à l'hippocampe qu'au cervelet. Dans le cadre de l'apprentissage de séquences temporelles, ce sont principalement les fonctions des structures EC, DG et CA3 qui m'intéressent. EC joue un rôle de détecteur d'entrée. De manière simple, il peut être interprêté comme un dérivateur qui effectuerait la dérivée temporelle des activités en entrée. Si la dérivée est nulle ou négative, alors le groupe ne répond rien en sortie. De cette manière, l'activité en sortie de EC est une impulsion détectant le début d'une activité en entrée. Ceci donne une propriété de filtrage sur la répétition non désirable d'un état. Le groupe DG permet de maintenir dans le temps une trace de l'activité transmise par EC. Le groupe CA3 apprend des transitions entre une activité en entrée et la précédente. Par conséquent, ce groupe a autant de neurones que de transitions possible, c'est-à-dire le nombre de neurones dans le groupe EC au carré. La figure 3.5 illustre le modèle que je considère ici.

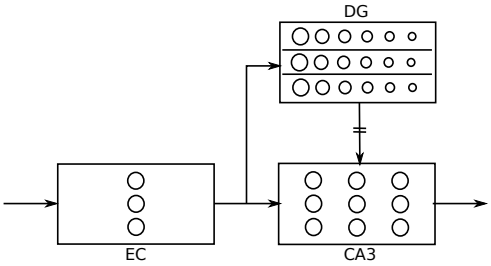


Fig. 3.5: Modèle d'apprentissage de séquence temporelle simple inspirée de structures de l'hippocampe. EC est l'entrée du modèle. DG maintient l'activité d'une entrée sur EC grâce à ses cellules granulaire. CA3 apprenant les associations entre l'état précédent maintenu dans DG et l'état courant venant de EC (connexions de un vers tous doublement barrées). Se référer à la figure 3.8 pour le détail des connexions

Lorsqu'une première entrée est présente dans EC, celui-ci la transmet à CA3. Le groupe DG n'ayant pas encore d'activité, CA3 n'apprend pas d'associations. L'activité sur EC est également transmise au groupe DG. Ce dernier est organisé en bancs de neurones chacun correspondant à une trace de l'activité sur EC. L'activité de chacun des neurones de DG suit l'équation :

$$Act_{j,l}^{DG}(t) = \frac{1}{m_j} \cdot \exp -\frac{(t-m_j)^2}{2 \cdot \sigma_j} \tag{3.1}$$

où l correspond à l'indice de la cellule activée sur la ligne, j est l'indice de la ligne, m_j est une constante de temps et σ_j son écart type associé. t est le temps en millisecondes. Les activités des cellules d'une ligne (représentées figure 3.6), se répartissent ainsi au cours du temps et représentent une trace sur plusieurs secondes de l'activation "EC" passée (le temps exact dépendant du nombre de cellules de la ligne et des constantes σ_j choisies).
Lorsqu'une seconde entrée est présente sur EC, l'activité est transmise au groupe CA3 qui déclenche l'apprentissage d'une transition entre l'activité maintenue dans DG et l'activité transmise par EC. La modifications des poids lors de l'apprentissage a lieu sur les connexions entre DG et CA3 et consiste en la normalisation des activités de DG dans les poids de connexions avec le neurone de CA3 activé par EC. L'apprentissage se fait en un coup suivant l'équation :

$$W_{CA3(i,j)}^{DG(j,l)} = \begin{cases} \frac{Act_{j,l}^{DG}}{\sum_{j,l}\left(Act_{j,l}^{DG}\right)^2} & si\ Act_j^{DG} \neq 0 \\ inchangé & sinon \end{cases} \tag{3.2}$$

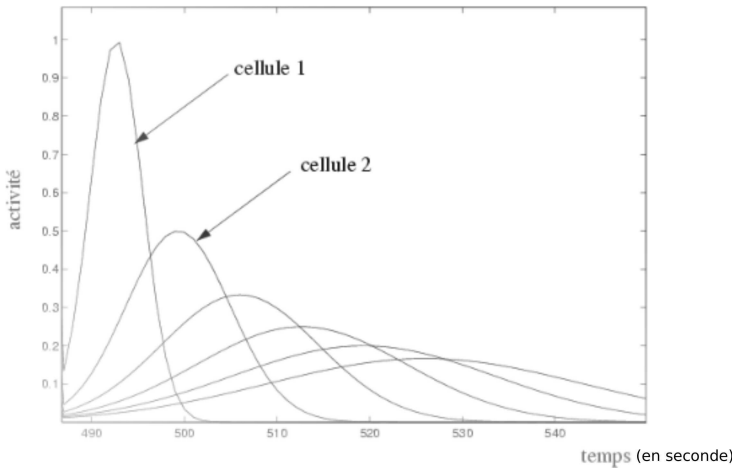


Fig. 3.6: Courbes d'activité de 6 cellules d'un banc de neurones du groupe temporel DG activé à $t = 485$ s.

avec $W_{CA3(i,j)}^{DG(j,l)}$ le poids de la connexion entre le banc de neurones j du groupe DG et le neurone i de CA3. $Act_{j,l}^{DG}$ l'activité du neurone l du banc j du groupe DG. Ainsi, le rappel de la transition apprise (c'est-à-dire l'activation du neurone sur CA3) ne pourra avoir lieu que si la somme des activités des cellules de DG est égale à celle rencontrée lors de l'apprentissage.

3.1.4.2 Simulations avec l'apprentissage de séquences temporelles simples

Dans cette section, je présente les résultats de tests réalisés sur l'architecture d'apprentissage de séquences temporelles. L'architecture considérée (figure 3.7.A) intègre une connexion de retour entre la sortie du groupe "CA3" et l'entrée du groupe "EC". Cette connexion ajoutée à l'architecture permet de simuler la fermeture de la boucle via l'environnement (figure 3.7.B).

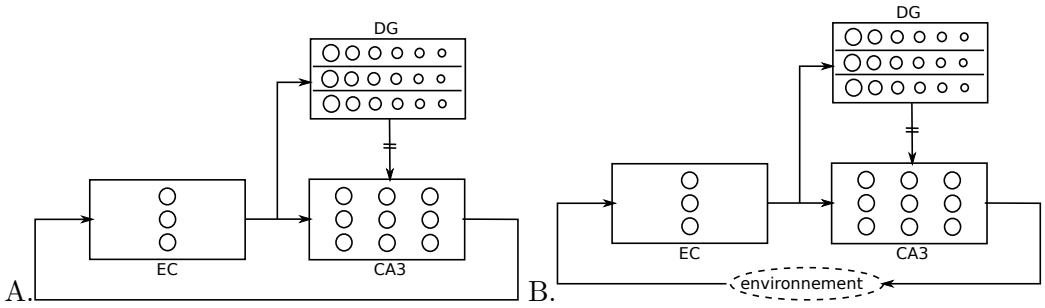


Fig. 3.7: A) Architecture neuronale utilisée en simulation. Le rebouclage entre la sortie du groupe CA3 et l'entrée du groupe EC est réalisé par une connexion directe entre ces groupes. B) Architecture neuronale utilisée lors d'expériences sur des robots. Le rebouclage entre la sortie du groupe CA3 et l'entrée du groupe EC est réalisé par l'environnement ou la proprioception.

L'architecture d'apprentissage de séquences temporelles permet de coder des séquences simples en apprenant les transitions entre chaque état. Ces transitions sont codées par chacun des neurones du groupe "CA3". Pour trois états en entrée, il existe $3 * 3 = 9$ transitions. Donc le groupe "CA3" possède neuf neurones (figure 3.8). Par conséquent, lorsque l'état "0" est présenté en entrée sur le groupe "EC", le neurone codant la transition dans "CA3" déclenchera un impulsion (figure 3.9).

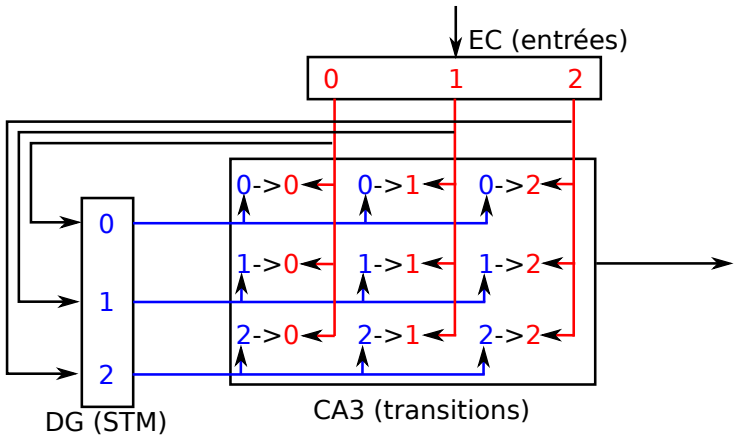


Fig. 3.8: Architecture neuronale détaillant l'encodage des transitions.

Par exemple, pour une séquence "0 1" apprise (figure 3.9), un neurone de "CA3" apprendra la transition "0" → "1".

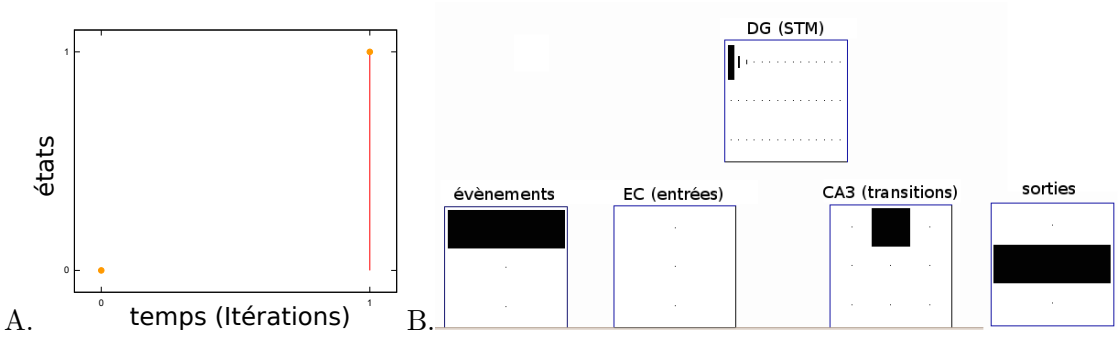


Fig. 3.9: A) Exemple d'une séquence d'états "0 1" apprise. B) Capture d'une simulation consistant à reproduire la séquence "0 1". L'état "0" déclenche un évènement qui est ensuite détecté par le groupe "EC" puis mémorisé par le groupe "DG". Le groupe "CA3" déclenche la transition correspondante et prédit l'état suivant de la séquence en sortie ; c'est-à-dire "1".

A travers cet exemple très simplifié, on se rend compte que cette architecture est capable d'apprendre des transitions permettant de rejouer une séquence. Les propriétés dynamiques des neurones du groupe "DG" permettent de maintenir l'activité de l'état du passé au delà d'une itération. Par conséquent, une séquence "0 1" peut être apprise avec des intervalles de temps différents. Pour matérialiser ces intervalles, je noterai le caractère "-" signifiant qu'il n'y a pas eu de nouvel évènement et que l'état précédent est maintenue en mémoire. De cette manière, dans la séquence "0 - 1" signifie que l'activité de l'état "0" est maintenue dans le groupe "DG" jusqu'à l'arrivée de l'état "1". De la même manière, la séquence "0 - - 1" montre que l'état "0" sera maintenue plus longtemps en mémoire. Ainsi, plus il y a d'états "-", plus l'état précédent est maintenue en mémoire. Ainsi la séquence peut être encodée de différentes manières (figure 3.10) : "0 - 1", "0 - - - 1", etc.
Le modèle simulé ici possède la propriété de permettre de débuter une séquence par n'importe quel état qui la compose. Cette propriété résulte de l'encodage de séquences par apprentissage des transitions. Par exemple, pour une séquence comme "0 - - 1 - - - - 2" , lorsqu'on donne l'état "1" à l'architecture, elle prédira l'état "2" en respectant le timing de la transition "1"→"2" (figure

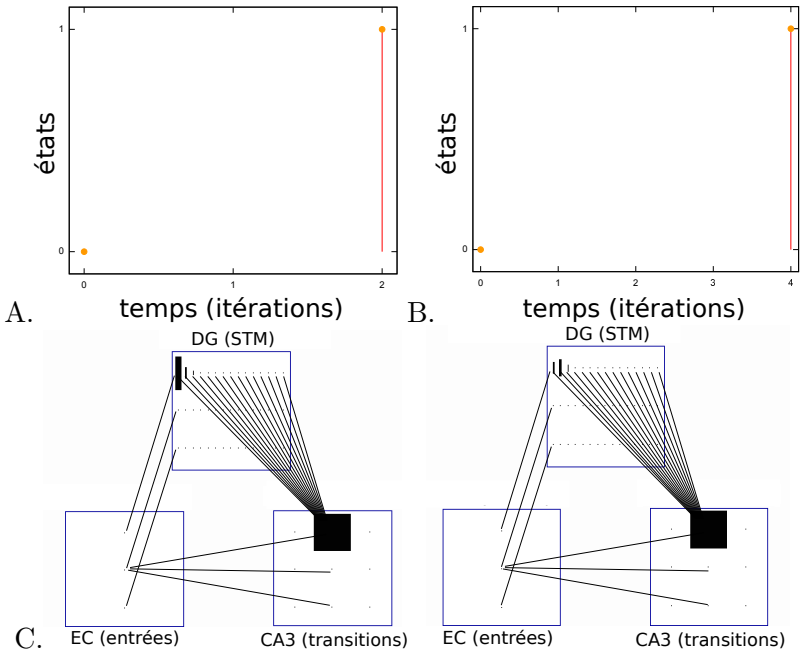


Fig. 3.10: Apprentissage et reproduction de la même séquence “0 1” avec deux intervalles de temps différent. A) Séquence apprise avec deux itérations entre les états “0” et “1”. B) Séquence apprise avec quatre itérations entre les états “0” et “1”. C) Captures des simulations lors de la reproduction de chacune des séquences. Lorsque le groupe “CA3” déclenche la transition, le premier état “0” a été mémorisé moins longtemps (en bas à gauche) lorsque la séquence a été apprise avec deux itérations entre les états que lorsqu’elle a été apprise avec quatre itérations entre les états (en bas à droite).

3.11).

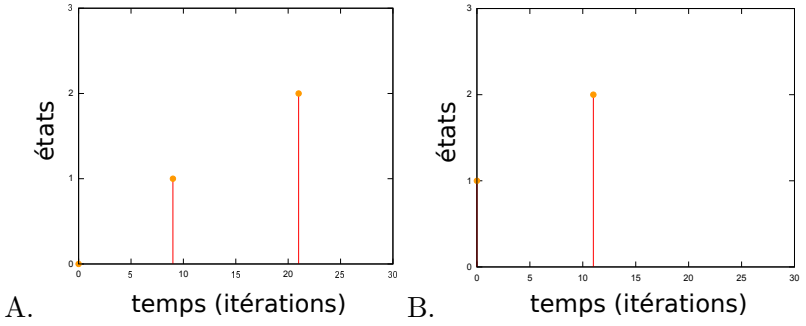


Fig. 3.11: A) Séquence “0 1 2” apprise avec 12 itérations entre l’état “1” et l’état “2”. B) Reproduction de la séquence “0 1 2” en débutant la séquence au milieu avec l’état “1”. 11 itérations plus tard, l’architecture déclenche la transition “1”→”2” permettant la prédiction de l’état “2”.

Cette architecture suppose que chaque état qui compose la séquence est unique. Si une séquence comporte deux fois un même état, alors l’architecture apprendra deux transitions à partir de cet état. Intuitivement, on peut penser que l’architecture réalisera la prédiction de deux états. Par exemple, pour une séquence “0 1 0 2”, le groupe “CA3” apprendra les transitions “0”→“1” et “0”→“2”. On s’attend alors, lorsque l’état “0” est donné, que le groupe “CA3” prédise les états “1” et “2” au même moment. Tout d’abord, l’apprentissage n’est pas seulement réalisé sur les transitions, mais sur les intervalles de temps des transitions.
Dans le cas de tests en simulation avec l’architecture utilisée jusqu’alors (figure 3.7.A), deux

cas de figure sont possibles. Le premier, apparait dans le cas où les deux transitions '0"→"1" et "0"→"2" ont le même timing. Dans ce cas, l'architecture aura le comportement décrit précédemment, c'est-à-dire que le groupe "CA3" prédira les deux états "1" et "2" au même moment (figure 3.12). Si un tel cas de figure se trouve dans une séquence plus longue, alors la suite de la séquence dépendera du choix de l'état prédit en fonction de la compétition entre les états prédits, du bruit en sortie, etc. Dans ce cas, le résultat ne sera pas déterministe. Le deuxième cas de figure apparait dans le cas où la première transition a un timing plus court que la deuxième. Dans ce cas, l'apprentissage de la deuxième transition ne s'effectuera pas. En effet, lors de la seconde présentation de l'état "0", le groupe "CA3" déclenchera la prédiction de l'état "1" correspondant à la première transition apprise. Comme l'architecture considérée ici à une connexion de retour de la sortie du groupe "CA3" vers le groupe "EC", cette prédiction sera réinjectée en entrée. La séquence continuera donc à partir de l'état "1" et ne correspondra pas à la séquence d'origine. On voulait faire apprendre "0 1 0 2" et finalement on obtiendra la séquence erronée "0 1 0 1 . . .".

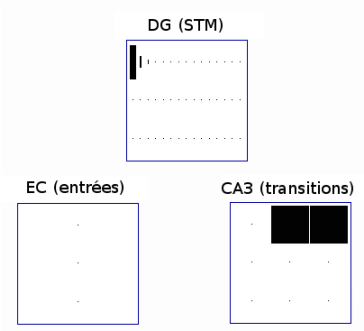


Fig. 3.12: Capture d'une simulation lors de la reproduction d'une séquence complexe "0 1 0 2". Lors de la reproduction, L'état "0" est donné en entrée de l'architecture. Le groupe "CA3" déclenche les deux transitions "0"→"1" et "0"→"2".

Cette limitation du modèle ne permet donc pas d'apprendre des séquences complexes (avec la répétition d'un même état). Pour apprendre des séquences plus complexes, il est nécessaire de lever l'ambiguïté de l'état "0" en créant des états "0a" (pour la première apparition de l'état "0") et "0b" (pour la seconde apparition de l'état "0") et les rendre observable. Une réponse est apportée par les modèles reposant sur les propriétés de réservoirs de dynamiques pour apprendre des séquences complexes.

3.2 Modèles à réservoir de dynamiques

Ce type de modèle est composé de trois couches : une couche d'entrée, une couche interne et une couche de sortie (figure 3.13). La couche d'entrée reçoit les événements extérieurs au réseau. La couche interne est un réservoir de dynamiques recevant les activités de la couche d'entrée et parfois même de la couche de sortie. La couche de sortie restitue les événements à partir des activités des couches précédentes.
D'une manière générale, un réservoir de dynamiques est un ensemble d'unités connectées entre elles grâce à des connexions récurrentes. Le réservoir génère des dynamiques influencées par les entrées transmises par la couche d'entrée et/ou de sortie du système.

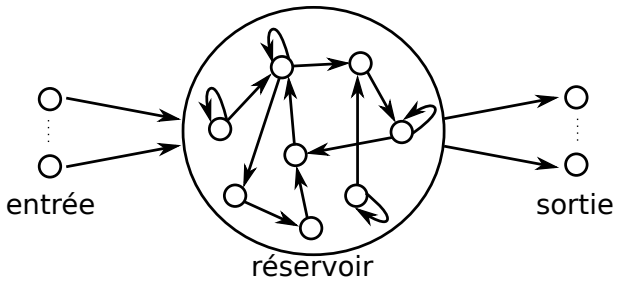


Fig. 3.13: Modèle d'un réseau de neurones à réservoir de dynamiques.

3.2.1 Les systèmes dynamiques

On définit un système dynamique comme un système évoluant dans le temps de façon causale et déterministe [Arrowsmith et Place, 1990]. En effet, à chaque instant, l'état du système dépend de l'état précédent et des conditions initiales. De plus, pour les mêmes conditions initiales, les dynamiques passeront par les mêmes états ; le système est donc déterministe.
L'objectif des modèles à base de réservoir de dynamiques est de faire converger les dynamiques du réservoir vers un ou des attracteurs dont chaque états est représentatifs des activités présentées en entrée du réseau.
Dans le cadre du contrôle d'un robot, la propriété d'attracteur permet de rendre robuste le comportement d'un robot. Un attracteur est un ensemble compact de l'espace des phases, invariant par le flot ou par l'application, vers lequel toutes les trajectoires voisines convergent. Le bassin d'attraction est alors l'ensemble des points dont les trajectoires convergent vers l'attracteur. Par conséquent, lorsque le système se trouve dans un bassin d'attraction, il convergera vers l'attracteur.
Les entrées connectées au réservoir permettent d'agir sur les dynamiques du réservoir les faisant converger vers un régime particulier. Pour les mêmes entrées, les dynamiques convergerons toujours vers un même régime. Ces liens avec les entrées permettent alors au réseau de se resynchroniser par rapport aux états du monde extérieur.
On retrouve les systèmes dynamiques dans de nombreux travaux portant sur le contrôle d'un bras robotique. Dans [Ijspeert *et al.*, 2003], les auteurs utilisent des politiques de contrôle (CP pour "control policiy") à base de systèmes dynamiques pour permettre à un robot de reproduire un geste particulier. Une CP est définie par le but connu à l'avance, les positions angulaires d'un degré de liberté ainsi que des constantes de temps. Dans l'expérience, il y a une CP assignée par degrés de liberté du robot ; il y a donc un système dynamique par articulation. [Degallier *et al.*, 2006] utilisent un système dynamique pour qu'un robot soit capable de réaliser des mouvements rythmiques et discrets. Dans [Iossifidis et Schöner, 2006], les auteurs utilisent un système dynamique qui permet de contrôler un bras robotique ayant des degrés de liberté redondants. Dans l'expérience, le robot doit réaliser une tâche en présence d'un être humain tout en évitant les obstacles.

3.2.2 Le chaos

Cette branche des mathématiques permet d'étudier le comportement de certains systèmes dynamiques. On retrouve les systèmes chaotiques dans divers domaines comme l'étude des dynamiques des fluides [Tritton et Gollub, 1978], l'étude des mouvements des satellites dans le système solaire [Kuang et Tan, 2000] ou même dans l'étude des phénomènes météorologiques [Waelbroeck,

1995]. De la même manière, les systèmes chaotiques permettent également le contrôle de comportements d'un robot [Li *et al.*, 2008].
Un système dynamique est dit chaotique s'il est sensible aux conditions initiales. La moindre variation sur l'état initial pourra engendrer une dynamique complètement différente. Par contre, pour le même état initial, les dynamiques seront exactement les mêmes. Cette propriété en fait donc un système déterministe. Un système chaotique possède un attracteur composé d'une infinité de cycles périodiques instables. Par apprentissage, il est possible d'associer un cycle limite stabilisé avec une entrée [Daucé et Doyon, 1998, Quoy *et al.*, 2001, Berthouze et Tijsseling, 2006]. Dans le cadre de la robotique, l'objectif est de faire converger le système vers une dynamique comportementale particulière. Dans [Duran *et al.*, 2007], les auteurs proposent un modèle reposant sur des systèmes chaotiques pour le contrôle d'un oeil simulé. L'objectif de ce modèle est de réaliser un suivi de cible.

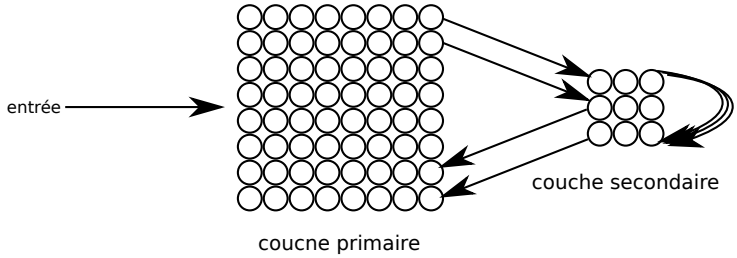


Fig. 3.14: Modèle ReST composé de deux couches. La couche primaire reçoit des activités spatio-temporelles en entrée. La couche secondaire a une dynamique chaotique grâce aux connexions récurrentes. L'activité de la couche primaire dépend alors des activités d'entrée ainsi que des activités de la couche chaotique. Par soucis de clareté, toutes les connexions ne sont pas représentées ; le réseau est entièrement connecté.

Dans [Daucé *et al.*, 2002], les auteurs proposent un modèle de réseau de neurones chaotiques composé de deux couches interconnectées. La première couche sert d'interface d'entrée. La seconde couche possède une dynamique spontannée (figure 3.14). Ce modèle permet d'apprendre des signaux spatio-temporels. Des signaux spatiaux, car il sont transmis sur une population de neurones définissant un espace (par exemple une image visuelle projetée sur la couche d'entrée.). Des signaux temporels, car par l'intermédiaire des connexions récurrentes et des connexions de retour de la couche chaotique vers la couche d'entrée, les activités à chaque instant dépendent des activités générées par les signaux précédents. Le modèle "Resonant spatio-temporal learning" (*ReST*) peut donc apprendre des séquences temporelles. Ce modèle apprend sur toutes les connexions suivant une règle hebbienne à temps discret. Le timing strict (non constant) entre les mouvements du robot et l'apprentissage limite son utilisation à des environnements très contraints. L'apprentissage diminue la dimensionnalité de la couche chaotique pour qu'elle converge vers un régime particulier représentatif des entrées. Cependant, il est difficile de déterminer quand l'apprentissage doit être stoppé le rendant inapproprié pour un robot autonome. Par contre, les propriétés apportées par l'utilisation d'un réservoir de dynamiques sont tout à fait intéressantes pour resynchroniser la reproduction de séquences temporelles complexes à partir d'informations sensorielles.
Ces propriétés sont également présentes dans les *echo state networks* (ESN). J'ai étudié ce modèle récent de plus près pour l'apprentissage de séquences complexes.

3.2.3 *Echo States Networks*

Les "Echo State Networks" (ESNs) sont des réseaux de neurones reposant sur un réservoir de dynamiques [Jaeger, 2001]. Cette idée de réservoir est également partagée dans le modèle des "Liquid State Machines" (LSMs) [Maass *et al.*, 2002] qui a été développé indépendemment et utilisant des neurones à décharge (*spiking neunrons*) à temps continu.
D'une manière générale, le modèle des ESNs permet d'associer les états ($u(n)$) du signal d'entrée avec les états ($y(n)$) du signal désiré. Les activités de chacun des signaux modifient la dynamique du réservoir pour la faire converger vers un régime représentatif de ces activités et de celles qui ont précédées. Sous certaines conditions, l'activation d'un état $x(n)$ du réservoir au temps n est une fonction de l'historique de l'état d'entrée $u(n)$:

$$x(n) = E(u(n), u(n-1), u(n-2), \dots) \tag{3.3}$$

avec $E()$ la fonction d'echo de l'historique des entrées.
Un ESN se compose de trois couches : la couche d'entrée K, la couche interne (le réservoir de dynamiques) N et la couche de sortie L (figure 3.15).

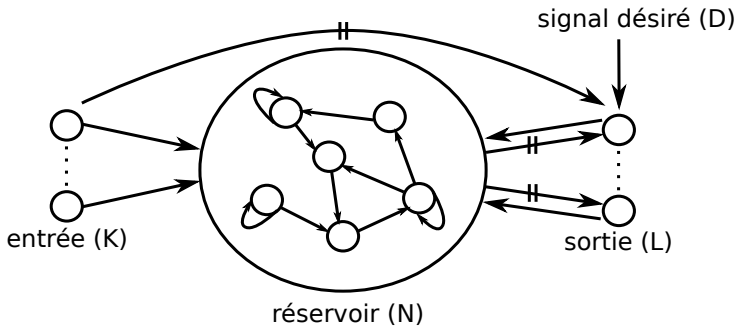


Fig. 3.15: Modèle d'un réseau Echo State Network (ESN) avec une couche d'entrée de dimension K, un réservoir de dynamiques de dimension N et une couche de sortie de dimension $L = 1$.

Les activités des neurones du réservoir de dynamiques sont mises à jour suivant l'équation :

$$x(n+1) = f(Wx(n) + W^{in}u(n+1) + W^{fb}y(n)) \tag{3.4}$$

avec $x(n)$ l'état du réservoir de dimension N au temps n. $u(n)$ est l'état de la couche d'entrée de dimension K. $y(n)$ est l'état de la couche de sortie de dimension L. f une fonction sigmoïde. W est la matrice NxN des poids des connexions entre les neurones du réservoir. W^{in} est la matrice KxN des poids des connexions entre la couche d'entrée et le réservoir. W^{fb} est la matrice NxL des poids des connexions entre la couche de sortie est le réservoir.
L'apprentissage est réalisé sur les connexions entre le réservoir et la couche de sortie. Toutes les autres connexions ont des poids non modifiables et choisis aléatoirement. L'apprentissage est réalisé hors ligne. Durant une première phase le réseau est entraîné avec le signal qu'on souhaite lui faire apprendre. Les activités des différentes couches sont enregistrées pour permettre de réaliser l'apprentissage, le calcul des poids des connexions modifiable (entre le réservoir et la couche de sortie) n'est pas réalisé directement par le réseau pendant la simulation, mais en dehors. L'apprentissage consiste à réaliser une régression linéaire sur le signal de sortie désiré. Le calcul des poids modifiables peut être réalisé par une pseudoinverse :

$$W^{out} = (S^{+}D)^{'} \tag{3.5}$$

Une fois le calcul des poids terminé, ils sont chargés sur les connexions entre le réservoir et la couche de sortie. Finalement, la séquence apprise peut être reproduite.
Le fonctionnement permet d'associer deux signaux différents. Mais il est facile de permettre au modèle d'apprendre l'équivalent de transitions d'un état $y(n-1)$ à l'état $y(n)$ suivant.

3.2.4 Tests avec les *Echo States Networks*

Les tests présentés dans cette section se concentrent principalement sur la capacité des ESNs à apprendre des séquences. Par conséquent, le modèle testé est composé de deux couches : le réservoir de dynamiques et la couche de sortie (figure 3.16).

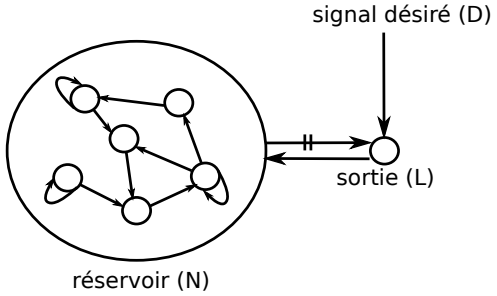


Fig. 3.16: Modèle d'un réseau Echo State Network (ESN) permettant l'apprentissage de séquence avec un réservoir de dynamiques de dimension N et une couche de sortie de dimension L.

La séquence à apprendre est donnée sur la couche de sortie qui est ensuite transmise au réservoir par les connexions de retour. En phase d'apprentissage, les connexions allant du réservoir à la couche de sortie sont initialisées à zéro. Tous les tests présentés par la suite avec les ESNs ont été réalisés avec les mêmes matrices de connexions, aussi bien pour les connexions internes au réservoir que pour les connexions allant de la couche de sortie au réservoir. La valeur des connexions internes du réservoir ont été reprises du rapport technique [Jaeger, 2001]. Il est nécessaire que ces poids vérifient des propriétés très précises pour que la dynamique du réseau soit suffisament riche. Le réservoir est composé de 400 neurones. La matrice des poids comporte donc $400 * 400 = 1600$ poids. 98.75% des connexions ont un poids à 0, 0.625% des connexions ont un poids de 0.4 et 0.625% des connexions ont un poids de -0.4. La distribution de ces poids est faites aléatoirement et reste la même pour les phases d'apprentissages et de reproduction de séquences. La séquence étant un signal "analogique" - dont les états sont des activités variant entre 0 et 1 - la couche de sortie ne contient qu'un seul neurone. Par conséquent, entre la couche de sortie et le réservoir il y a 1×400 poids qui sont assignés aléatoirement avec des poids allant de -2 à 2.
Dans les ESNs, le réservoir de dynamiques joue le rôle de mémoire à court terme. On se rend compte de cette propriété en observant les activités des neurones du réservoir lorsque l'on applique un signal impulsionnel en entrée (figure 3.17).
De manière à rester cohérent avec le modèle d'apprentissage de séquences temporelles vu précédemment, les états "0", "1" et "2" sont transformés en signaux analogiques suivant la relation suivante :

$$etat_{esn} = \frac{etat_{sequencetemporelle}}{nbetats - 1} \tag{3.6}$$

avec $etats_{ens}$ la valeur du signal, $etat_{sequencetemporelle}$ l'état de la séquence et $nbetats$ le nombre d'états total pouvant composer une séquence. Ici $nbetats$ vaut 4. Par conséquent, ce qui était les états "0", "1" et "2" pour l'apprentissage de séquences temporelles devient "0", "0.333333"

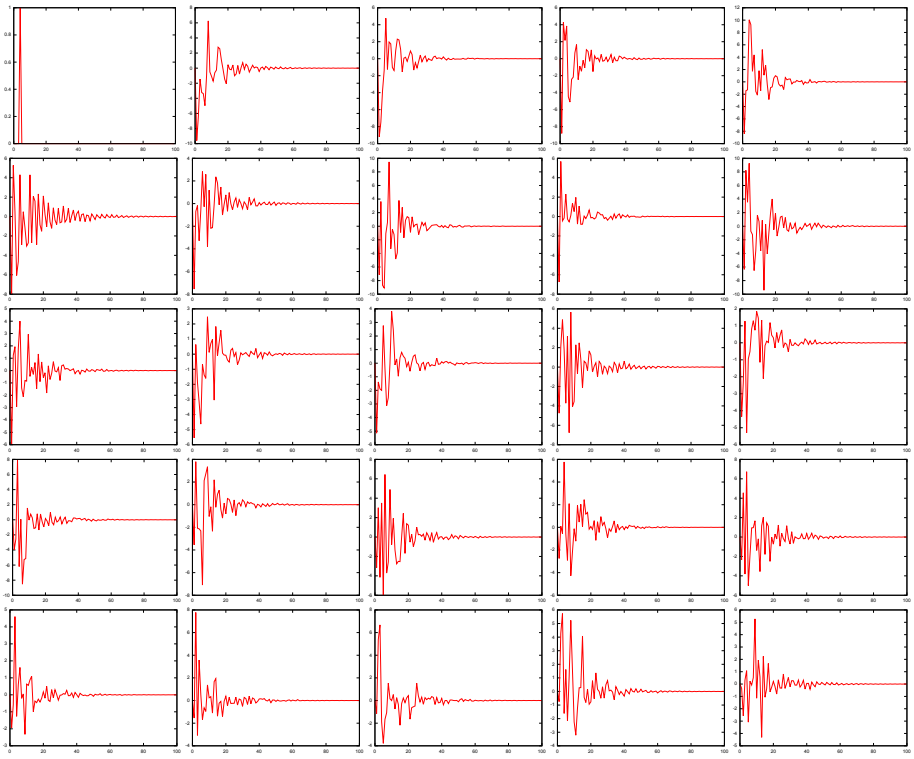


Fig. 3.17: Activités de 24 neurones (sur 400 neurones) choisis arbitrairement du réservoir de dynamiques. L'activité résulte d'un signal sous forme d'impulsion avec une activité de 1 (première courbe en haut à gauche).

et "0.666666" pour les ESNs. Par souci de lisibilité, je garderai dans cette section la notation adoptée dans la section précédente en notant les états "0", "1", etc.
Lors de la phase d'apprentissage, il est indispensable de répéter un nombre suffisant de fois la séquence que l'on souhaite faire apprendre. Ceci pour deux raisons : la première, pour que les dynamiques du réservoir aient le temps de converger vers un état stable. La seconde pour que l'apprentissage soit stable. Par exemple, pour une séquence simple de deux états comme "0 1" il faut environ 50 itérations avant que les dynamiques du réservoir se stabilisent sur un état stable (figure 3.18).
Durant la phase d'apprentissage de la séquence "0 1", les activités des neurones du réservoir sont enregistrées dans un fichier, ainsi que celles de la séquence à apprendre. Ensuite, le calcul des poids allant du réservoir à la couche de sortie est réalisé suivant l'équation 3.5 avec le programme *Scilab*. Pour réaliser l'apprentissage, les activités des 50 premières itérations ne sont pas prises en compte, car durant cette période les dynamiques du réservoir ne sont pas encore stables. Une fois le calcul terminé les poids sont chargés sur les connexions et la simulation est relancée. Durant la phase de reproduction, la séquence est présentée durant les 50 premières itérations. Ensuite le signal correspond aux activités que l'architecture calcule. On observe que la séquence est correctement reproduite. Malgré le fait que le signal ne soit plus forcé, l'architecture continue de le reproduire en sortie. (figure 3.19).
Comme le montre la figure 3.17, le réservoir est capable de maintenir une activité durant une courte période lorsqu'il n'y a plus d'activité présente en entrée. Cette propriété peut alors permettre d'apprendre un certain timing entre les états. Par exemple, avec une séquence du type "1 - 2" où "-" représente l'absence d'activité en entrée, les ESNs sont capable de restituer la séquence avec le même timing (figure 3.20).

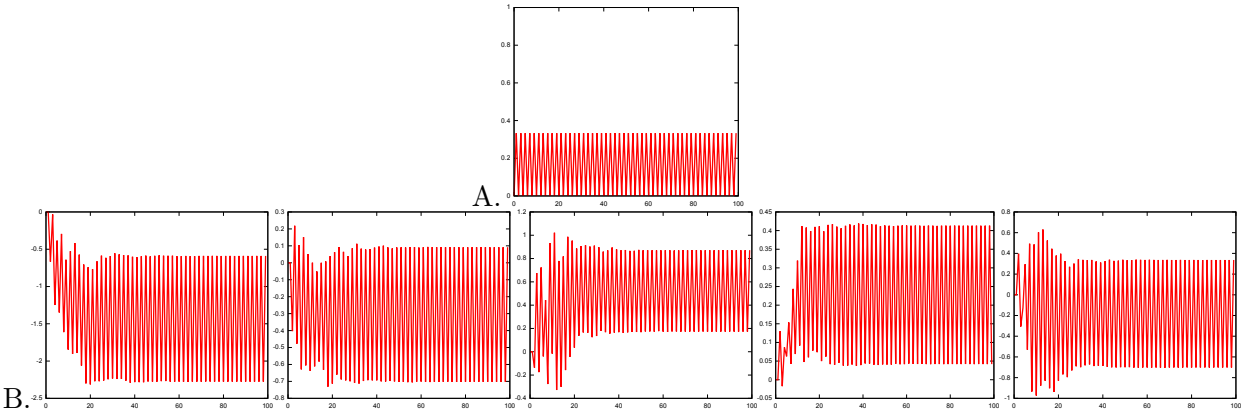


Fig. 3.18: A) Activité du signal donné à l'architecture. B) Activités de 5 neurones du réservoir de dynamiques choisis arbitrairement. On observe que les dynamiques mettent environ 50 itérations avant de converger vers un état stable.

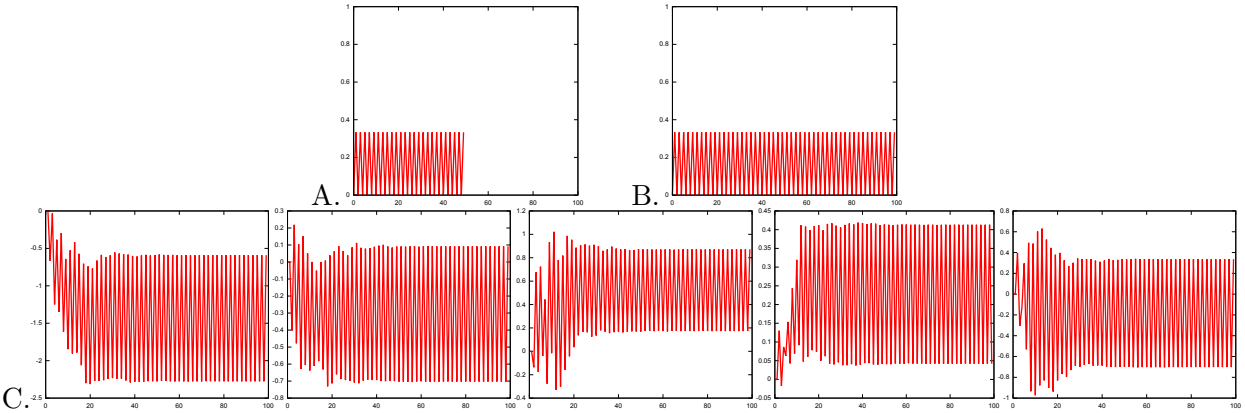


Fig. 3.19: A) Activité du signal donné à l'architecture durant les 50 premières itérations. B) Activité de sortie. Durant les 50 premières itérations, ce signal est forcé par le signal en A. Ensuite le signal correspond aux activités que l'architecture calcule. On observe que la séquence est correctement reproduite. C) Activités de 5 neurones du réservoir de dynamiques choisis arbitrairement. On observe que les dynamiques sont identiques à la phase d'apprentissage.

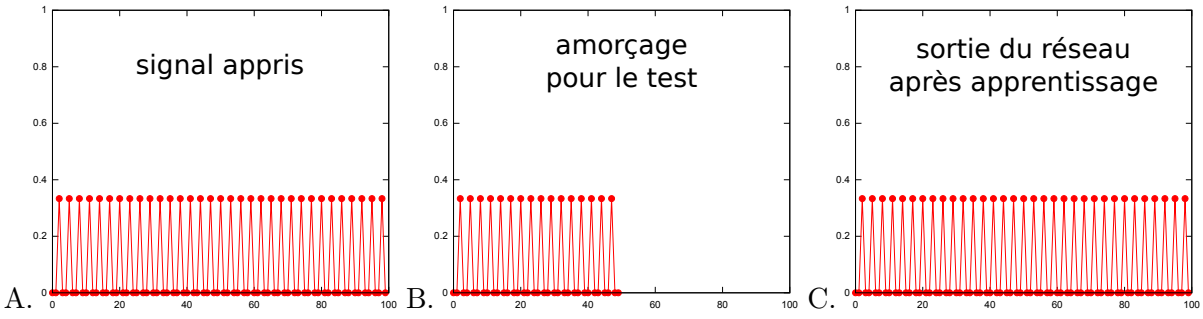


Fig. 3.20: A) Activité du signal fourni à l'architecture durant la phase d'apprentissage. Ce signal représente la séquence "0 - 1" B) Activité du signal fourni à l'architecture durant la phase de reproduction. Le signal est fourni durant les 50 premières itérations. C) Activité de sortie de l'architecture. Durant les 50 premières itérations, ce signal est forcé par le signal en B. Ensuite le signal correspond aux activités que l'architecture calcule. On observe que la séquence est correctement reproduite malgré que le timing soit plus long pour le premier état "0".

Si les ESNs permettent d'apprendre le timing des transitions, cette propriété reposant sur le temps de convergence vers un état stable est limitée. En effet, plus ce temps est long, plus les dynamiques du réservoir se seront dissipées jusqu'à la présentation d'un nouvel état. Comme vu précédemment sur la figure 3.17, les dynamiques du réservoir sont maintenues durant environ 50 itérations. Maintenant, si l'on considère une séquence composée de deux états "2" et "1" avec un intervalle de 50 itérations entre les deux états, alors on constate lors de la reproduction que le réseau restitue la séquence à partir de sa propre dynamique, si la séquence est ré-amorcée pendant 200 itérations. On observe aussi que le signal de sortie se déforme au fil du temps (figure 3.21).

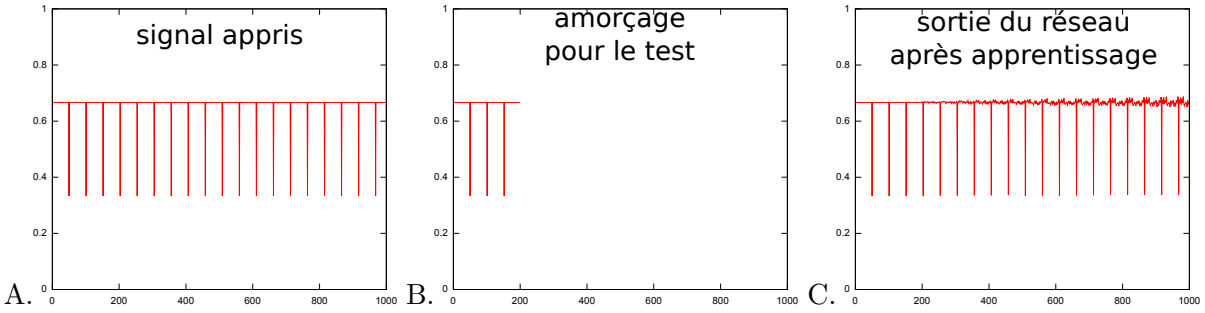


Fig. 3.21: A) Activité du signal fourni à l'architecture durant la phase d'apprentissage. Ce signal représente une séquence "2 - 1" avec un intervalle de 50 itérations pendant lesquelles l'état "2" est maintenu avant de présenter l'état "1" au réseau. B) Activité du signal fourni à l'architecture durant la phase de reproduction. Le signal est fourni durant les 200 premières itérations. C) Activité de sortie du réseau. Durant les 200 premières itérations, ce signal est forcé par le signal en B. Ensuite le signal correspond aux activités que le réseau calcule. On observe que la séquence n'est pas reproduite correctement ; elle se déforme au fil du temps.

Dans la section précédente, j'ai pu montrer que le modèle d'apprentissage de séquences temporelles simples permet de restituer la suite d'une séquence lorsqu'on fournit non pas le premier état de la séquence, mais un état au milieu de la séquence. D'une certaine manière, cette propriété est une resynchronisation extrêmement frustre permettant la restitution de la suite de la séquence. Cette propriété de resynchronisation est également présente dans les ESNs, mais pour des raisons différentes. Tout d'abord, pour que les ESNs apprennent une séquence, celle-ci doit être fourni non pas une seule fois, mais plusieurs fois (de manière répétitive). Par conséquent, du point de vue du réservoir, une fois les dynamiques stabilisées, il n'y a pas vraiment de début, de fin ou même de milieu d'une séquence. De plus, une fois que les dynamiques du réservoir ont convergé vers un état stable, elles n'en sortent plus tant que les activités d'entrée respectent la séquence.
Pour mettre en évidence cette propriété, nous considérons une séquence plus complexe que celles testées jusqu'ici. Le nombre d'états *nbetats* dans ce test est fixé à 6. En suivant la correspondance fournie par l'équation 3.6, les différentes activités "0", "0.2", "0.4", "0.6", "0,8" correspondent respectivement aux états "0", "1", "2", "3" et "4". Nous faisons apprendre la séquence "2 3 2 0 1 4 3 4 0", puis lors de la phase de reproduction, le signal est redonné, mais directement au milieu de la séquence au non pas par le premier état (figure 3.22). Une fois les dynamiques du réservoir stabilisées, le réseau restitue correctement la séquence qui a été apprise.
Comme le montre la séquence testée ici, les ESNs permettent l'apprentissage de séquences complexes ; c'est à dire des séquences dans lesquelles un même état peut apparaître plusieurs fois. En effet, dans la séquence "2 3 2 0 1 4 3 4 0" seul l'état "1" est unique. Cette propriété vient du fait que les dynamiques du réservoir ne dépendent pas seulement de l'activité présente en entrée à un instant donné, mais également des états précédent qui représentent l'historique de

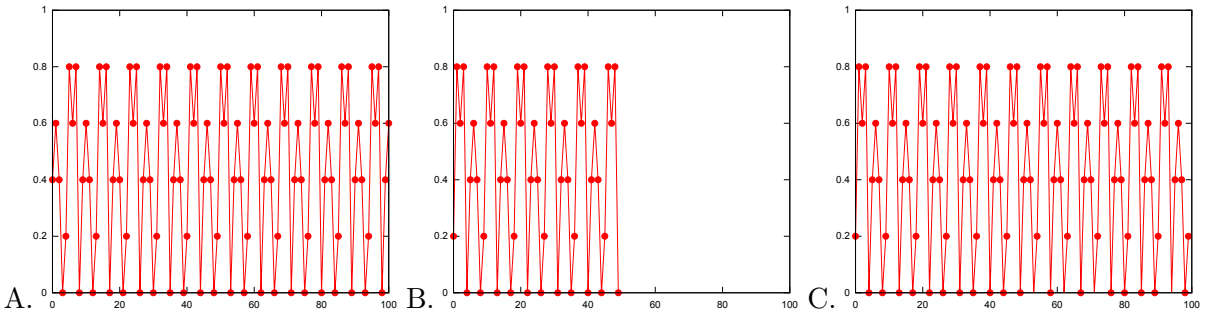


Fig. 3.22: A) Activité du signal fourni à l'architecture durant la phase d'apprentissage. Ce signal représente une séquence composée de neuf états "2 3 2 0 1 4 3 4 0". B) Activité du signal fourni à l'architecture durant la phase de reproduction. Le signal est fourni durant les 50 premières itérations. On remarque que le signal fourni est temporellement décalé par rapport à celui donné lors de la phase d'apprentissage en A. C) Activité de sortie de l'architecture. Durant les 50 premières itérations, ce signal est forcé par le signal en B. Ensuite le signal correspond aux activités que l'architecture calcul. On observe que la séquence est correctement reproduite.

la séquence.
Comme je l'ai montré jusqu'ici, les ESNs ont des propriétés intéressantes qui permettent d'apprendre des séquences temporelles complexes. Mais dans le contexte d'apprentissage de comportements sur des robots que l'on souhaite autonomes et interactifs, l'apprentissage hors ligne ne convient pas. Alors, comment apprendre en ligne avec des ESNs ? Pour l'apprentissage hors ligne donné par [Jaeger, 2001], l'algorithme utilisé réalise une régression linéaire. L'objectif de l'apprentissage est de trouver les poids permettant de corréler l'état des dynamiques du réservoir avec la sortie désirée - l'état de la séquence - à chaque instant. Ceci revient à trouver les invariants des dynamiques du réservoir pour chacun des états d'une séquence. J'ai alors utilisée une descente de gradient stochastique minimisant l'erreur quadratique moyenne (LMS - Least Mean Square) [Widrow et Hoff, 1960] qui permet de réaliser cet apprentissage en ligne. L'utilisation de cette règle d'apprentissage avec les ESNs a déjà été discutée rendant compte que l'apprentissage pouvait ne pas converger ou en tout cas mettre énormément de temps [Lukosevicius et Jaeger, 2009]. Néanmoins les auteurs montrent que l'apprentissage en ligne avec une règle de type *Recursive Least Square (RLS)* converge plus rapidement mais avec un coup calculatoire plus important. Toutefois, un apprentissage de type *LMS* reste possible et est utilisé dans les tests présentés dans cette section.
L'activité Act_s du neurone s de la couche de sortie est calculée suivant l'équation :

$$Pot_s = \sum_{r=1}^{nbres} Act_r.w_{rs} \tag{3.7}$$

$$Act_s = f(Pot_s) \tag{3.8}$$

avec $nbres$ le nombre de neurones dans le réservoir, Act_r l'activité du neurone r du réservoir et w_{rs} le poids de la connexion entre le neurone r du réservoir et le neurone s de la couche de sortie. La fonction f est une fonction lineaire entre 0 et 1. Les poids des connexions modifiables sont calculés suivant :

$$\Delta w_{rs} = \varepsilon(Pot_s.(Sd_s - Pot_s)) \tag{3.9}$$

ε la vitesse d'apprentissage et Sd_s la sortie désirée sur le neurone s de la couche de sortie.
Etant donné que l'apprentissage est en ligne, il est nécessaire de modifier l'architecture de manière à ce que le réseau de neurone n'apprenne pas d'informations erronées. En effet, jusqu'ici, les activités fournies étaient imposées à la couche de sortie, puis grâce aux connexions de retour

vers le réservoir, ces activités alimentaient les dynamiques. Dans le cas de l'apprentissage en ligne, la couche de sortie ne fournit pas directement les états de la séquence au réservoir. J'ai séparé le signal correspondant à la séquence que je désire faire apprendre et le signal qui alimente les dynamiques du réservoir (figure 3.23). En phase de reproduction, le réservoir est d'abord alimenté par la couche d'entrée en attendant que les dynamiques aient convergées vers un état stable, puis le réservoir est alimenté par la couche de sortie par l'intermédiaire des connexions de retour vers la couche d'entrée. De plus, les couches d'entrée et de sortie ne comportent plus un unique neurone, mais chacune un vecteur de neurones, car les états d'une séquence ne sont plus les activités du neurone d'entrée, mais l'activation ou non de neurones dans un vecteur de neurones. Enfin, de manière à stopper l'apprentissage de la couche de sortie lorsqu'il n'y a plus de signal imposé, une connexion de neuromodulation est ajoutée : si cette neuromodulation est à 1, alors la couche de sortie apprend, si elle est à 0 alors l'apprentissage est arrêté.

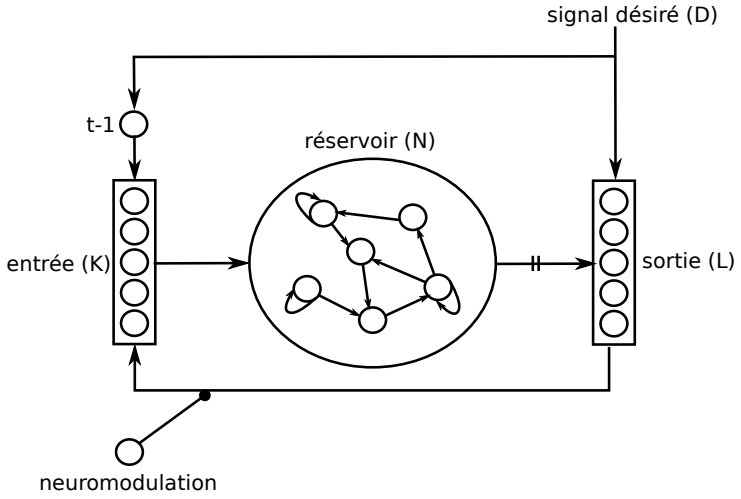


Fig. 3.23: Modèle modifié des ESNs qui permet l'apprentissage en ligne de séquences temporelles complexes. L'apprentissage est réalisé sur les connexions entre le réservoir et la couche de sortie suivant la règle *LMS*.

Pour ce test, je reprend l'exemple de la séquence complexe "2 3 2 0 1 4 3 4 0" où chaque état est l'index du neurone dans les vecteurs des couches d'entrée et de sortie. Comme l'apprentissage est réalisé durant 1500 itérations, la connexion de retour entre la couche de sortie et la couche d'entrée est complètement inhibée par la neuromodulation. Après l'itération 1500, la séquence n'est plus fournie par le professeur et la neuromodulation tombe à 0. Par conséquent le système calcule les états prédits en sortie uniquement à partir des dynamiques du réservoir.
S'il est possible d'apprendre en ligne avec un ESN, ce n'est pas sans contrepartie. En effet, pour que l'apprentissage se fasse correctement, la vitesse d'apprentissage doit être très basse. Dans l'exemple testé ici, la vitesse d'apprentissage $\varepsilon = 0.0005$ dans l'équation 3.9. Il faut environ 800 itérations (figure 3.25) de la phase d'apprentissage pour que les poids des connexions modifiables entre le réservoir et la couche de sortie se stabilisent. Par conséquent, comme la séquence est composée de 8 éléments, il faut donc la présenter environ 100 fois avant qu'elle soit correctement apprise.
En conclusion, les ESNs permettent d'apprendre des séquences temporelles complexes. Le réservoir de dynamiques apporte des propriétés intéressantes qui permettent de maintenir en mémoire un état (mémoire à court terme), ainsi que la possibilité de resynchroniser sur une séquence qui ne débute pas son premier état. Néanmoins, dans le contexte d'apprentissage de comportement

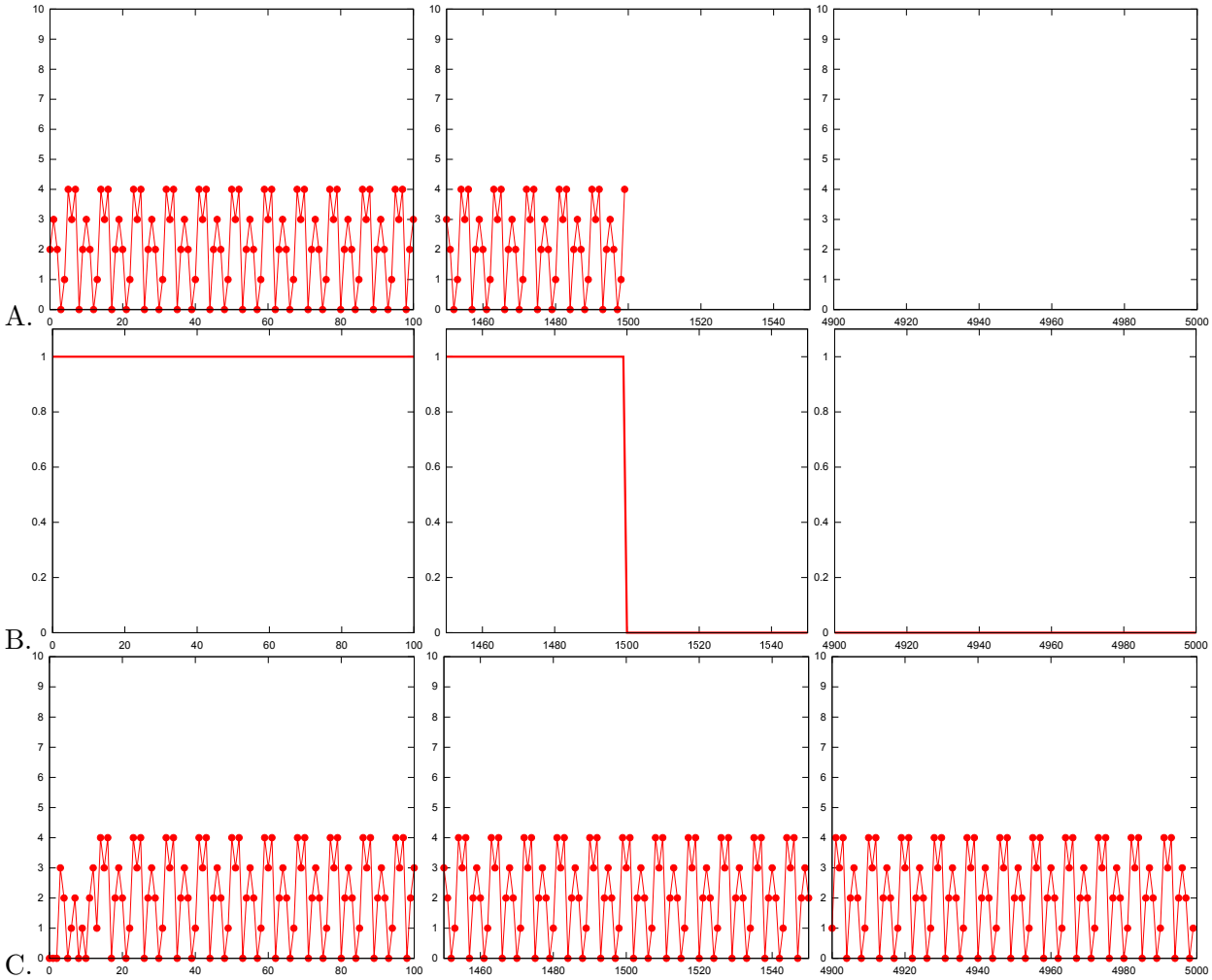


Fig. 3.24: A) Séquence "2 3 2 0 1 4 3 4 0" que l'on désire faire apprendre à l'architecture. A partir de l'itération 1500, la séquence n'est plus fournie par le professeur. L'architecture calcule les états de sortie à partir des dynamiques du réservoir. B) Activité de la neuromodulation qui permet d'inhiber les connexions de retour durant l'apprentissage lorsque son activité est à 1. A partir de l'itération 1500, l'activité tombe à 0. Par conséquent, les liens de retour ne sont plus inhibés et l'apprentissage est stoppé. C) Activité de sortie de l'architecture. Jusqu'à l'itération 1500 les états actifs sont ceux fournis par le professeur. A partir de l'itération 1500, les états sont ceux qui sont prédis. On observe alors que la séquence est correctement reproduite, même au bout de 5000 itérations.

sur des robots, le modèle souffre de quelques faiblesses. La première est qu'il est nécessaire que la séquence soit compacte ; c'est-à-dire que le temps entre deux états doit rester très court. De plus, dans le cas de l'apprentissage en ligne, le temps d'apprentissage est très important et implique de répéter un grand nombre de fois la même séquence en boucle. Dans le cadre d'un apprentissage avec un humain, ce nombre de répétitions peut être fastidieux. De la même manière, les ESNs restituent la séquence en boucle et pas seulement une seule fois. De plus, il est assez compliqué de déterminer les poids des connexions du réservoir.

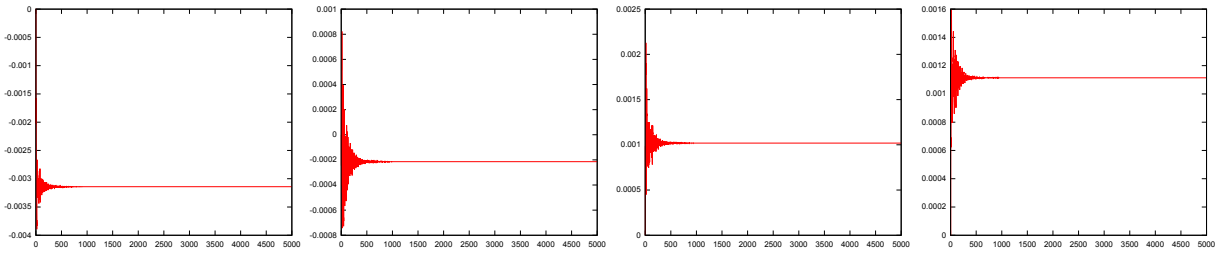


Fig. 3.25: Evolution des poids de quatre connexions entre le réservoir et la couche de sortie choisis arbitrairement. On observe qu'il faut environ 800 itérations avant que l'apprentissage se stabilise. La séquence étant composée de 8 éléments, cela signifie qu'il faut présenter environ 100 fois la séquence avant qu'elle soit effectivement apprise.

3.3 Modèle d'apprentissage de séquences temporelles complexes

En partant du modèle d'apprentissage de séquences temporelles simples (section 3.1.4.1), la question est : comment apprendre une séquence complexe ? Dans la section précédente, j'ai mis en évidence qu'un réservoir de dynamiques offrait entre autres la propriété d'apprendre des séquences complexes. En effet, les dynamiques du réservoir ne codent pas uniquement pour l'élément en entrée, mais pour l'élément et ceux le précédent. Dit d'une autre manière, c'est le moment où arrive l'élément qui permet de le différencier des autres éléments de la séquence. Cette idée de contexte n'est pas nouvelle en soit. Dans [Cohen *et al.*, 1990], les auteurs discutent d'apprentissages hiérarchiques permettant d'apprendre des séquences complexes. Un premier apprentissage consiste à associer un élément à celui qui l'a précédé ne permettant d'apprendre que des séquences simples. Pour une séquence complexe, les auteurs utilisent une hiérarchie de séquences ; c'est-à-dire une représentation hiérarchique d'une séquence permettant de supprimer les ambiguïtés par l'introduction de nouveaux états.
Dans [Elman, 1990], l'auteur reprend un modèle appelé *SRN* pour *Simple Recurrent Network* [Servan-Schreiber *et al.*, 1989] qui permet d'apprendre des séquences (figure 3.26). Le modèle est un réseau à retro-propagation composé de trois couches : la couche d'entrée, la couche cachée qui encode les entrées en états internes et la couche de sortie. L'apprentissage est réalisé sur toutes les connexions *feedforward*. Dans ces travaux, les auteurs ajoutent une recopie de la couche cachée comme contexte qui avec un délai reboucle sur la couche cachée. De cette manière, chaque élément qui compose une séquence est associé avec celui qui l'a précédé. Ici, le contexte est donc une mémoire à court terme de l'état précédent. Avec un modèle simple comme le *SRN*, on se rend déjà compte de la nécessité d'une mémoire à court terme, mais aussi d'un contexte qui permet de différencier les éléments d'une séquence. De plus, la richesse de ce contexte joue un rôle important dans la capacité à différencier correctement les différents éléments d'une séquence. Dans [Maskara et Noetzel, 1993], les auteurs enrichissent le contexte d'un *SRN* avec une mémoire auto-associative *RAAM* (Recursive Auto-Associative Memory) [Pollack, 1990].
Néanmoins le modèle *SRT* ne permet pas d'apprendre le timing entre les éléments qui composent une séquence. Dans [Dominey et Ramus, 2000], les auteurs proposent un modèle de réseau de neurones récurrent temporel *TRN* (*Temporal Recurrent Network*) (figure 3.27) inspiré du système corticostriatal du cerveau des primates [Dominey *et al.*, 1995]. Ce modèle est proche du *SRN*, mais la principale différence est qu'il permet de ne pas perdre le timing entre les états d'une séquence. Cette propriété vient principalement de l'utilisation de neurones intégrateurs à fuite (*leaky integrator*) à temps continu. De plus, contrairement au modèle *SRN*, l'apprentissage est réalisé uniquement sur les connexions entre les états internes et la couche de sortie. Ce modèle

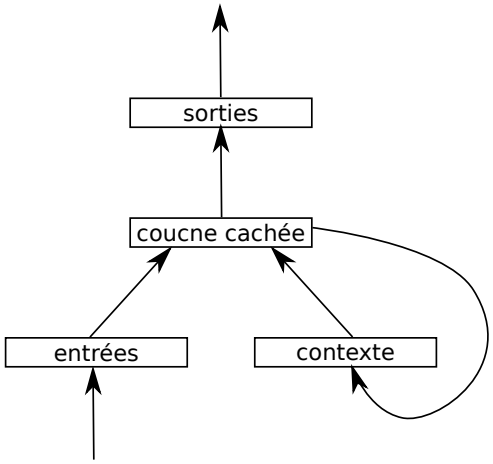


Fig. 3.26: Le modèle *SRN* permet d'apprendre des séquences complexes. Ce modèle apporte un contexte permettant de différencier les éléments d'une séquence ambigüe [Servan-Schreiber *et al.*, 1989].

à permis d'apprendre différentes séquences de comportement de navigation ou de gestes sur différents robots comme *Aibo*[1], *HRP2*[2], *Khepera*[3] et *Lynx*[4]

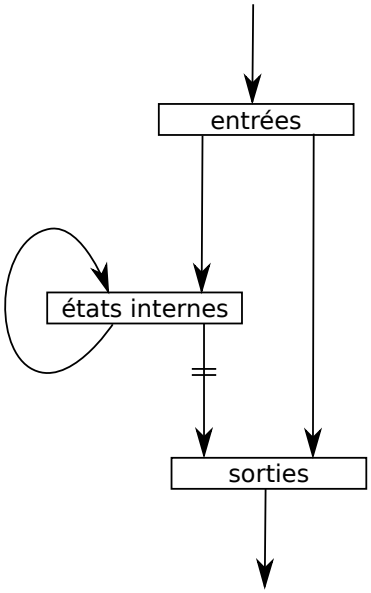


Fig. 3.27: Le modèle *TRN* permet d'apprendre des séquences complexes tout en respectant le timing entres les différents éléments d'une séquence. Comme le modèle *SRN*, ce modèle apporte un contexte mais dynamique permettant de différencier les éléments d'une séquence ambigüe grâce à la connexion récurrente sur les états internes.

Dans [Dominey, 2005], l'auteur ajoute au modèle *TRN* un réseau *Abstract Temporal Recurent*

[1]Robot chien de Sony

[2]Robot humanoïde japonais developpé par le groupe Humanoid Research Group de l'Intelligent Systems Research Institute appartenant à l'AIST (National Institute of Advanced Industrial Science and Technology)

[3]Robot mobile de K-Team

[4]Bras robotique à base de servo moteurs

Network (*ATRN*) entre la couche d'entrée et celle de sortie permettant d'apprendre des règles sémantiques à partir des séquences apprises. Cette nouvelle partie du modèle mémorise un nombre prédéfinis d'élément d'une séquence qui sont ensuite comparé aux éléments courants. En reprenant l'exemple donné par l'auteur, avec une mémoire de trois élément, une séquence ABCBAC sera représentée par "u u u -2 -4 -3" avec "u" représentant que l'état n'a pas était rencontré précédemment et "-N" dénotant que l'élément a été rencontré N éléments plus tôt. Ce mécanisme permet alors au modèle de s'abstraire du contenu des séquences. Si ce mécanisme s'applique bien aux problèmes du langage, il peut être également pertinent pour des séquences motrices. Cependant, appliqué directement à mes travaux, l'ajout d'un tel mécanisme paraît peu pertinent, car les éléments des séquences sont des orientations.
Ces travaux montrent que l'ajout d'un contexte permet de supprimer l'ambiguïté des états que l'on retrouve plusieurs fois dans une séquence. Lorsque le contexte est dynamique, il permet d'être représentatif d'un historique plus riche (dans le sens ou cet historique n'est pas uniquement l'entrée précédente) de la séquence. Cela permet de différencier l'élément courant de la séquence aux précédents. D'une certaine manière, ce contexte codant pour l'historique de la séquence permet au système de se "localiser" dans la séquence et d'une certaine manière dans le temps. L'ajout d'information permet alors la transformation des états cachés en états observables.

3.3.1 Le contexte interne

Les réseaux de neurones récurrents à temps continu (CTRNN : *Continous Time Recurrent Neural Network*) sont une généralisation de réseaux de neurones de Hopfield Continue [Hopfield, 1984]. [Beer, 1994] fait l'étude de la dynamique d'un neurone de Hopfield en ajoutant pour chaque neurone, une connexion à lui même. Il permet de faire un rapprochement entre le comportement de ce modèle de neurone et celui des neurones biologiques. Contrairement à des systèmes à temps discret [Daucé *et al.*, 2002], l'utilisation de neurones à temps continu permet à chaque instant de tenir compte du temps écoulé entre deux calculs des activités. Par conséquent, le contexte interne est une référence temporelle avec laquelle les états d'une séquence temporelle peuvent être associés. Un CTRNN couplant deux neurones produit un oscillateur (figure 3.28) suivant le système d'équations différentielles :

$$\tau_e.\frac{dx}{dt} = -x + S((w_{ii} * x) - (w_{ji} * y) + w_{e_{const}}) \quad (3.10)$$

$$\tau_i.\frac{dy}{dt} = -y + S((w_{jj} * y) + (w_{ij} * x) + w_{i_{const}}) \quad (3.11)$$

avec τ_e une constante de temps du neurone excitateur et τ_i pour le neurone inhibiteur. x et y sont les activités respectives des neurones excitateur et inhibiteur. w_{ii} est le poids de la connexion récurrente du neurone excitateur et w_{jj} le poids de la connexion récurrent du neurone inhibiteur. w_{ij} est le poids de la connexion partant du neurone excitateur vers le neurone inhibiteur et w_{ji} le poids de la connexion dans le sens inverse. $w_{e_{const}}$ et $w_{i_{const}}$ sont les poids de connexions arrivant d'entrées constantes. S est la fonction de transfert de chacun des neurones. Dans mon modèle c'est la fonction identité. De plus, j'utilise des constantes de temps égales pour les neurones d'un même oscillateur : $\tau_e = \tau_i$.
Ce modèle d'oscillateur neuronal est reconnu pour sa stabilité, sa robustesse et sa résistance à de potentielles perturbations. Il a également l'avantage d'être facile à implémenter.
Dans la perspective de permettre l'apprentissage de séquences complexes, j'ai réalisé une étude sur la stabilité des oscillateurs. La figure 3.29.A illustre l'activité des neurones d'un oscillateur. La figure 3.29.B montre l'activité de chacun des deux neurones (excitateur et inhibiteur) composant

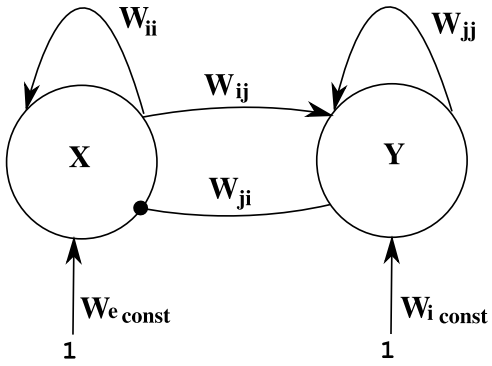
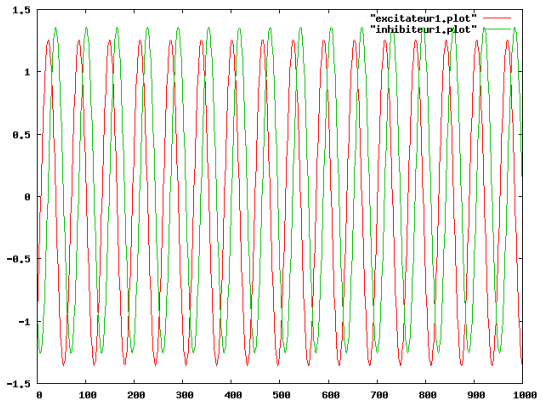


Fig. 3.28: Oscillateur à 2 neurones de type CTRNN. Le neurone de gauche est le neurone excitateur, celui de droite est inhibiteur. Les connexions terminant par une flèche sont des connexions excitatrices. La connexion se terminant par un rond est une connexion inhibitrice.

l'oscillateur. On remarque qu'en modifiant l'activité sur les entrées $u1$ et $u2$, une augmentation de l'amplitude des oscillations sur les 600 premiers pas de temps, puis un écrasement de l'amplitude par la suite.

A. 

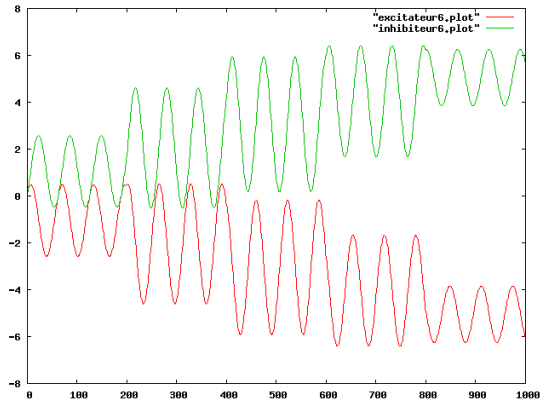
B. 

Fig. 3.29: à gauche : Illustration de l'activité des neurones excitateurs et inhibiteur d'un oscillateur. Au départ, $X = 1$ et $Y = 1$, $u1 = 0$ et $u2 = 0$ et la constante de temps $h = 0.1$ A droite : Illustration de l'activité des deux neurones (excitateur et inhibiteur) d'un oscillateur. Au départ $X = 0$ et $Y = 0$. La constante de temps h = 0.1. Les entrées $u1 = 1$ et $u2 = 1$. Ces 2 entrées sont chacune incrémentées de 1 tous les 200 pas de temps. C'est le changement d'activité sur les entrées $u1$ et $u2$ qui modifie le comportement des oscillations.

3.3.2 Apprentissage d'états internes

De manière à pouvoir dissocier la répétition d'un même état d'entrée dans une séquence, chacun des états est associé à la dynamique courante générée par les oscillateurs. Pour mieux comprendre le rôle de la dynamique interne et son implication dans l'algorithme d'apprentissage, j'ai testé deux mécanismes qui permettent la création d'états internes. Le premier est un mécanisme simple de compétition entre les états d'entrée d'une séquence couplés avec les oscillateurs. Le second mécanisme est basé sur une règle d'apprentissage qui recrute les états internes en fonction de l'activité générée par les oscillateurs et les entrées de la séquence.

3.3.2.1 Mécanisme de compétition

La création d'états internes réalisée par le mécanisme de compétition (figure 3.30) se déroule comme suit : chaque neurone ij du groupe de compétition agit comme un neurone réalisant une opération de type ET logique (neurone à seuil élevé ou neurone de type *Sigma-Pi* [Durbin et Rumelhart, 1989]) entre les neurones du groupe d'entrée et les neurones des oscillateurs :

$$Pot_{ij} = (w_{input_i} * x_{input_i} + w_{osci_j} * x_{osci_j}) - threshold_{ij} \tag{3.12}$$

avec $w_{input_i} = 1$, $w_{osci_j} = 1$, $threshold_{ij} = 1.2$, x_{input_i} l'activité du neurone d'entrée à l'index i et x_{osci_j} l'activité de l'oscillateur à l'index j.
Dans une seconde étape, tous les neurones ij du groupe Competition sont mis en compétition :

$$Winner_{ij} = \begin{cases} 1 & si\ ij = Argmax_{ij}(Pot_{ij}) \\ 0 & sinon \end{cases} \tag{3.13}$$

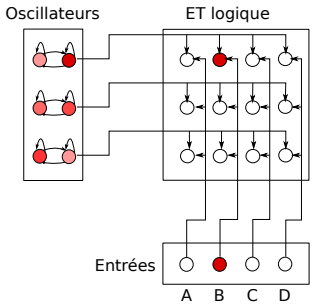


Fig. 3.30: Modèle du réseau de neurones qui couple chacune des entrées avec un oscillateur. Toutes les connexions ont des poids fixent.

Le neurone gagnant devient alors l'entrée du réseau d'apprentissage de séquences temporelles (section 3.1.4.1). De cette manière, un réservoir d'oscillateurs peut être utilisé pour associer le même état avec différents états de la dynamique des oscillateurs. Intuitivement, le mécanisme simple de la compétition (ne demandant aucun apprentissage) permet de sélectionner directement les états internes correspondant à une même entrée répétée plusieurs fois dans une séquence. Par exemple dans la figure 3.30, chacune des entrées (A, B, C, D) peut apparaître jusqu'à trois fois dans la même séquence en fonction des activités des oscillateurs. De plus, un tel mécanisme ne perturbe ni la prédiction ni la reproduction de la séquence.
Néanmoins, il est encore possible d'avoir des séquences ambigüe. Une entrée peut être associée avec le même oscillateur plusieurs fois. Par conséquent, il y a potentiellement encore des ambigüités avec les états internes du modèle, et certaines séquences ne peuvent par conséquent ne pas être restituées correctement. Pour mesurer précisément ce problème, on peut calculer la probabilité pour qu'un même état soit associé avec le même oscillateur plusieurs fois. La capacité à discriminer des répétitions en entrée dépend du nombre d'oscillateurs et de leur phase. Pour éviter un nombre très important de neurones codant les conjonctions potentielles, on peut remplacer la matrice pleine par un système de recrutement de neurones.

3.3.2.2 Mécanisme de recrutement associatif (création d'états internes)

Le processus d'apprentissage d'une association (figure 3.31) entre un état en entrée et une configuration d'oscillateurs (figure 3.32) se déroule en différentes étapes. Dans ce chapitre, nous supposons un départ synchrone entre les entrées et les oscillateurs. Nous verrons dans le chapitre 4 le détail de ce mécanisme.

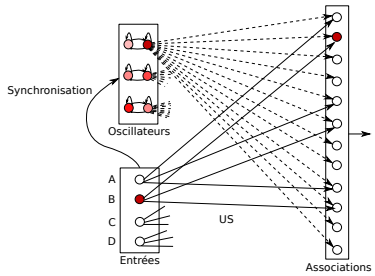


Fig. 3.31: Modèle du réseau de neurones utilisé pour associer chaque entrées avec la configuration des oscillateurs. Pour des raisons de lisibilité, tous les liens ne sont pas représentés. Les connexions sur lesquelles est réalisé l'apprentissage sont en pointillées. Les poids des connexions en trait plein sont fixes.

Tout d'abord, le potentiel de l'entrée inconditionnelle est calculé comme suit :

$$US = w_i * x_i \tag{3.14}$$

avec w_i le poids du lien inconditionnel venant du neurone à l'index i du groupe d'entée et x_i son activité. Si le potentiel est supérieur à un seuil (choisi arbitrairement), alors nous calculons le potentiel et l'activité arrivant des oscillateurs :

$$Pot_j = \sum_{j=1}^{M_{osci}} |(w_j - e_j)| \qquad Act_j = \frac{1}{1 + Pot_j} \tag{3.15}$$

avec M_{osci} le nombre d'oscillateurs, w_j le poids de la connexion venant de l'oscillateur d'index j (initialement très haut), et e_j l'activité de l'oscillateur j. Le neurone qui a l'activité la plus basse ; c'est à dire la plus éloignée de la configuration des oscillateurs, est recruté :

$$Win = Argmin_j(Act_j) \tag{3.16}$$

La configuration des oscillateurs est apprise en fonction de l'erreur de distance :

$$\Delta w_j = \varepsilon(e_j - w_j) \times Win \tag{3.17}$$

avec ε la vitesse d'apprentissage, w_j le poids de la connexion venant de l'oscillateur d'index j et e_j l'activité de l'oscillateur j.

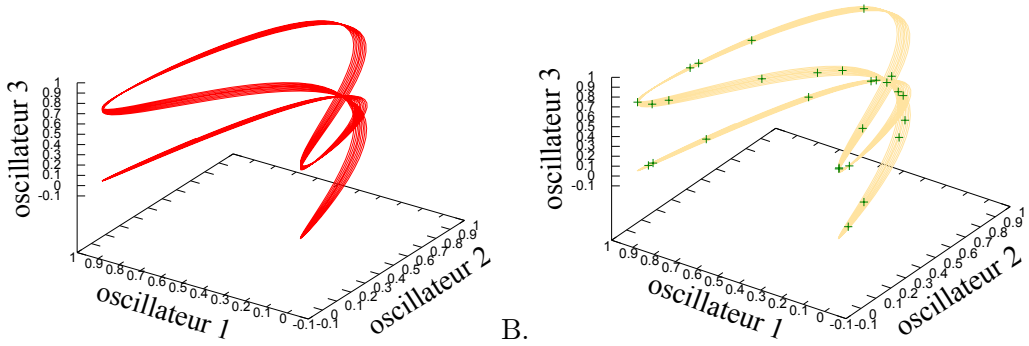


Fig. 3.32: A) Dynamique générée par 3 oscillateurs. B) Associations des états d'une séquence à différent instants de la dynamique.

Le groupe associatif codant les états internes devient alors l'entrée du réseau d'apprentissage de séquence temporelle. Comme le montre la figure 3.31, une entrée permet de recruter trois neurones différents. Cette quantité correspond au choix de la connectivité des liens inconditionnels entre le groupe d'entrée et le groupe associatif.
Le mécanisme associatif assure de recruter un nouvel état interne à chaque état d'une séquence (A, B, C ou D). La connectivité entre le groupe d'entrée et le groupe associatif a été choisie de manière a ce que pour chaque état du groupe d'entrée il y ait le même nombre d'états internes possibles que le mécanisme de compétition. Il est possible de changer la connectivité de ces liens pour permettre d'avoir plus d'états internes et éviter des erreurs lorsqu'un état est répété un grand nombre de fois dans une même séquence.
J'ai testé les mécanismes de compétition et de recrutement associatif dans une architecture permettant d'apprendre des séquences en simulation et dans une application robotique.

3.3.3 Simulations avec l'apprentissage de séquences temporelles complexes

Dans le cadre de l'apprentissage de comportements sur des robots, une séquence temporelle motrice est rarement rejouée deux fois avec le même rythme. Le temps entre deux états peut varier, particulièrement quand la séquence est apprise par démonstration à un robot. Dans les simulations présentées ici, nous introduisons des variations de temps entre deux états des séquences testées afin d'observer le comportement des trois architectures. La première architecture testée utilise l'apprentissage de séquences temporelles simples présentée en section 3.1.4.1. La seconde architecture est la même que la première à laquelle j'ai ajouté le mécanisme de compétition vue en section 3.3.2.1. Enfin, dans la troisième architecture le mécanisme d'association vue en section 3.3.2.2 a été ajouté.
Un premier jeu de séquences est généré aléatoirement, les états ainsi que le timing entre chaque états sont choisis aléatoirement, mais de manière à ce qu'elles soient correctement reproduites avec la seconde architecture (avec le mécanisme de compétition). Un second jeu de séquences correspond au premier, mais en ajoutant aléatoirement une variation sur le timing entre les états pouvant aller jusqu'à 5% du timing original. Un troisième jeu est généré de la même manière avec une variation pouvant aller jusqu'à 10%. Cette variation du timing sur les états est appliquée a priori lors de la génération des jeux de séquences. Les trois architectures sont entraînées avec ces trois mêmes jeux de séquences. Dans ces tests, pour amorcer une séquence, le neurone correspondant au premier état de la séquence est activé. Par conséquent, cet état ne sera pas ambigu dans la séquence. Par exemple, dans une séquence complexes comme "D B C B A C A B", l'état "D" permet d'amorcer la séquence et ne sera pas répété par la suite.
La figure 3.33 montre les performances de chacune des trois architectures testées. On remarque que l'apprentissage de séquences simples a de bonnes performances avec des séquences composées de 3 et 4 états. Ceci s'explique par le fait que dans des séquences de ces tailles il n'y pas d'état répétés plusieurs fois (séquences simples). Avec des séquences de tailles supérieures, les performances chutent drastiquement, car il y a au moins un état répété dans ces séquences. On peut également observer que la variation du timing entre les états n'a pas d'effet sur les performances de l'architecture.
La seconde architecture qui utilise le mécanisme de compétition a de meilleures performances. Malgré tout, il peut arriver qu'un même état interne soit répété plusieurs fois. Dans ce cas, l'ambigüité de la séquence n'est pas levée et l'architecture ne reproduit pas correctement la séquence. On remarque que les performances diffèrent selon la variation appliquée sur le timing entre deux états de la séquence. Ceci est expliqué par le fait que la configuration des oscillateurs est différente lorsqu'un état en entrée est présenté à un instant différent. Par conséquent, lorsqu'un

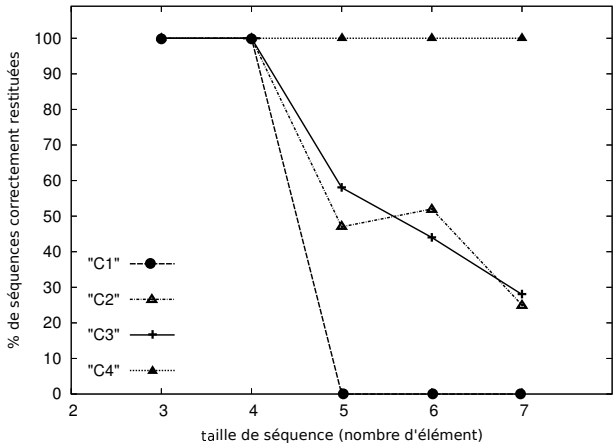


Fig. 3.33: C1 : première architecture : apprentissage de séquences simples. Les courbes des trois jeux de séquences sont superposées, car l'architecture a les mêmes performances. C2 : seconde architecture : apprentissage de séquences complexes avec le mécanisme de compétition. La courbe montre les performances de l'architecture avec le second jeu de séquences (variation du timing entre les états pouvant aller jusqu'à 5%). C3 : seconde architecture : apprentissage de séquences complexes avec le mécanisme de compétition. La courbe montre les performances de l'architecture avec le troisième jeu de séquences (variation du timing entre les états pouvant aller jusqu'à 10%). C4 : troisième architecture : apprentissage de séquences complexes avec le mécanisme associatif. Les courbes des trois jeux de séquences sont superposées, car l'architecture à les mêmes performances.

même état est présenté avec un timing différent, il sera associé avec deux oscillateurs différents et donc ce n'est pas le même état interne qui gagnera.
Grâce au mécanisme de recrutement, la troisième architecture a de meilleures performances avec 100% des séquences qui ont été testées. La variation appliquée sur le timing entre les états des séquences n'a pas d'effet sur les performances de l'architecture.

3.3.4 Application robotique à l'apprentissage de séquences temporelles

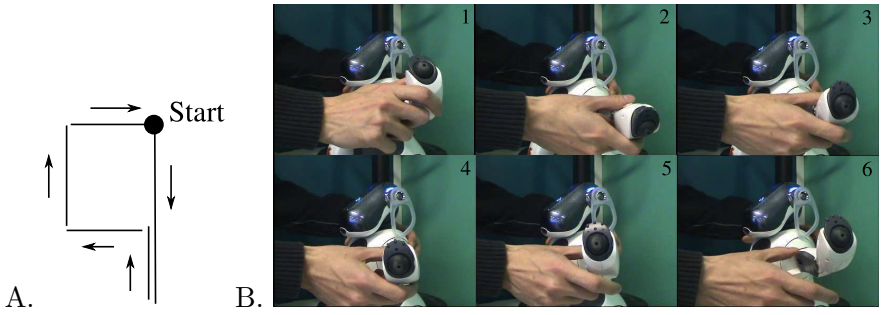


Fig. 3.34: A) Illustration de la séquence désirée débutant à partir du point. B) La patte du robot Aibo est manipulée passivement. Le robot apprend alors la succession des orientations à partir des informations motrices de sa patte avant gauche.

Le robot utilisé est un Aibo ERS7[5]. Dans cette application, la patte avant gauche du robot est utilisée. La séquence de gestes est apprise au robot en manipulant passivement la patte (figure 3.34.B). La figure 3.34.A montre la séquence que j'ai apprise au robot. Dans cette application seule l'architecture de séquences temporelles complexes avec le mécanisme associatif est utilisée.

[5]Robot chien de Sony

Durant la phase d'apprentissage, la patte avant gauche du robot est manipulée passivement. Les figures 3.35.X-*apprentissage* et 3.35.Y-*apprentissage* sont les enregistrements durant l'apprentissage des positions motrices de chacun des deux moteurs de la patte du robot (un moteur pour les mouvements horizontaux et un moteur pour les mouvements verticaux). Durant la démonstration du mouvement, le réseau de neurone apprend en ligne et en un coup la succession des orientations de l'extrémité de sa patte grâce aux informations motrices de sa patte (proprioception). Donc, les entrées du modèle sont les orientations de la patte.
Pour amorcer la reproduction de la séquence par le robot, le premier état de la séquence ("bas") est donné au système. Comme Aibo ne peut pas être manipulé une fois ses moteur actifs, cet état est envoyé directement au robot. Une fois la séquence amorcée, le robot rejoue seul la séquence (figure 3.35, en haut). Avec cet état d'amorçage, le modèle prédit le prochain état ; c'est-à-dire la prochaine orientation et envoie la commande motrice correspondante au robot.

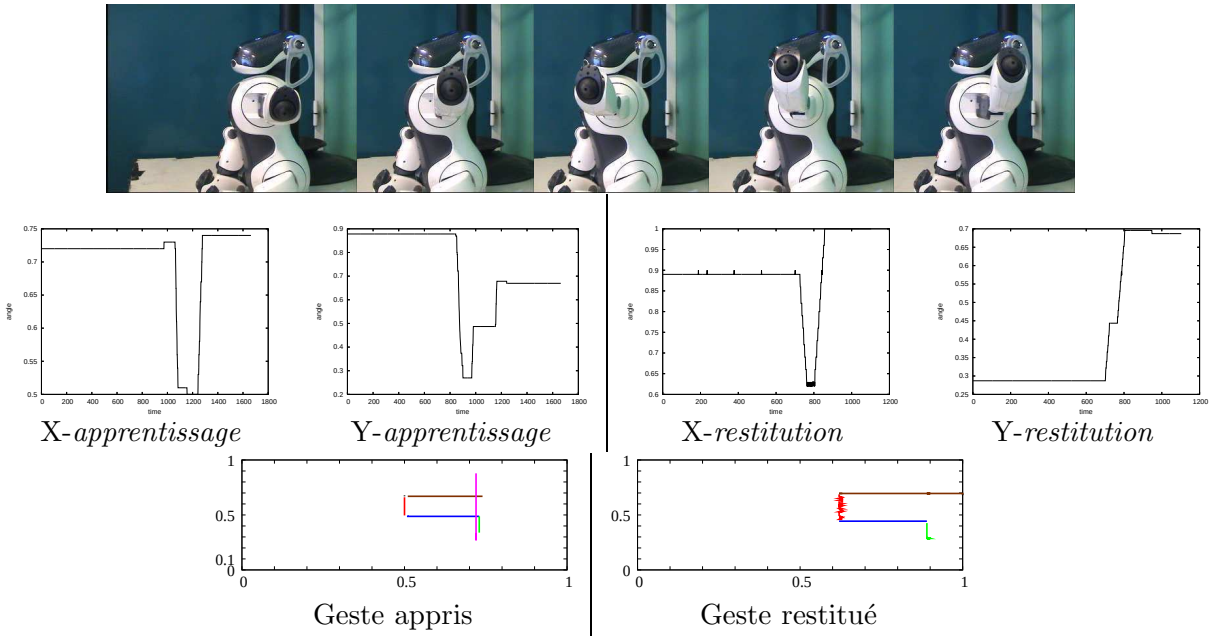


Fig. 3.35: En haut : Aibo reproduit la séquence apprise Au centre : X-*apprentissage* et Y-*apprentissage* sont respectivement les informations motrices horizontales et verticales pendant que le robot apprend la séquence. X-*restitution* et Y-*restitution* sont les informations motrices durant la reproduction de la séquence. Sur la figure Y-*restitution*, le premier mouvement n'est pas reproduit (non prédit), mais il est fourni par l'expérimentateur pour amorcer la restitution de la séquence. Les axes horizontaux sont le temps en itérations et les axes verticaux sont les angles des moteurs de la patte du robot.

3.4 Conclusion

Dans ce chapitre, j'ai présenté le cervelet et l'hippocampe comme structure du cerveau permettant l'acquisition de nouveaux comportements. J'ai développé et testé deux modèles permettant l'apprentissage de séquences temporelles. Le premier apprend en ligne le timing de séquences temporelles simples (séquences n'ayant pas d'éléments répétés). Le second modèle repose sur les propriétés d'un réservoir de dynamiques, il apprend en ligne des séquences complexes. Néanmoins, les modèles reposant sur des réservoirs de dynamiques nécessitent une certaine période de stabilisation des dynamiques. Par conséquent, durant cette période les réponses en sortie ne

correspondent pas à la séquence apprise. Finalement, une architecture apprenant le timing d'une séquence complexe a été proposée. Avec l'ajout de dynamiques internes, l'architecture permet de créer des états cachés. Ceux-ci sont recrutés permettant d'économiser le nombre de neurones pour apprendre une séquence complexe. Il permettent de lever les ambigüités des séquences (séquences ayant des éléments répétés). Les tests en simulation et sur robot ont montré la nécessité d'ajouter un mécanisme de resynchronisation permettant de retrouver les bons états cachés pour permettre d'amorcer une séquence complexe par un état intermédiaire.

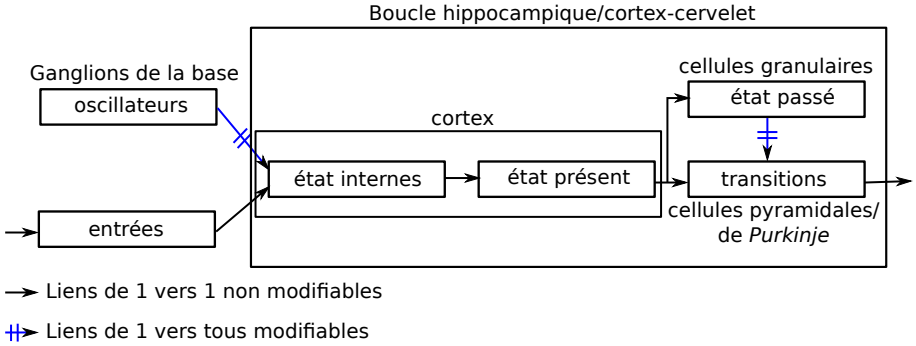


Fig. 3.36: modèle proposant les structures pouvant être impliquées dans l'aprentissage de séquences temporelles complexes. Les dynamiques générées par les oscillateurs proviendraient des ganglions de la base via le cortex. L'apprentissage des états internes est réalisé dans le cortex (entrée de l'hippocampe ou du cervelet). Les états sont mémorisés dans les cellules granulaire et l'apprentissage des transitions entre les états en entrée et ceux en mémoire est réalisé par les cellules pyramidales de la corne d'Ammon (CA3) ou les cellules de Purkinje pour le cervelet.

Le modèle proposé (figure 3.36) garde les inspiration neurobiologique de l'hippocampe ou du cervelet avec les cellules pyramidales de la corne d'Ammon (CA3) ou les cellules de *Purkinje* apprenant les transitions entre les événements mémorisés dans les cellules granulaires et les nouveaux événements arrivant du cortex. Les dynamiques permettant de lever les ambigüités des séquences complexes pourraient provenir des ganglions de la base via des voies corticales.

Chapitre 4

Apprentissage de propriétés spatiales et temporelles

L'objectif de ce chapitre est de permettre à un robot d'apprendre et de restituer des comportements. Comme nous l'avons vu dans le chapitre précédent, un comportement peut être appris comme une séquence, mais il faut aussi lier ces séquences avec les états ou catégories apprises dans l'environnement. Pour valider les différentes architectures possibles, j'ai travaillé sur un robot mobile allant de lieux en lieux avec l'objectif d'apprendre la séquence des déplacements de son itinéraire. Cela impose au préalable d'être capable de rejoindre des lieux, de les reconnaître et d'aller d'un lieu à l'autre.
Une partie des travaux en navigation visent à s'inspirer des mécanismes du cerveau des mammifères. L'un des animaux sûrement le plus étudié est le rat. En effet, il a été mis en évidence que certaines cellules du cerveau du rat réagissaient à des lieux particuliers de son environnement. Les cellules de lieux réagissent à des indices visuels ou à des informations d'intégration de chemin [Knierim *et al.*, 1995]. Ces cellules sont des neurones qui ont été localisés dans l'hippocampe du rat [O'Keefe et Dostrovsky, 1971]. D'autres travaux sur les rats montrent que des neurones codent pour des orientations particulières de la tête de l'animal [Muller *et al.*, 1996].
Dans le chapitre 3, j'ai présenté un modèle d'apprentissage de séquences temporelles. Ce modèle repose sur l'hypothèse que l'hippocampe peut apprendre des transitions grâce aux cellules pyramidales de CA3. De plus, ce modèle permet d'apprendre le timing des transitions, c'est-à-dire le temps entre deux états d'une séquence. Cette gestion du timing repose sur l'hypothèse que les cellules granulaires de DG permettent de maintenir l'activité d'un état plus ou moins longtemps selon leurs tailles. Finalement, cette modélisation de l'hippocampe permet d'apprendre des informations essentiellement temporelles.
Nous avons donc ici deux visions différentes sur le rôle de l'hippocampe dans l'apprentissage de comportement. D'un coté, il apprend des informations spatiales et de l'autre des informations temporelles. Malgré ces différences dans la manière d'encoder un comportement, est-ce que ces deux propriétés de l'hippocampe permettent d'apprendre un même comportement ? Comment un comportement peut-il être encodé par ses propriétés spatiales et temporelles ?
Nous avons ici deux facettes d'un même mécanisme. Si les lieux ou primitives de lieux sont reconnues dans EC, l'hippocampe apprendrait des transitions de lieux forcément temporelles. Dans le cadre d'une tâche spatiale, les neurones de l'hippocampe, les transitions donnent aussi l'impression de correspondre à une activité principalement spatiale.
Ces deux inspirations soulèvent la question d'un modèle unifié de la boucle hippocampique. Comment fusionner ces propriétés dans un seul modèle permettant à un robot d'apprendre des comportements spatio-temporels ? Dans l'objectif de fusionner deux modèles de la boucle hippocampique, je montre dans ce chapitre qu'un même comportement peut être appris d'une part, à partir d'informations spatiales et d'autre part d'informations temporelles. Dans une première partie je présenterai un modèle qui permet à un robot mobile de se déplacer dans l'environnement en apprenant des informations spatiales. Puis dans une seconde partie je présenterai un modèle qui permet d'apprendre le même comportement à partir des informations temporelles.
Le comportement testé ici est la navigation sur un robot mobile *Robulab10*[1] équipé d'une caméra montée sur deux moteurs *Pan* et *tilt* (figure 4.1), ainsi que d'une boussole électronique jouant le rôle de proprioception.

4.1 Construction d'un attracteur spatial

Pour créer des cellules de lieu, notre robot réalise un panorama de son environnement visuel et extrait de chaque capture des points d'intérêts (*what*) ainsi que leurs positions dans le panorama

[1] Plateforme mobile robotique de Robosoft

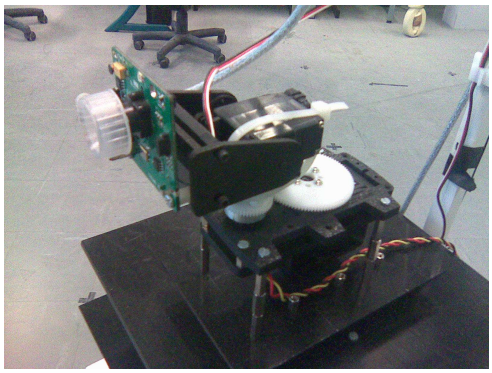


Fig. 4.1: Caméra montée sur deux moteurs en configuration *Pan-Tilt*

(*where*). Ensuite, en associant chaque cellules de lieu à une direction, le robot peut se déplacer dans son environnement (figure 4.2) [Giovannangeli et Gaussier, 2007,Giovannangeli et Gaussier, 2008].

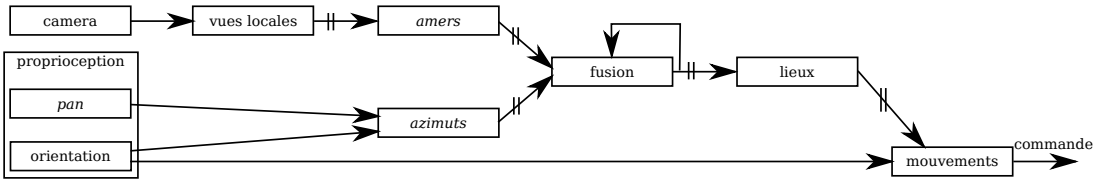


Fig. 4.2: Modèle permettant à un robot de naviguer à partir d'information visuelles. Des *amers* (information du *what*) sont appris à partir de vues locales grâce aux images capturées par une caméra. En parallèle, les *azimuts* (information du *where*) correspondant à la position angulaire de chacun des *amers* sont calculés à partir des informations proprioceptives (boussole électronique et direction de la tête (moteur *pan*)). Le *what* et le *where* sont ensuite fusionnés dans un tenseur (groupe "fusion"). Ce tenseur est ensuite appris et permet d'encoder un nouveau lieu (groupe "lieux"). Les lieux sont finalement associés aux informations proprioceptives (boussole électronique) du robot.

4.1.1 Direction de la tête

Pour se localiser dans son environnement quel que soit l'orientation de son corps, le robot doit avoir accès aux informations visuelles sur un panorama entier. C'est pourquoi le robot est équipé d'une caméra montée sur un moteur qui permet une rotation sur 360°. Ici le comportement de mouvement de la tête est entièrement pré câblé ; le robot tourne la tête de manière réflexe. Les 360 positions possibles sont projetées sur un vecteur de neurones de taille plus réduite (figure 4.3).
En activant successivement ces neurones, on peut faire tourner la caméra pour permettre au robot de capturer une partie ou la totalité d'un panorama visuel. Néanmoins, le temps de stabilisation de l'asservissement du moteur ainsi que le temps de capture d'une image font qu'un panorama entier n'est traité qu'en quatre secondes environ. De manière à diminuer ce temps, je considèrerai deux modes de fonctionnement de la tête : le premier est la capture d'un panorama entier lors de la phase d'apprentissage. Ce mode permet alors de reconnaître un lieu quelques soit l'orientation du robot. Le second mode consiste à ne capturer que la moitié d'un panorama lors de la phase de navigation autonome du robot. Ce mode est suffisant pour permettre une reconnaissance plus rapide du lieu dans lequel il se trouve. Pour accélérer encore les choses, le robot ne s'arrête pas lors de l'acquisition des images. La capture du demi panorama revient donc à mettre à jour

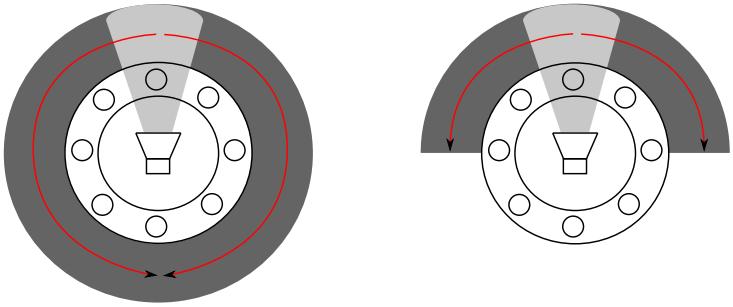


Fig. 4.3: Illustration du mouvement de la caméra. A gauche, la caméra parcours un panorama complet, alors qu'à droite elle parcours la moitié d'un panorama. Chaque neurone du vecteur autour de la caméra code pour une position de la caméra. La zone en gris clair est le champs de capture de la caméra. La zone en gris foncé illustre l'angle total. Les déplacements de la caméra se font selon les flèches.

en continu une mémoire dynamique. Le fait que les informations ne soient pas toutes capturées pour la même position induit un "bruit" qui ne perturbe pas le réseau de neurones de navigation, car on se contente de mettre en competition des cellules de lieu (leur activité change en fonction de la trajectoire, mais pas leur rang). Pour chaque capture, l'architecture extrait des points d'intérêts ainsi que leurs positions dans le panorama visuel.

4.1.2 Le traitement visuel bas niveau

Après avoir capturé une vue (figure 4.4 en haut), le robot calcule le gradient de celle ci. C'est à partir de ce gradient que sont extraits les points d'intérêt par ordre de saillance (figure 4.4 au milieu). Pour chaque point d'intérêt, une imagette autour de ce point est extraite puis transformée en coordonnées log-polaire (figure 4.4 en bas).

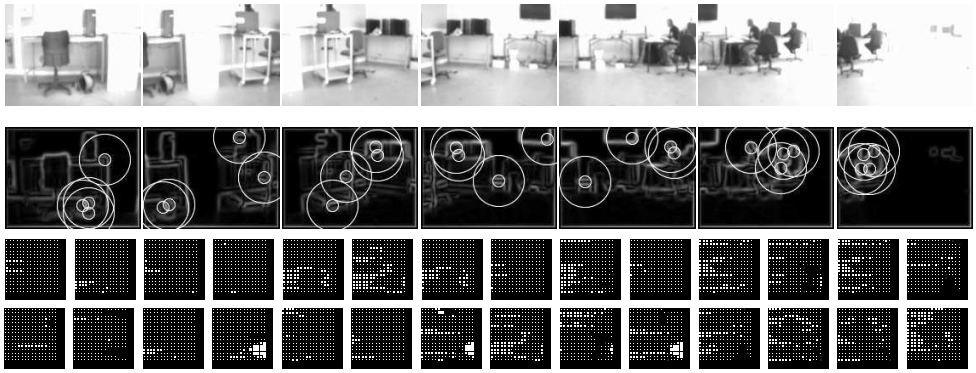


Fig. 4.4: Exemple de point d'intérêts extraits d'un demi panorama. Le robot calcul le gradient de chaque vue capturée. Quatre points d'intérêts sont extrais du gradient. Chacun des points d'intérêt ainsi que ses voisins est converti en coordonnées log-polaire.

Cette transformation s'inspire de la répartition des capteurs de la rétine ainsi que de l'aspect rétinotopique de la projection de la rétine sur le cortex visuel. Par le caractère polaire, elle apporte une certaine robustesse aux faibles variations d'assiette du robot (effet de quantification $360^{\circ}/32 \approx 10^{\circ}$) par rapport à la rotation centrale. Le caractère radial logarithmique apporte une certaine robustesse face aux changements d'échelle. Le résultat de cette transformation est appelé vue locale. Les vues locales sont ensuite apprises par un vecteur de neurones. Chaque neurone est un amer codant l'information du "what".

L'activité d'un neurone codant un amer fournit un niveau de confiance sur la reconnaissance de la vue locale courante. L'activité l_k d'un neurone k ayant appris une vue locale dans l'image du gradient est donnée par la formule suivante :

$$l_k = f^{\alpha^L}(\frac{1}{n_{I^L}.m_{I^L}}.\sum_{i=1}^{n_{I^L}}\sum_{j=1}^{m_{I^L}}(1 - \|w_{ij,k}^{I^L} - Act_{ij}^L\|)) \tag{4.1}$$

$$f^{\alpha^L}(x) = \frac{1}{1-\alpha^L}[x - \alpha^L]^+ \tag{4.2}$$

avec n_{I^L} et m_{I^L} respectivement le nombre de lignes et le nombre de colonnes des imagettes log-polaire. $w_{ij,k}^{I^L}$ est le poids synaptique entre le point i,j de la vue locale courante et le k^{eme} neurones du groupe. Act_{ij}^L est l'activité du ij^{eme} point de la vue locale courante compris dans l'intervalle [0,1]. $f^{\alpha^L}(x)$ est une fonction d'activation qui rehausse la dynamique des réponses (α^L est un seuil de reconnaissance), avec la fonction $[y]^+$ telle que $[y]^+ = y \; si \; y > 0 \; sinon \; 0$. Dans l'architecture que j'utilise, $\alpha^L = 0.8$ (en dessous de 0.8, l'activité des neurones n'est pas significative. La dynamique est entre 0.8 et 1).

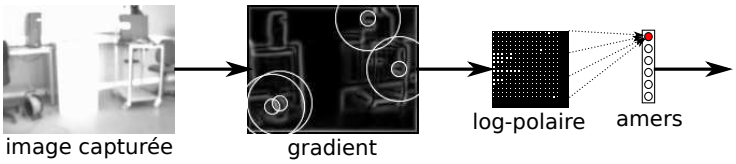


Fig. 4.5: Illustration de la chaîne de traitement d'un point d'intérêt. A partir du gradient d'une capture, un amer est appris à partir de l'imagette en log-polaire. L'apprentissage est réalisé sur les connexions en pointillés. Pour des raisons de lisibilité, toutes les connexions ne sont pas représentées.

L'apprentissage d'un amer est réalisé en un coup en suivant la règle :

$$w_{ij,k}^{I^L} = V.R_k^L.Act_{ij}^L \tag{4.3}$$

avec R_k^L un signal de recrutement du neurone k qui vaut 1 lors de l'apprentissage, sinon 0. Le principe du recrutement d'un neurone k du groupe consiste à faire passer R_k^L à 1 en même temps qu'un signal de vigilance V tel que $V = 0,1$. Pour que le robot apprenne rapidement, l'apprentissage est ici réalisé en un coup. Néanmoins, le signal de recrutement R_k^L pourrait être calculé de manière à permettre d'adapter un neurone déjà appris, ou alors d'en recruter un nouveau.
En plus de cette information, la position de chaque point d'intérêt dans le panorama visuel est extraite et recalée par rapport à un référentiel absolu en tenant compte de la rotation de la caméra. Le référentiel est fourni par une boussole électronique donnant la distance en degrés par rapport au nord. La boussole agit alors comme une proprioception du robot. Avec un tel mécanisme, quelque soit la rotation du robot dans un même endroit, il traitera le même panorama. La position des points d'intérêt dans un panorama visuel fourni les *azimuts* codant une information de "where".

4.1.3 Fusion des informations et apprentissage des cellules de lieu

Les informations "what" et "where" sont ensuite fusionnées dans un vecteur de configuration spatiale d'amers. La fusion est réalisée dans un espace produit (un tenseur du second degré compressé dans un vecteur de neurones produit m_k) définissant un tenseur M de la configuration

spatiale des amers. Cette fusion réalise un ET analogique. Les neurones du tenseur caractérisent un point (ou une région) dans l'espace *landmark-azimut*. L'apprentissage d'un neurone du tenseur suit les équations suivantes :

$$\Delta\omega_{ak}^{LM} = \Gamma_1(l_l(t)) \cdot R_k^M(t) \tag{4.4}$$

$$\Delta\omega_{lk}^{AM} = \Gamma_1(\theta_a(t)) \cdot R_k^M(t) \tag{4.5}$$

avec $\Gamma_1(x) = 1$ *si* $x \geq 1$ *sinon* 0 (fonction rampe). l_l et θ_a respectivement le l^{eme} *landmark* et le a^{eme} *azimut*. R_k^M est un signal de recrutement du k^{eme} neurone. $R_k^M = 1$ *si* le neurone k est le neurone recruté, *sinon* 0. Une fois l'apprentissage réalisé, les activités des neurones m_k du tenseur M codent la configuration spatiale d'amers d'un panorama appris. Chaque élément du tenseur répond proportionnellement au couple *landmark-azimut* suivant :

$$\begin{aligned} \mathcal{L}_k(t) &= \sum_{l=1}^{n_L} \omega_{lk}^{LM}(t).l_l(t) \\ \mathcal{A}_k(t) &= \sum_{a=1}^{n_\Theta} \omega_{ak}^{AM}(t).\theta_a(t) \end{aligned}$$

avec $\omega_{lk}^{LM}(t)$ et $\omega_{ak}^{AM}(t)$ respectivement les poids des connexions entre le l^{eme} *landmark* et le k^{eme} neurone du tenseur et des connexions entre le a^{eme} *azimut* et le k^{eme} neurone du tenseur (ω_{lk}^{LM} et ω_{ak}^{AM} sont initialisés à 0). n_L et n_Θ sont le nombre de *landmarks* et *azimut* recrutés.
De plus, la réponse de chaque neurones m_k du tenseur dépend d'une mémoire à court terme (STM) et des nouveaux *landmarks* $\mathcal{L}_k$ et *azimuts* $\mathcal{A}_k$. La mémoire à court terme est utilisée pour maintenir les activités du tenseur M durant tout le traitement d'un panorama complet.

$$m_k(t) = \max\left(\mathcal{L}_k(t) \cdot \mathcal{A}_k(t), \left[\lambda^M(t).m_k(t - d_t) - r_k(t)\right]^+\right) \tag{4.6}$$

avec $r_k(t)$ un signal de remise à zéro de l'activité du neurone k du tenseur au début de l'exploration du panorama visuel. Un terme d'oubli $\lambda^M(t)$ est appliqué sur l'activité du k^{eme} neurone du tenseur. Une fois que le panorama visuel entier est appris, le robot apprend une cellule de lieu p. Cet apprentissage est réalisé en un coup suivant la règle suivante :

$$\Delta\omega_{kp}^P = \Gamma_1\left(m_k(t)\right) \cdot R_p^P(t) \tag{4.7}$$

avec ω_{kp}^P poids binaires initialisés à 0 des connexions entre le k^{eme} neurone du tenseur et le p^{eme} neurone du vecteur des cellules de lieu. L'algorithme de recrutement est le même que celui utilisé dans les équations 4.4 et 4.5. L'activité d'une cellule de lieu résulte du calcul de la distance entre le tenseur appris et le tenseur courant. Donc l'activité P_p de la p^{eme} cellule de lieu est exprimée comme suit :

$$P_p(t) = \frac{1}{W_p}\left(\sum_{k=1}^{n_M} \omega_{kp}^P(t) m_k(t)\right) \tag{4.8}$$

avec $\omega_{kp}^P(t)$ qui exprime le fait que le k^{eme} neurone du tenseur a été utilisé pour encodé la cellule de lieu p. Le nombre de neurone du tenseur utilisé par la p^{eme} cellule de lieu est donné par $W_p = \sum_{k=1}^{n_M} \omega_{kp}^P$ avec n_M le nombre de neurones recrutés dans le tenseur.

4.1.4 Test des cellules de lieu

Pour illustrer les réponses des cellules de lieu, j'ai placé 25 marques au sol dans une salle. Ces marques sont positionnées régulièrement et forme un carré de 5 par 5 de coté comme le montre la figure 4.6.a. Il est important de noter que le robot ne voit pas ces marques au sol. Dans un premier temps, le robot est placé successivement sur chaque marque et à chaque fois je lui fais apprendre un nouveau lieu. Il y a donc 1 lieu par marque. Ensuite, je laisse le robot se déplacer sur chaque ligne de marques. Durant sont déplacement j'observe les activités de chacune des cellules de lieu. La figure 4.6.b montre ces activités.

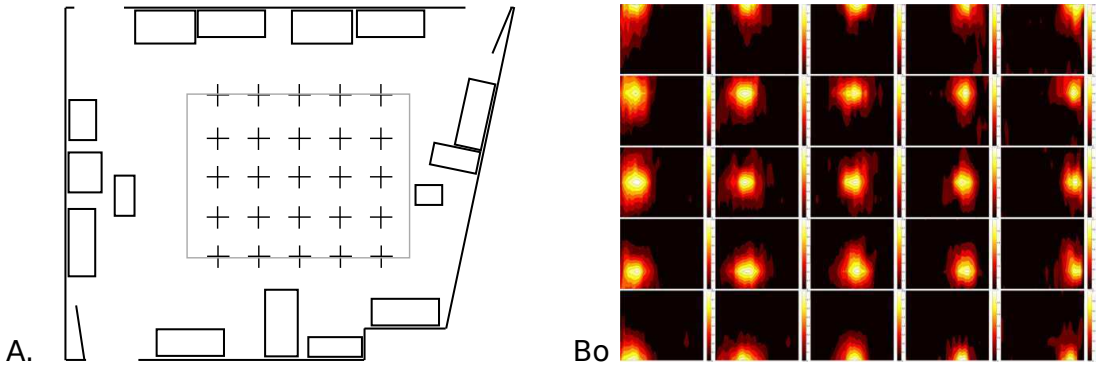


Fig. 4.6: A) Salle d'expérience avec 25 marques à égale distance (90 cm). 25 lieux sont appris régulièrement sur chaque marque. B) Test de la réponse des cellules de lieu appris en A. Une compétition entre tous les lieux engendre le pavage de l'environnement.

Un lieu appris dans un lieu A réponds à son maximum en A et créé un large champ décroissant autour de A. Un tel système est capable d'apprendre plusieurs régions de l'environnement. Par conséquent, le robot peut se localiser visuellement grâce aux cellules de lieu. Ces cellules réagissent dans des régions particulières de l'environnement.

4.1.5 Du lieu à l'action

En associant le mouvement courant du robot avec la cellule de lieu active, Le robot est capable de se déplacer dans l'environnement en suivant une trajectoire particulière. Ici, le robot navigue de manière totalement réactive, il ne réalise pas de prédiction.

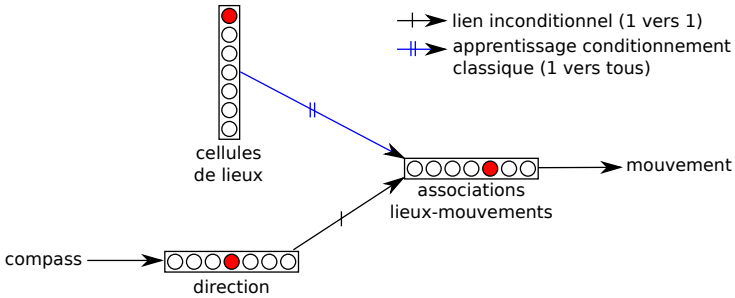


Fig. 4.7: Apprentissage associatif de lieux-mouvements. Une fois l'apprentissage réalisé, la cellule ayant la plus forte activité permet de restituer le mouvement que le robot doit réaliser.

L'activité Act_a du neurone a du groupe des *associations lieux-mouvements* est calculé suivant

l'équation :

$$Pot_a = \sum_{l=1}^{nblieux} Act_l.w_{la} \quad (4.9)$$

$$Act_a = f(Pot_a) \quad (4.10)$$

avec *nblieux* le nombre de neurones codant les cellules de lieu, Act_l l'activité du neurone l du groupe des cellules de lieu et w_{la} le poids de la connexion entre le neurone l du groupe des cellules de lieu et le neurone a du groupe des associations lieux-mouvements. La fonction f est une fonction identité.
L'apprentissage est réalisé sur les connexions entre les cellules de lieu et le groupe des associations lieux-mouvements suivant l'équation :

$$Sd_a = \sum_{d=1}^{nbdirection} Act_d.w_{da} \quad (4.11)$$

$$\Delta w_{la} = w_{la} + \varepsilon(Sd_a * Act_l) \quad (4.12)$$

avec ε la vitesse d'apprentissage. Ici $\varepsilon = 1$, car nous sommes dans le cadre d'un apprentissage en un coup. Sd_a est l'activité de la sortie désirée pour le neurone a du groupe des associations lieux-mouvements calculée à partir de l'activité Act_d du neurone d du groupe des directions. w_{da} est le poids entre les groupes des directions et le groupe des associations lieux-mouvements.

4.1.6 Navigation spatiale sur robot mobile

Dans cette expérience, le robot doit apprendre à se déplacer dans l'environnement. Par défaut, le robot avance à vitesse constante. Lorsque le robot s'éloigne de la trajectoire que le professeur souhaite lui faire apprendre, ce dernier le corrige avec un joystick ou un cou artificiel équipé de capteurs tactiles (voir annexe 9.2). Ici le joystick joue le rôle d'une laisse qui sera autour du cou du robot. Lorsque le professeur tire sur la laisse, cela modifie la dynamique du robot et le fait changer de direction.
Cela va avoir pour effet de déclencher l'apprentissage d'un nouveau lieu ainsi que d'une nouvelle association lieu-mouvement. A chaque position de la tête, le robot extrait quatre points d'intérêt qu'il apprend. L'ensemble des points d'intérêt du panorama permettent ensuite de coder un nouveau lieu. Ce dernier est ensuite associé au mouvement courant du robot. Le robot a accès à son mouvement courant grâce à une boussole électronique qui joue le rôle d'information proprioceptive (équivalent à un système vestibulaire).
En répétant ces corrections dans différentes zone de l'environnement, le robot se construit un attracteur dans lequel il se laisse porter. Après environ trois tours, le robot est capable de suivre la trajectoire désirée de manière autonome et de revenir sur la trajectoire si on le kidnappe ou si on le fait partir d'une position jamais apprise, mais apportant un même voisinage visuel.
La figure 4.8.A montre deux rondes apprises et reproduites par le robot. La première montre l'apprentissage sur un tour ainsi que la reproduction sans intervention du professeur. On observe que le robot reproduit la trajectoire désirée. Malgré tout, on remarque qu'il existe une certaine dérive de la trajectoire reproduite par rapport à celle apprise. Cette dérive s'explique par l'étendue de la reconnaissance des lieux. Lorsque le robot apprend une nouvelle cellule de lieu, il se trouve en son centre. Après apprentissage, lorsque le robot arrive dans cette même cellule de lieu, il commence à la reconnaître quand il est proche de la frontière du lieu, donc avant d'être au centre du lieu. Par conséquent, la trajectoire reproduite par le robot tend à être contractée.

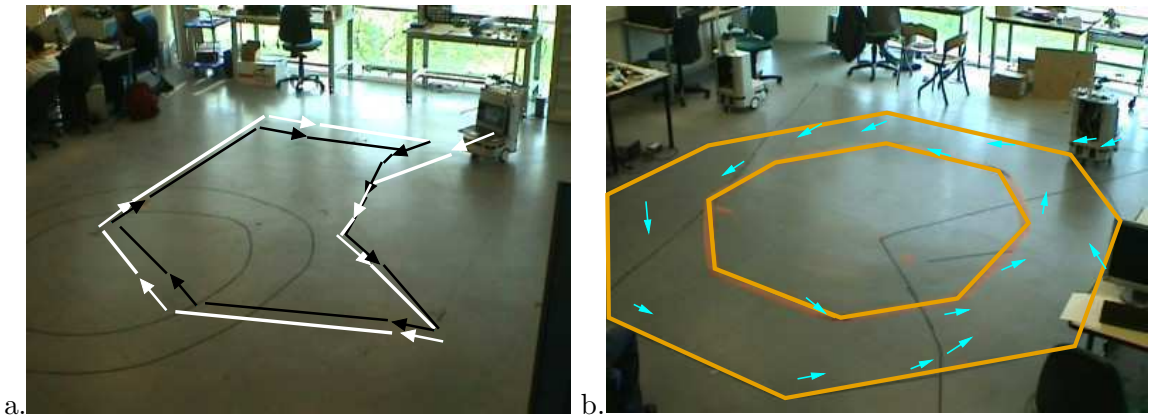


Fig. 4.8: Apprentissage de trajectoires par correction de la dynamique du robot. Le robot avance à vitesse constante de manière réflexe. Le professeur corrige le robot en modifiant la dynamique du robot. Chaque flèche représente une correction appliquée par le professeur. Donc le robot apprend une association entre le lieu et le mouvement courant. A) L'architecture apprend une ronde précise et après un tour de correction par le professeur (flèches blanches) le robot reproduit la trajectoire de manière autonome (flèches noires). B) Après trois tours d'apprentissage (les flèches montrent où le robot a appris de nouvelles associations lieux-mouvements), le professeur ne corrige plus le robot.

La figure 4.8.B montre quelle trajectoire (lignes orange) le robot doit suivre ainsi que les endroits où il a appris de nouvelles associations lieux-mouvements (flèches bleues). Dans cette expérience le robot apprend non pas sur un seul tour, mais jusqu'à ce qu'il reproduise la trajectoire de manière autonome : trois tour dans cette expérience. Au fur et à mesure des tours, le professeur à de moins en moins besoin d'intervenir pour apprendre au robot à suivre la bonne trajectoire. Les erreurs de généralisation du premier tour servent à contre balancer les premiers vecteurs appris de manière à créer un véritable bassin d'attraction grâce à la compétition entre lieux.

4.2 Construction d'un attracteur temporel

Pour permettre à un robot d'apprendre une trajectoire particulière comme une séquence de mouvements (navigation proactive) et non plus comme un mouvement restitué à partir d'un lieu (navigation réactive), j'utilise une seconde boucle sensori-motrice (figure 4.9).

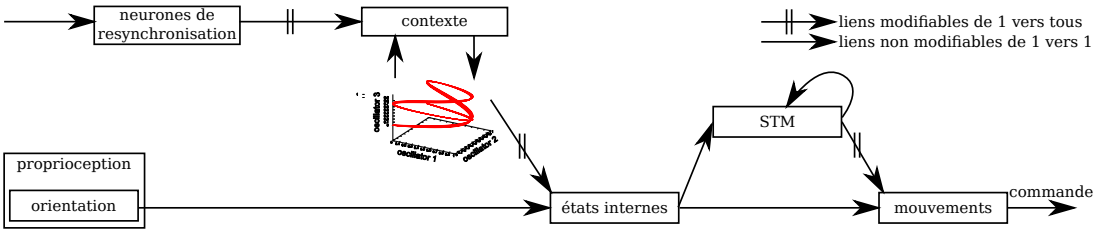


Fig. 4.9: Modèle permettant d'apprendre des séquences temporelles complexes de mouvements. Lors de la détection d'un nouveau mouvement à partir des informations proprioceptives (boussole électronique), le nouvel état (nouvelle orientation au sol) est associé à la dynamique interne pour créer un nouvel état caché. Cet état permet alors l'apprentissage d'une nouvelle transition avec l'état précédent (présent en mémoire à court terme). Le nouvel état est ensuite gardé en mémoire à court terme à la place du précédent. A chaque nouveau mouvement détecté, le contexte fourni par la dynamique interne est appris sur les connexions entre le neurone de resynchronisation actif et les neurones de "contexte". De cette manière, lors de la restitution de la séquence, ce mécanisme permet de resynchroniser la dynamique interne permettant alors de retrouver le bon état caché

Dans le chapitre 3, j'ai présenté un modèle d'apprentissage de séquences temporelles complexes. Ce modèle utilise des dynamiques générées par des oscillateurs pour construire des états internes et ainsi lever l'ambiguïté des séquences complexes. Ce modèle a été testé sur un robot Aibo[2] pour l'apprentissage et la restitution d'un geste.
Néanmoins, cette architecture ne peut pas s'appliquer en l'état dans une expérience de navigation. En effet, lorsque l'architecture envoie une nouvelle commande motrice à la patte d'Aibo, elle est tout de suite appliquée et la patte prend l'orientation désirée immédiatement, ou en tout cas dans un temps extrêmement court. Il n'y a donc que très peu de temps entre le moment où le robot réalise le nouveau mouvement et l'instant où il le perçoit sur ces moteurs. Dans le cadre de la navigation, la commande motrice envoyée est l'orientation que doit prendre le robot pour suivre correctement la trajectoire. Le temps que le robot passe de son orientation courante à l'orientation désirée n'est absolument plus négligeable. Il faut donc que l'architecture soit capable de tenir compte de la dynamique propre du robot mobile.

4.2.1 Resynchronisation des dynamiques internes

Pour permettre au robot de tenir compte de sa propre dynamique motrice, il doit alors pouvoir resynchroniser ses propres dynamiques internes sur son état courant. C'est pourquoi j'ai ajouté un mécanisme permettant au robot d'apprendre l'état de ses dynamiques internes pour pouvoir les restaurer ensuite.

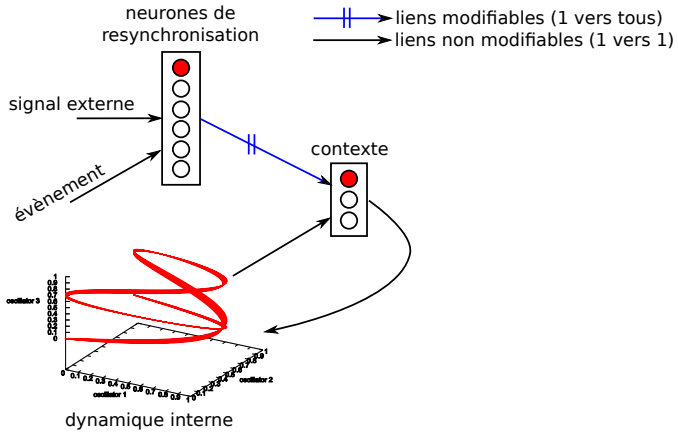


Fig. 4.10: Modèle permettant d'apprendre les états de la dynamique interne en les associant à des neurones de resynchronisation. A chaque événement, le neurone de resynchronisation actif passe au suivant. Lorsqu'un signal externe est détecté, alors il active le neurone correspondant à ce signal. L'apprentissage des contextes est réalisé sur les poids des connexions double barrées (connexions de un vers tous). Les flèches non barrées sont des connexions de un vers un dont le poids est fixe.

A chaque événement, le neurone actif dans le groupe passe au suivant. Lorsqu'un signal externe est actif, alors le neurone correspondant devient actif.
L'activité Act_c du neurone c du groupe apprenant les contextes à partir des neurones de resynchronisation est calculée suivant l'équation :

$$Act_c = f\Big(\sum_{i=1}^{nbresynchro} Act_i.w_{ic}\Big) \tag{4.13}$$

[2]Robot chien de Sony

avec *nbresynchro* le nombre de neurones dans le groupe de resynchronisation. Act_i est le i^{eme} neurone du groupe de resynchronisation et w_{ic} le poids de la connexion entre le i^{eme} neurone du groupe de resynchronisation et le c^{eme} neurone du groupe de contexte. $f()$ est une fonction identité.
Lorsqu'un nouvel événement arrive et que le neurone de resynchronisation n'a pas encore été associé à un contexte, alors l'apprentissage est réalisé suivant l'équation :

$$Pot_c = \sum_{j=1}^{nboscillateur} Act_j.w_{jc} \tag{4.14}$$

$$\Delta w_{ic} = w_{ic} + \varepsilon.Pot_c; \tag{4.15}$$

avec Pot_c le potentiel du neurone c du groupe de contexte calculé à partir des activités des oscillateurs. *nboscillateur* est le nombre d'oscillateurs qui génère la dynamique interne. Act_j l'activité de l'oscillateur j dans le groupe de la dynamique interne. w_{jc} est le poids de la connexion entre l'oscillateur j et le neurone c du groupe de contexte.

4.2.2 Test de la resynchronisation

L'objectif de ce test est de montrer que l'architecture est capable de restituer les contextes fourni par la dynamique interne lors de l'apparition des états d'une séquence. Pour chaque nouvel état détecté, un contexte est appris à partir des valeurs des oscillateurs. Les valeurs des oscillateurs sont associées à l'état de la séquence pour créer un état interne.
Cette propriété de resynchronisation a été évoquée dans le chapitre 3 avec le modèle d'apprentissage de séquence temporelle simple et avec les ESNs (*Echo State Network*). Le bon fonctionnement de l'architecture reposait sur un "reset" de la dynamique interne avant la phase de reproduction et également sur une latence nulle entre le moment de la prédiction d'un état et de son arrivée en entrée de l'architecture. Dans le cadre de la navigation sur un robot mobile dont les états de la séquence sont les orientations du robot, cette latence est non nulle : le robot prend un certain temps avant d'arriver dans son orientation cible. Il est alors indispensable de pouvoir resynchroniser la dynamique interne pour permettre de restituer la séquence correctement.
La séquence testée ici est une séquence temporelle simple : "1 2 0" (figure 4.11). La séquence est apprise telle quelle par l'architecture. Lors de l'arrivée de chaque état, l'état des oscillateurs faisant office de contexte interne est appris et associé à un neurone de synchronisation.
Lors de la reproduction, aucune remise à zéro de la dynamique interne n'a été faite. Pour mettre en évidence que le mécanisme de resynchronisation fonctionne correctement, la séquence est amorcée non pas par le premier état qui la compose, mais par l'état "2" (milieu de la séquence). Lorsque cet état est présenté en entrée, alors un état interne est activé en fonction des activités du contexte interne. Cet état interne ne correspond pas nécessairement à ce qui a été appris lors de la période d'apprentissage. Lorsqu'un signal externe est envoyé pour resynchroniser la séquence, les activités des oscillateurs sont restaurées en conséquence. Le signal externe déclenche à nouveau l'état "2" de la séquence. Cet état est de nouveau détecté et le bon état interne est actif permettant à l'architecture de prédire l'état suivant de la séquence "0".

4.2.3 Synchronisation de séquence ou apprentissage de plusieurs séquences

Le mécanisme de resynchronisation de séquences temporelles permet de retrouver le contexte de la dynamique interne à partir d'un signal de resynchronisation lorsqu'un état intermédiaire

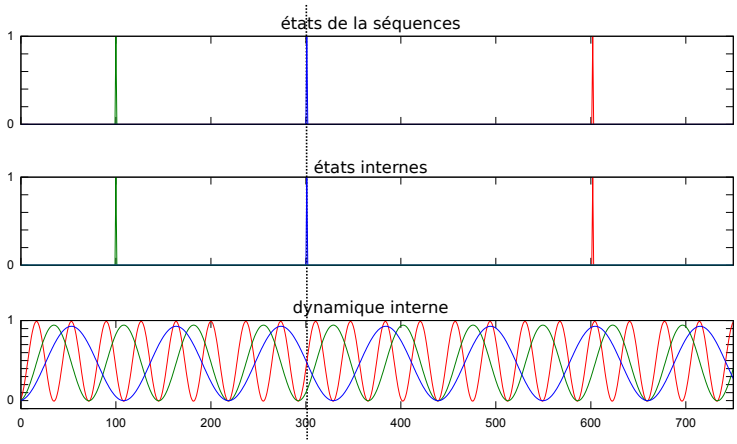


Fig. 4.11: Apprentissage d'une séquence temporelle simple. La séquence testée ici est "1 2 0". Sur le graphique du haut, l'état "1" est en vert, l'état "2" en bleu et l'état "0" en rouge. Sur le graphique des états internes, chaque couleur représente un état interne différent. Les couleurs n'ont aucun lien avec les états qui composent la séquence. Le graphique du bas montre les activités des neurones des oscillateurs qui génèrent la dynamique interne. Ici aussi les couleurs n'ont pas de lien avec les états de la séquence ni les états internes. Lors de la reproduction, la séquence débutera par le deuxième état : l'état "2". La ligne en pointillés permet de mettre en évidence l'état de la dynamique interne lors de l'apparition de l'état "2" de la séquence.

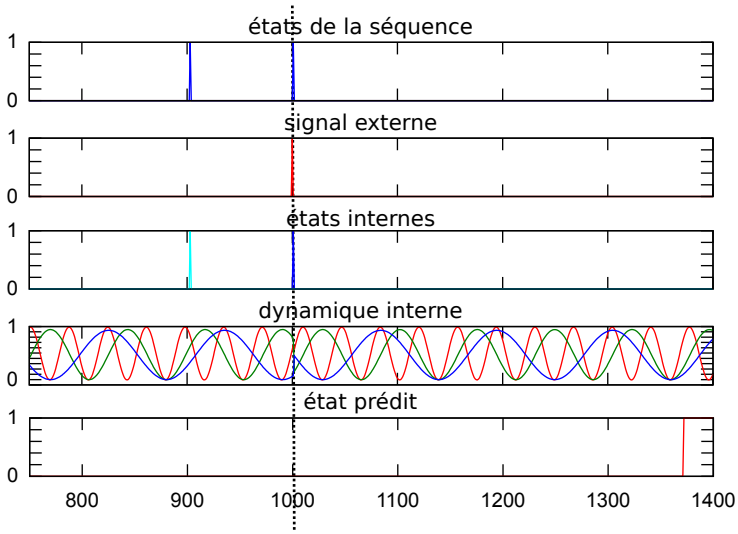


Fig. 4.12: Reproduction d'une séquence temporelle simple. La séquence qui a été apprise est "1 2 0". L'état "1" est en vert, l'état "2" en bleu et l'état "0" en rouge. Sur le graphique du haut on observe que l'état fourni pour amorcer la séquence est l'état "2". La séquence est donc amorcée non pas par le premier état qui la compose mais par un état intermédiaire. Le second graphique montre à quel moment le signal externe est survenu pour permettre de resynchroniser la dynamique interne. Le troisième graphique montre les états internes actifs à différents moments. Le quatrième graphique montre les activités des neurones des oscillateurs qui génèrent la dynamique interne. Le graphique du bas montre l'état prédit par l'architecture. On observe que lorsque l'état "2" est fourni, un état interne est actif, mais il ne correspond pas à celui qui a été appris précédemment. A ce moment, la dynamique interne reste inchangée et continue d'évoluer. Lorsque le signal de resynchronisation arrive, alors la dynamique interne est modifiée revenant dans l'état à laquelle elle a été apprise précédemment. Au même moment, le signal de resynchronisation réactive l'état "2" de la séquence et permet de réactiver le bon état interne. L'architecture peut ensuite déclencher la prédiction de l'état suivant de la séquence, l'état "0".

d'une séquence est détecté en entrée de l'architecture. Une question se pose : que se passe-t-

il si plusieurs séquences sont apprises successivement ? Pour répondre à cette question, nous considérons deux séquences ayant des états communs. La première est “1 2 3 4”, la seconde est “6 7 2 3 8”.

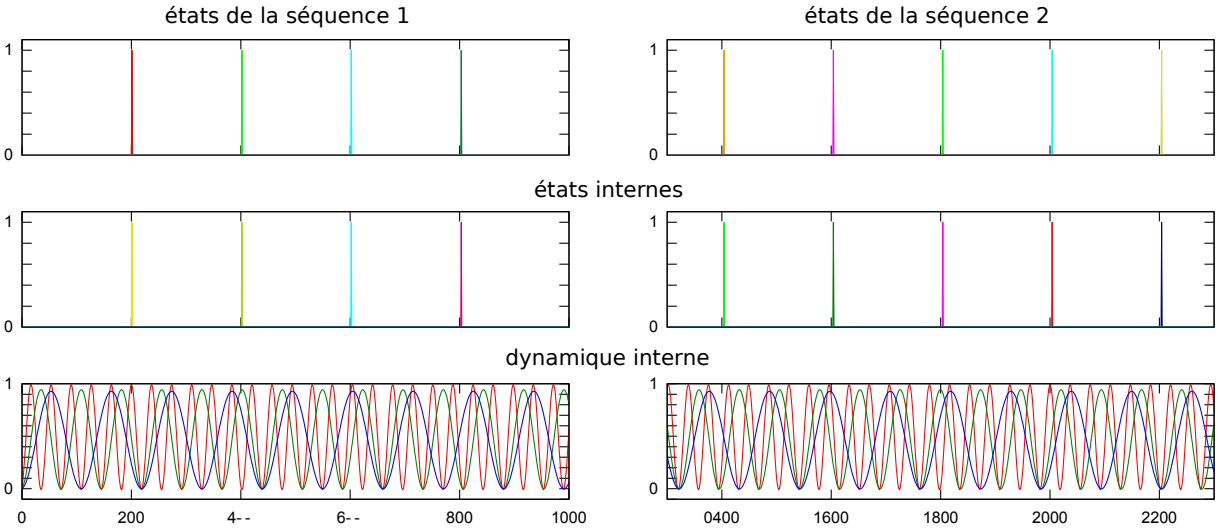


Fig. 4.13: Apprentissage successif de deux séquences temporelles : “1 2 3 4” (première ligne à gauche) et “6 7 2 3 8” (première ligne à droite). Ces séquences ont deux éléments communs. Chacun des éléments appris entraîne l’apprentissage d’états cachés (seconde ligne). On observe alors que les états cachés appris sont différents pour les éléments communs. La dernière montre l’état des dynamiques internes durant l’apprentissage des deux séquences.

La figure 4.13 montre les deux séquences apprises. Dans un premier temps, les deux séquences sont présentées successivement ; c’est à dire qu’il n’y a pas de remise à zéro de la dynamique interne entre les deux séquences. Tout au long de cette phase, l’architecture est en mode apprentissage. Pour chacun des états des deux séquences présentées en entrée, un état interne est créé à partir du contexte de la dynamique interne au même moment. Ce contexte est lui même associé à un neurone de synchronisation.
La figure 4.14 montre la reproduction des deux séquences. De manière à mettre en évidence le mécanisme de resynchronisation, la phase de reproduction débute arbitrairement par la seconde séquence. Dans un premier temps, un signal externe est envoyé permettant ensuite la resynchronisation. Le premier état qui compose la deuxième séquence est alors fourni. Lors de la détection de cet état, le contexte de la dynamique est restauré, ce qui permet de retrouver l’état interne qui avait été appris précédemment. L’architecture peut finalement prédire l’état suivant. Grâce à l’ajout d’une connexion récurrente entre la sortie prédite et l’entrée de l’architecture, l’état prédit est directement détecté en entrée de la séquence. Le contexte de la dynamique est alors restauré, l’état interne retrouvé puis détecté pour finalement prédire l’état suivant qui compose la séquence. Cette enchainement est répété jusqu’à la fin de la séquence où il n’y aura plus de prédiction.
Néanmoins, après la fin de l’apprentissage de la seconde séquence, il n’y a plus d’états présenté en entrée et donc pas de transition apprise. Lors de la fin de l’apprentissage de la première séquence, la présentation de la seconde séquence à apprendre s’enchaine directement. L’effet constaté est donc que lorsqu’on amorce la reproduction de la première séquence, l’architecture reproduit correctement la première séquence suivi de la seconde (figure 4.15). Une raison possible à cet effet peut s’expliquer au niveau des neurones du groupe de resynchronisation. En effet, ils sont tous liés à leurs successeurs à la manière d’une chaine qui a été pré câblé (figure 4.16).

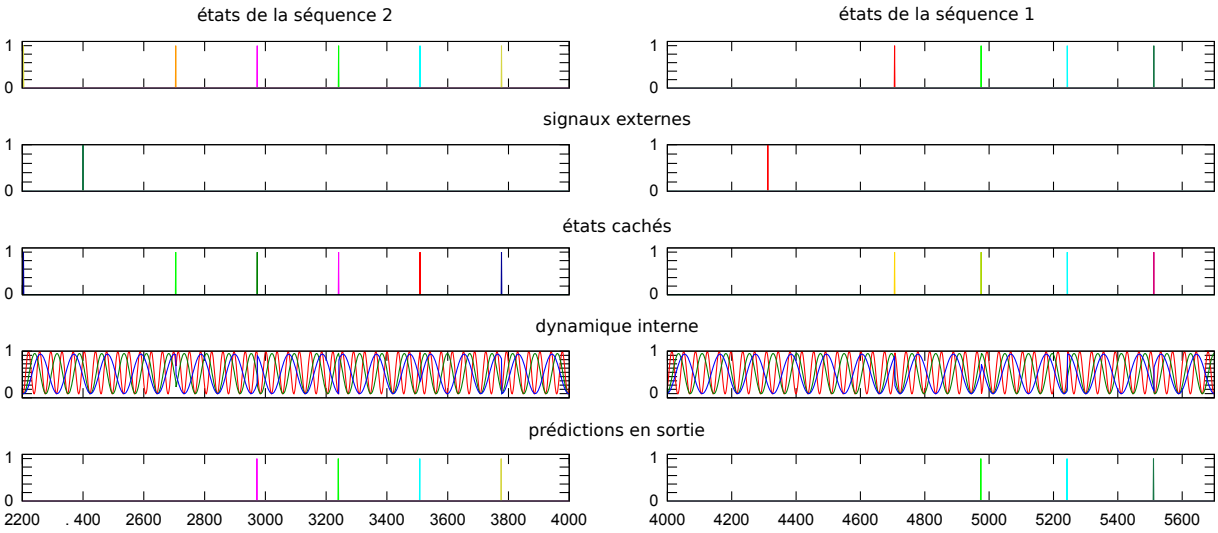


Fig. 4.14: Reproduction des deux séquences précédemment apprises. La seconde séquence est tout d'abord reproduite (à gauche), puis la première (à droite). Avant d'amorcer la reproduction d'une séquence en particulier en fournissant le premier état, un signal externe est donné (seconde ligne) de manière à resynchroniser les dynamiques internes pour la séquence souhaitée. La troisième ligne montre les états cachés répondant à chaque élément détecté. On remarque alors qu'ils correspondent bien à ceux qui ont été créés lors de l'apprentissage. Chaque élément détecté en entrée déclenche la resynchronisation des dynamiques internes (quatrième ligne) afin de retrouver l'état caché correspondant. Finalement, on observe que les éléments prédits en sortie (cinquième ligne) correspondent effectivement bien à la séquence apprise. Chaque élément prédit est détecté en entrée grâce à un lien de retour.

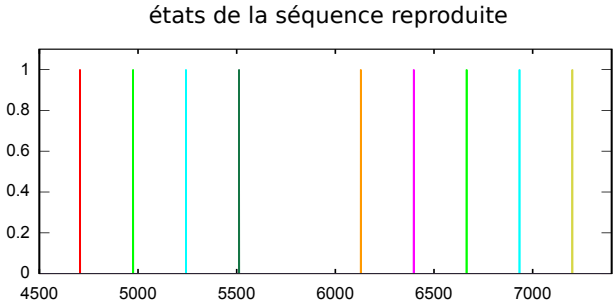


Fig. 4.15: Séquence reproduite lors du réamorçage de la première séquence. Une fois la séquence reproduite, on observe que la seconde s'enchaine juste après. En effet, l'apprentissage ayant été actif durant l'apprentissage des deux séquences, une transition liant le dernier état de la première séquence avec le premier état de la seconde a été apprise. Par conséquent, lorsque la prédiction du dernier élément de la première séquence est détecté, il entraîne la prédiction du premier élément de la seconde séquence et ainsi de suite avec les transitions suivantes de la seconde séquence.

Cet effet de combinaison de deux séquences successives, peut aussi être dû au fait que le groupe "CA3" qui apprend les transitions entre les états d'une séquence, apprend justement la transition entre le dernier état de la première séquence et le premier état de la seconde séquence. Une solution pour séparer les deux séquences serait de ne pas apprendre cette transition. Mais ajouter un signal qui permette de déclencher ou non l'apprentissage d'une transition pose la question de comment le système détecte le début et/ou la fin d'une séquence, mais aussi de détecter si la transition est nouvelle ou non. Il nous est donc apparu plus correct de laisser ce problème potentiel et de faire l'hypothèse que le lien appris problématique pourra être oublié si des apprentissanges concurrents ont lieu.
L'apport de neurones de synchronisation permet d'amorcer la reproduction d'une séquence pas

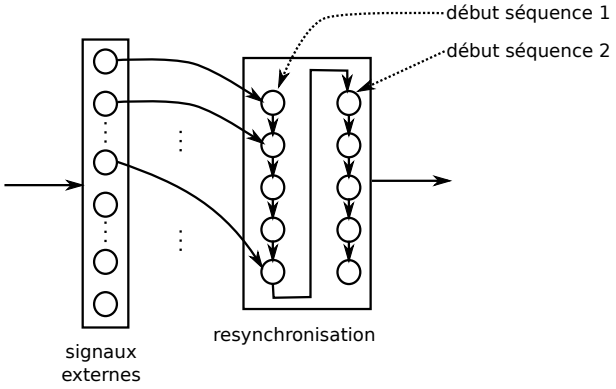


Fig. 4.16: Représentation schématique de deux séquences apprises mettant en évidence les connexions entre les signaux externes et le groupe de resynchronisation, ainsi que les connexions internes au groupe de resynchronisation. Les flèches en pointillé ne sont pas des connexions, mais montrent quel neurone de resynchronisation permet de réamorcer la séquence correspondante.

seulement par son premier élément, mais également par un état au milieu. En combinant ce mécanisme avec les états internes qui permettent de lever l'ambigüité (présence d'un même état plusieurs fois dans une séquence), il est possible d'apprendre plusieurs séquences ayant des états communs. On peut faire un parallèle avec des modèles de navigations reposant sur l'utilisation d'une carte cognitive, ici le groupe de synchronisation joue d'une certaine manière le rôle d'une carte cognitive pré câblée. Là où chaque noeud d'une carte cognitive peut représenter une transition de lieux (état spatial), ici chaque noeud de synchronisation représente un contexte de la dynamique interne (état temporel). On peut alors imaginer que les connexions entre les noeuds de synchronisation soient apprises de la même manière que peuvent l'être les noeuds d'une carte cognitive. En phase d'utilisation, en faisant l'hypothèse qu'une diffusion est réalisée entre les différents noeuds pour permettre de rejoindre un but/motivation, cette carte cognitive "temporelle" pourrait permettre de choisir quelle séquence amorcer à partir de l'objectif à atteindre (De futurs travaux visent à étudier les problématiques liées à la construction de cartes cognitives).

4.2.4 Navigation temporelle sur robot mobile

L'objectif de cette expérience est de montrer qu'un même modèle d'apprentissage de séquences temporelles qui a permis d'apprendre et de restituer un geste sur un bras robotique, peut également apprendre à naviguer. Ici le robot doit apprendre la succession de ses orientations sur le sol. L'information proprioceptive est fournie par une boussole électronique donnant la direction du robot. Le robot avec lequel j'ai réalisé cette expérience est un Robulab10[3].
La figure 4.17 montre l'expérience réalisée dans laquelle le robot apprend à naviguer avec le modèle d'apprentissage de séquences temporelles. Dans cette expérience, le robot avance à vitesse constante. La trajectoire en clair est le chemin appris au robot. Les flèches sur cette trajectoire montrent où le robot a appris chaque transition entre son orientation précédente et son orientation courante. Plus la distance entre deux changements d'orientation est grande, plus le timing entre deux transitions est long. La trajectoire foncée est le chemin reproduit par le robot. Les flèches foncées montrent où le robot a déclenché les prédictions lui permettant de prendre l'orientation correspondante.
En comparant les deux tracés de la figure 4.17, on observe une certaine dérive de la trajectoire

[3]Plateforme mobile Robosoft

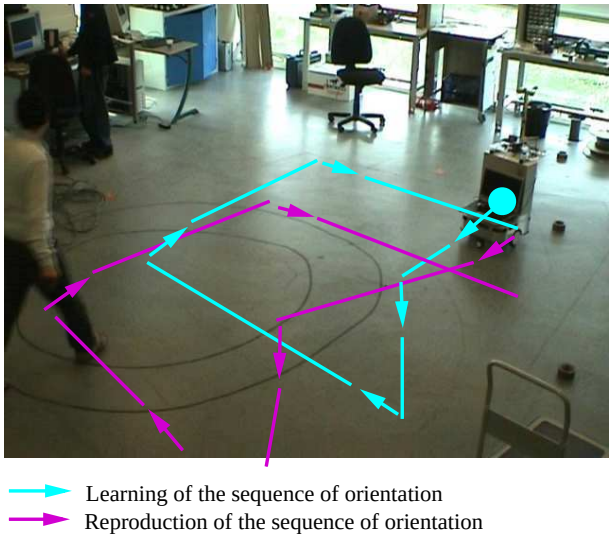


Fig. 4.17: Expérience de navigation avec le modèle d'apprentissage de séquences temporelles. La trajectoire claire est le chemin appris au robot. La trajectoire foncée, est le chemin reproduit par le robot. Les flèches sur la trajectoire claire montrent où le robot a appris chaque transition entre son orientation précédente et son orientation courante. Plus la distance entre deux changements d'orientation est grande, plus le timing entre deux transitions est long. Les flèches foncées montrent où le robot à déclenché les prédictions lui permettant de prendre l'orientation correspondante.

du robot. Cette dérive s'explique par la différence du contrôle du robot par le professeur et celle du robot lui même. Durant l'apprentissage, le professeur influe sur la dynamique du robot en lui imposant de tourner. Lors de ce processus, le professeur modifie l'orientation du robot jusqu'à ce que le robot soit dans la bonne direction. Ce temps passé à tourner fait alors partie du timing de la nouvelle transition qui va être apprise. Durant la phase de reproduction, le robot va déclencher la prédiction de la prochaine direction à prendre. Il prédit donc une position angulaire cible qu'il doit atteindre. La dérive est donc due à la différence entre le timing appris qui inclut le temps de rotation du robot et celui qui est prédit auquel il faut ajouter le temps de rotation du robot (en plus du temps de rotation qui a été appris) suite à cette prédiction.

4.3 Conclusion

Dans ce chapitre, j'ai présenté deux architectures permettant d'apprendre une même tâche de navigation. La première permet d'encoder le comportement sous forme d'associations lieux-mouvements. Les expériences ont montrées que la précision de la trajectoire est de l'ordre de l'étendue des réponses des cellules de lieux. La trajectoire est donc directement dépendante de leurs profils. Cependant, la résolution et le nombre de cellules de lieux peuvent-être changés [Giovannangeli *et al.*, 2006, Giovannangeli, 2007]. Travaillant en environnement ouvert, la trajectoire du robot peut facilement être modifiée par une personne passant prés du robot (évitement d'obstacle). La seconde architecture encode la trajectoire sous forme de séquences temporelles de mouvements. Dans le cadre de l'application de l'apprentissage de séquences temporelles complexes appliquée à une tâche de navigation, j'ai proposé un mécanisme de resynchronisation des dynamiques internes à partir de signaux externes. Ce mécanisme permet alors de retrouver les états cachés précédemment appris. Ceci permet également d'amorcer une séquence par un état au milieu, ou même d'apprendre plusieurs séquences. Avec une telle architecture, le robot est

sensible aux perturbations (pas d'évitement d'obstacles). L'ajout d'une intégration de chemin pourrait permettre au robot de revenir sur la trajectoire en cas de perturbations. Cependant, un mécanisme de resynchronisation de la séquence sensori-motrice reste nécessaire pour que le robot puisse retrouver la séquence sensori-motrice et correctement restituter la trajectoire.
Chacun des deux modèles s'inspirent de propriétés de la boucle hippocampique et permettent à un robot d'apprendre un même comportement suivant une stratégie spatiale ou temporelle. Cette inspiration commune soulève alors la question de la cohabitation de ces stratégies dans un seul modèle pour permettre à un robot d'apprendre et de restituer un comportement spatio-temporel.

Chapitre 5

Fusion des comportements

Dans le cadre de ma thèse, j'ai développé un modèle permettant d'apprendre des séquences gestes sur un robot Aibo[1], ainsi que des séquences de déplacements sur un robot mobile Robulab10[2] équipé d'une boussole électronique jouant le rôle de proprioception. J'ai également développé un modèle de navigation qui permet à un robot de se déplacer en associant des lieux à des mouvements.
Les modèles développés s'inspirent tous les deux de structures du cerveau et plus particulièrement de la boucle hippocampique et du cervelet. Ayant une source d'inspiration commune, comment ces deux modèles peuvent-ils cohabiter dans un seul et unique système ? Exécuter ces deux mécanismes (séquences et associations sensori-motrices) en parallèle pose également la question de la fusion et/ou de la sélection des réponses de chacun. En effet, alors qu'à chaque instant chacun peut délivrer une réponse correspondant à un mouvement à réaliser, comment une architecture peut-elle permettre à un robot de réaliser un comportement cohérent ?
La sélection de l'action est un mécanisme dont le rôle est de choisir l'action à réaliser parmi un ensemble d'actions possible en fonction d'un objectif donné [Tyrrell, 1993, Girard *et al.*, 2002, Girard *et al.*, 2005]. On peut distinguer différents types de travaux permettant la sélection de l'action. Il y a les travaux qui se basent sur la construction et l'utilisation d'un plan permettant alors de planifier ses actions en fonction d'un but à atteindre. Dans le cadre d'études dans une tâche de navigation, les plans sont représentés sous la forme de cartes représentant plus ou moins explicitement l'environnement. On peut alors distinguer les cartes métriques et les cartes topologiques [Meyer et D., 2003]. D'autres travaux proposent des modèles réposant sur un ensemble de comportements hierarchisés ayant chacun des priorités [Brooks, 1986]. Enfin des modèles proposent d'apprendre par renforcement les bonnes actions prédites pour une situation particulière [Khamassi *et al.*, 2006].

5.1 Subsomption

Une approche dite classique de l'intelligence artificielle repose sur une décomposition en modules chaînés traitant les flux sensoriels en série. Les flux sensoriels sont récupérés par un module de perception qui sont ensuite transmis à un module de représentation interne de l'environnement, puis à partir de cette représentation, un module de plannification détermine l'action à réaliser qui est donné à un module de contrôle des degrés de liberté du robot. Par conséquent, plus le comportément désiré est complexe, plus les traitement réalisés sur les flux sensoriel deviennent long et complexes. Les architectures de subsomption proposent une structure hiérarchique de modules comportementaux simple et en parallèle. Les différents niveaux hierarchiques ont chacun une priorité sur les autres garantissant ainsi la viabilité du système. Chaque module reçoit les flux sensoriels et chacun délivre une réponse sur l'action à réaliser. Les calculs réalisés sur les flux sensoriels sont extrêmement simple et rapide permettant de respecter des contraintes de temps, tout en permettant à un robot d'exhiber des comportements complexes.
Dans [Brooks, 1986], l'auteur décompose le comportement d'un robot mobile en plusieurs niveaux de comportements parmi lesquels :
– 0 : évitement d'obstacles
– 1 : naviguer sans but/motivation particulière
– 2 : explorer l'environnement en se dirigeant vers des lieux visible
– 3 : Construire une carte de l'environnement et planifier les trajectoires d'un lieu à un autre.
– 4 : Notifier de changements dans l'environnement "statique"

[1] Robot chien de Sony
[2] Plateforme mobile de Robosoft

Dans ces travaux, ces différents comportements sont hiérarchisés du niveau 0 (bas niveau, comportement réflexe) au niveau n (haut niveau) comme le montre la figure 5.1 tiré de l'article de l'auteur.

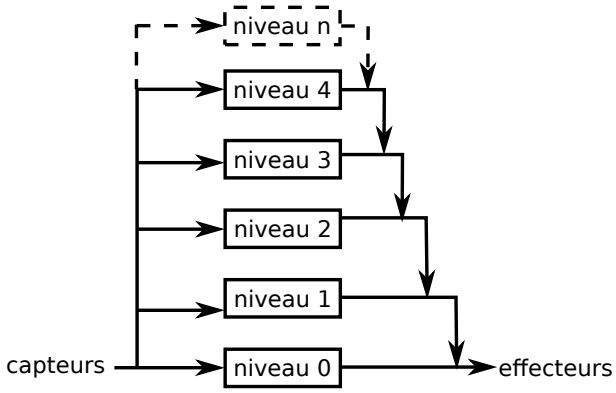


Fig. 5.1: Schémas d'architecture de subsomption. Les comportements sont organisés en hiérarchie. Ceux de bas niveau on des priorités plus hautes que ceux de haut niveaux.

Les comportements de bas niveaux ont alors une priorité plus forte que ceux de haut niveau. En effet, à choisir entre entrer en collision avec un obstacle et continuer à explorer l'environnement, il est préférable pour le robot d'éviter l'obstacle.
Ce type de modèle implique de définir a priori les priorités des comportements sur les autres. Dans le contexte d'étude d'un modèle développemental pour un robot, je ne fais pas d'hypothèse sur la priorité d'une boucle sensori-motrice sur une autre. Je pars de l'hypothèse que le processus de sélection de l'action est appris pendant le développement. Néanmoins, ce type d'architecture offre des éléments intéressants sur les aspects de temps réel et de contraintes de temps de réaction que nous avons introduit dans notre simulateur (voir chapitre 6)

5.2 Les ganglions de la base

Les ganglions de la base sont un ensemble de noyaux présents dans le cerveau de la plupart des vertébrés. Cette structure est impliquée dans les fonctions du contrôle moteur, de la cognition et émotionnelles [Cohen et Frank, 2009].
Les ganglions de la base se composent du striatum, du pallidum, de la substance noire ainsi que des noyaux sous-thalamique (STN) (figure 5.2). Le striatum est composé du noyau caudé et du putamen. Le pallidum est formé du globus pallidus interne (GPi) ou médian, et du globus pallidus externe (GPi) ou latéral. La substance noire est composée de la substance noire compacte (SNc) et de la substance noire réticulée (SNr). La principale entrée des ganglions de la base est le striatum qui reçoit de nombreuses connexions du cortex cérébral.
Les ganglions de la base sont organisés en boucles parallèles [Alexander *et al.*, 1986, McHaffie *et al.*, 2005] qui prennent en charge les fonctions oculomotrices, motrices, associatives et limbiques. Le circuit moteur est le plus connu et il comporte deux circuits. (figure 5.3). Le premier est le circuit direct cortex→striatum→GPi→thalamus→cortex. Le second est le circuit indirect cortex→striatum→GPe→STN→GPi→thalamus→cortex. La différence entre ces deux circuits est que le circuit indirect passe par les noyaux sous-thalamique avant d'arriver sur GPi. Le circuit direct aurait alors une fonction de sélection de l'action en désinhibant l'action à réaliser.

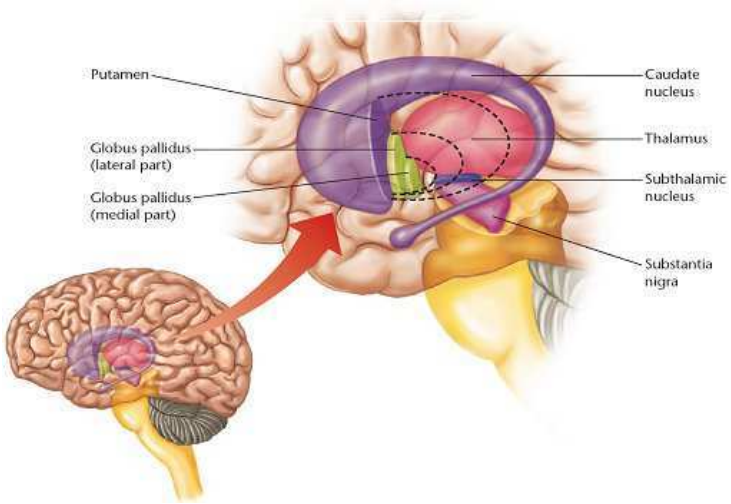


Fig. 5.2: Illustration des ganglions de la base dans le cerveau ainsi que ces différents composants. Image Marc Savasta (INSERM Grenoble)

Néanmoins, le circuit indirect joue également un rôle en modulant le circuit direct au niveau du GPi à travers les noyaux sous-thalamique.

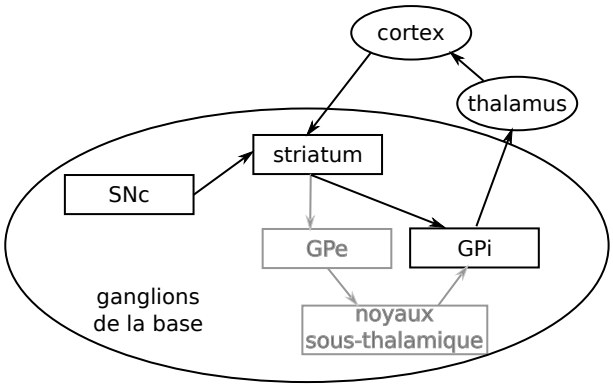


Fig. 5.3: Modèles mettant en avant les circuits direct et indirect dans les ganglions de la base. A) Le circuit direct est composé d'une boucle impliquant le cortex, le striatum, le GPi, le thalamus et reviens vers le cortex. B) Le circuit indirect est composé d'une boucle impliquant le cortex, le striatum, le GPe, le STN, le thalamus et reviens vers le cortex.

Composée principalement de neurones dopaminergétiques, la substance noire SNc jouerait un rôle de renforcement des actions motrices grâce à ces connexions excitatrices et inhibitrices sur le striatum. La sortie principale des ganglions de la base est le GPi dont les connexions vont sur le thalamus qui projette lui même ses connexions vers le cortex moteur.

5.3 Acteur-critique

Dans les travaux de Schultz réalisés sur le singe, l'auteur a mis en évidence que les neurones dopaminergétiques déchargent d'une manière similaire à l'algorithme d'apprentissage *TD* [Schultz, 1998]. En neurosciences computationnelles, les ganglions de la base ont souvent été modélisés

par un modèle appelé *Acteur-Critique* [Joel *et al.*, 2002]. Comme son nom l'indique, ce modèle se compose de deux parties principales : la partie *acteur* et la partie *critique* (figure 5.4). La partie acteur représente l'ensemble des commandes motrices pouvant être exécutées. Quant à la partie critique, elle a la fonction de fournir un signal de renforcement (positif ou négatif) qui permet d'évaluer les actions réalisées et ainsi sélectionner les actions motrices appropriées.

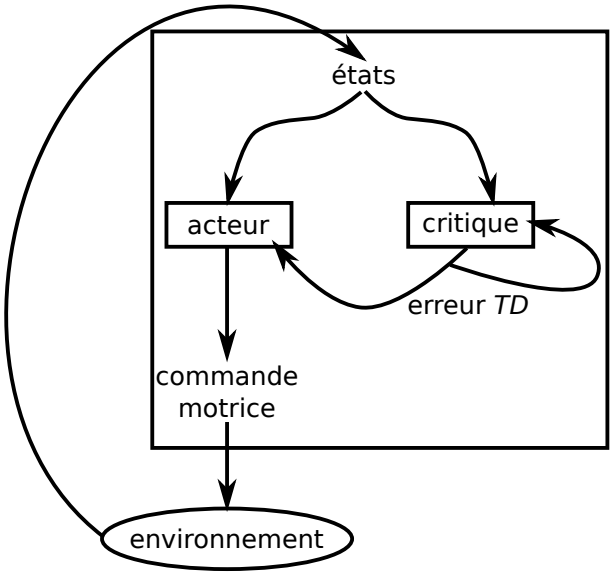


Fig. 5.4: Le modèle acteur-critique comporte deux parties. L'acteur propose les différentes actions à exécuter à partir d'états arrivant d'autres structures comme l'hippocampe à travers le subiculum. L'évaluateur ou critique reçoit également les états, mais permet de prédire quelle sera la récompense pour chaque action potentiellement exécutable. L'estimation de la récompense va alors permettre de sélectionner quelle action sera réalisée menant à la récompense la plus forte.

A partir de travaux mettant en évidence la présence de deux types de récepteurs de dopamines *D1* et *D2*, Mehdi Khamassi propose un modèle des ganglions de la base qui repose sur le modèle *acteur-critique* [Girard, 2003, Khamassi *et al.*, 2005]. Mais à la différence d'autres modèles, ici le modèle est composé de plusieurs modules acteurs et plusieurs modules critiques. Un module acteur reçoit des saillances de différents senseurs. Ces saillances permettent ensuite de déclencher différents comportements moteurs possible. Les sorties de ce type de module sont alors les différentes actions qui peuvent être réalisées. Quant aux modules critiques, ils fournissent aux modules acteurs la prédiction du signal de renforcement sur l'action sélectionnée. Les modules critiques sont des experts spécialisés dans des actions particulières. Chacun d'entre eux fournit un renforcement, puis ces signaux de renforcement sont additionnés pour fournir un renforcement global. Chacune des prédictions du renforcement des experts est pondérée par une valeur de crédibilité variant au fur et à mesure des expériences. Par conséquent, un expert qui au départ à une forte crédibilité, permettra de sélectionner une action particulière à réaliser. Si l'expert se trompe plusieurs fois, alors il perdra en crédibilité et son influence sur le choix de l'action diminuera. Par conséquent, d'autres actions pourront être sélectionnées. Les différents modules acteurs ont également ce même mécanisme de crédibilité sur les actions que chacun sélectionne. D'une certaine manière, dans ces travaux, l'utilisation de plusieurs modules acteurs et critiques permet de prendre en compte que plusieurs réponses peuvent être délivrées par d'autres structures. Mais ici, les sensations et les actions sont des symboles qui décrivent des sensations et

actions de hauts niveaux comme “voit du blanc”, “boire”, etc.
Dans [Dolle *et al.*, 2008], à partir d'une expérience réalisée sur le rat [Pearce *et al.*, 1998], les auteurs analysent comment deux stratégies de navigation peuvent être apprises en parallèle. La première stratégie repose sur une carte de l'environnement. La seconde repose sur des associations sensori-motrices. Pour tester la compétition et/ou la coopération de ces stratégies, les auteurs simulent un robot dans un environnement artificiel carré. Des amers sont placés dans l'environnement permettant ainsi de localiser quatre plateformes que le robot doit atteindre. Lorsque le robot atteint la plateforme désirée, alors il reçoit un renforcement positif. Les tests sont organisés en quatre sessions. Durant la première session, les deux stratégies sont actives. Pendant la seconde session, seule la stratégie sensori-motrice est active et lors de la troisième session, seule la stratégie qui repose sur une carte est active. Les auteurs montrent alors que les deux stratégies sont en compétition en début de session lorsque la plateforme cible à rejoindre change. Néanmoins, le modèle met en évidence une coopération des deux stratégies durant l'apprentissage de la plateforme cible. Les résultats mettent en évidence que la stratégie sensori-motrice est généralement préférée pour permettre de se diriger dans une direction générale, mais une fois proche de la plateforme, la stratégie qui repose sur une carte prend le dessus. Ici, cette étude porte sur l'interaction entre deux stratégies de même nature, car les deux apprennent des informations spatiales. Le travail que je présente a consisté à étudier comment deux stratégies sensori-motrices spatiales et temporelles se comportent lors de l'apprentissage et de la reproduction d'une tâche de navigation.

5.4 La boucle hippocampique

Les modèles d'apprentissage de séquences temporelles ainsi que l'association de lieux-mouvements sont tous les deux inspirés de l'hippocampe. Par conséquent, ces deux boucles sensori-motrices devraient pouvoir être exécutées dans un même modèle. D'une manière générale, le modèle proposé est composé de trois principales parties : la vision, l'hippocampe et le sensori-moteur (figure 5.5). La vision extrait des vues locales en coordonnées log-polaire, fusionne l'identité des vues locales (*what*) et leurs azimuts (*where*). L'hippocampe réalise une reconnaissance multimodale des entrées et apprend des transitions d'évènements multimodaux. Le niveau sensori-moteur permet d'associer les réponses provenant de l'hippocampe avec les informations proprioceptives permettant de fournir la commande motrice à réaliser.
L'apprentissage de séquences temporelles et l'apprentissage d'associations lieux-mouvements répondent en partie à ce modèle. En effet, la reconnaissance de lieux peut être réalisée par le cortex enthorinal en entrée de l'hippocampe et par le gyrus dentelé. Ces lieux peuvent être construits à partir d'informations visuelles. Ils sont ensuite associés aux informations proprioceptives au niveau sensori-moteur permettant ainsi de délivrer la commande motrice lors de la reproduction (figure 5.6).
L'apprentissage de séquences temporelles permet à un robot d'apprendre des séquences de gestes et des séquences de déplacements. Ces séquences sont donc codées par des transitions sensori-motrices. A partir d'informations proprioceptives, cette boucle sensori-motrice permet de prédire le mouvement suivant à réaliser. Dans mes travaux, les associations transitions-mouvements sont directement représentées par la topologie du groupe des transitions (figure 5.7).
Finalement, on se rend facilement compte que les deux boucles sensori-motrices (séquences et associations lieux-mouvements) peuvent s'exécuter en parallèle. L'idée est de simuler les deux stratégies d'hippocampe (figure 5.8). Chacune des stratégies permet alors d'encoder une tâche de navigation en apprenant différentes propriétés. Les associations lieux-mouvements permettent

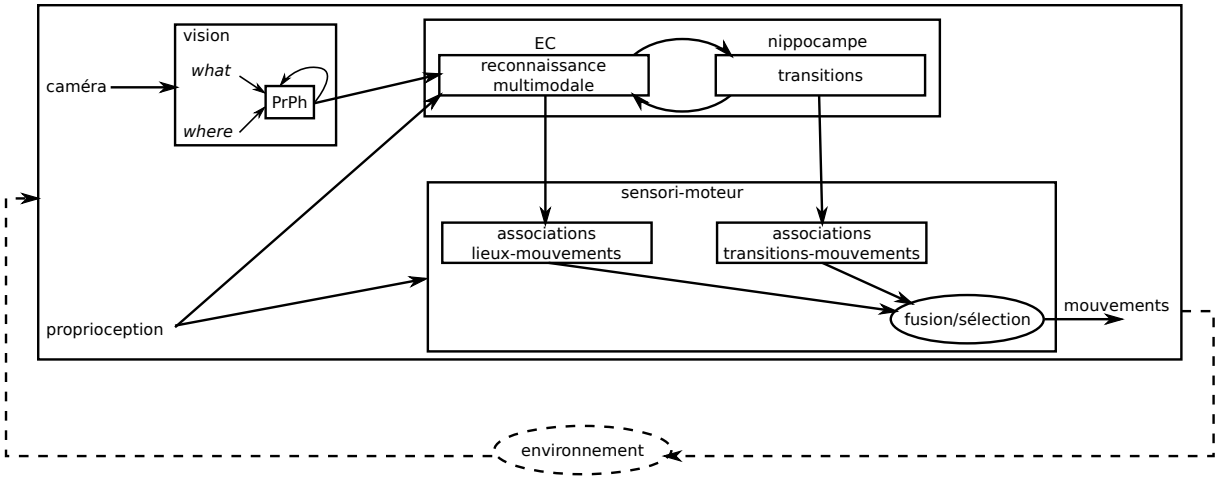


Fig. 5.5: Modèle général permettant l'apprentissage d'associations sensori-motrices et l'apprentissage de séquences. Ce modèle est composé du traitement bas niveau de la vision, de l'hippocampe permettant la reconnaissance multimodale, de l'apprentissage de transitions et d'un niveau sensori-moteur qui associe les réponses délivrées par l'hippocampe aux informations proprioceptives. Ces associations permettent de délivrer les mouvements à réaliser.

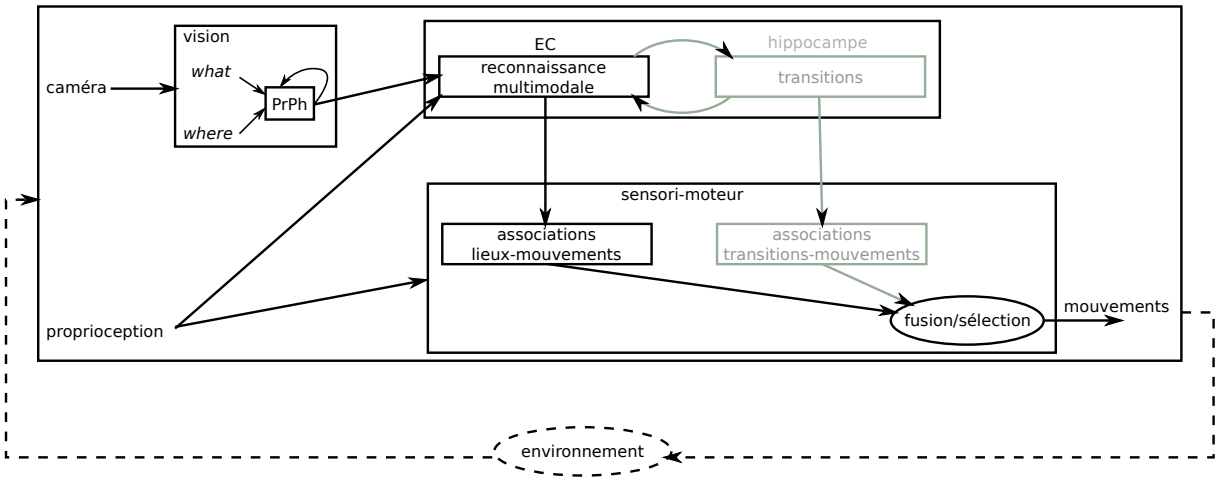


Fig. 5.6: Modèle permettant l'apprentissage d'associations sensori-motrices. Ce modèle est composé du traitement bas niveau de la vision, d'une partie de l'hippocampe permettant la reconnaissance de lieux et d'un niveau sensori-moteur qui associe les lieux reconnus par l'hippocampe aux informations proprioceptives. Ces associations permettent de délivrer les mouvements à réaliser. Les blocs grisés sont les parties du modèle général qui ne sont pas utilisés par l'apprentissage d'associations lieux-mouvements

d'apprendre des propriétés spatiales. L'apprentissage de séquences permet d'apprendre des propriétés temporelles. Le comportement est alors encodé sous forme de dynamiques spatiales d'un coté, et sous forme de dynamiques temporelles de l'autre.
Comment ces stratégies sont-elles utilisées ? Ces stratégies fournissent chacune leurs propres réponses quant aux mouvements à réaliser. Comment ces réponses sont-elles traitées ? Comment réaliser la fusion ?

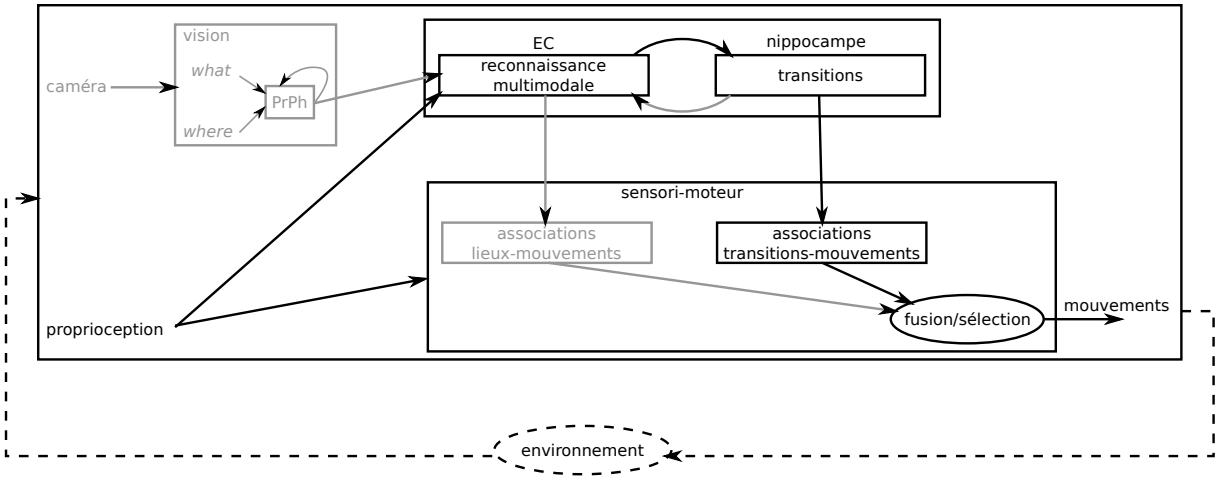


Fig. 5.7: Modèle permettant l'apprentissage de séquences temporelles. Ce modèle est composé de l'hippocampe qui reçoit en entrée les informations proprioceptives et qui permet l'apprentissage de transitions et d'un niveau sensori-moteur qui associe les transitions reconnues par l'hippocampe aux informations proprioceptives. Ces associations permettent de délivrer les mouvements à réaliser. Les blocs grisés sont les parties du modèle général qui ne sont pas utilisés par l'apprentissage de séquences.

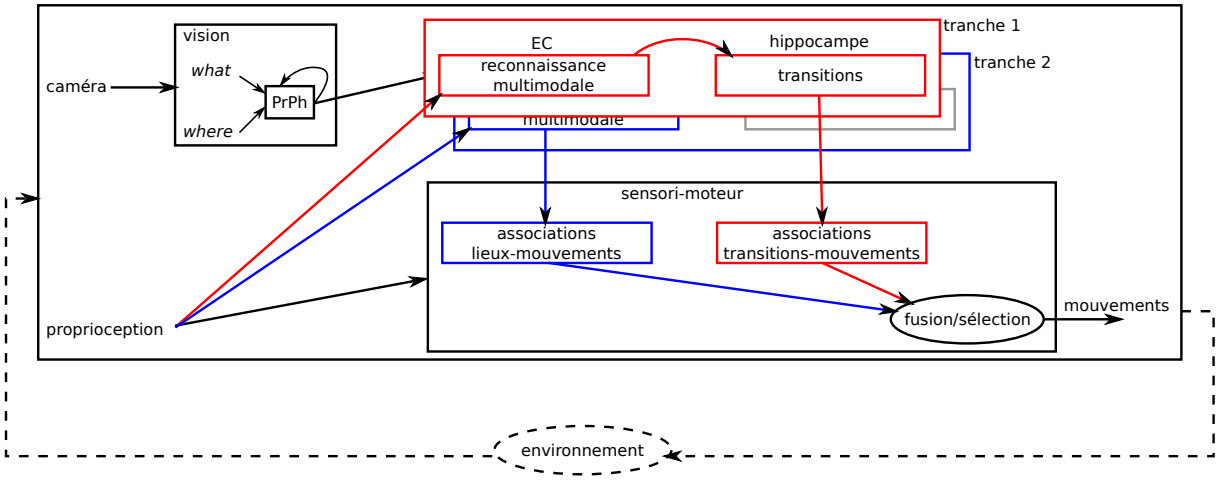


Fig. 5.8: Modèle permettant l'apprentissage de séquences temporelles et l'apprentissage d'associations lieux-mouvements en parallèle. Ce modèle est composé de deux "tranches" d'hippocampe. La première tranche permet d'apprendre les propriétés temporelles d'une tâche. La deuxième tranche permet quant à elle l'apprentissage de propriétés spatiales.

5.5 Le champ de neurones dynamiques

Les champs de neurones dynamiques [Amari, 1977] ont des propriétés qui permettent la fusion et/ou la sélection d'actions. Dans mon travail, les champs de neurones codent l'espace des mouvements et ils sont calculés suivant l'équation :

$$\tau \frac{dv(\phi,t)}{dt} = -v(\phi,t) + S(\phi,t) + h + \int_{-\infty}^{+\infty} W(\phi-\phi').v(\phi',t).d\phi' \qquad (5.1)$$

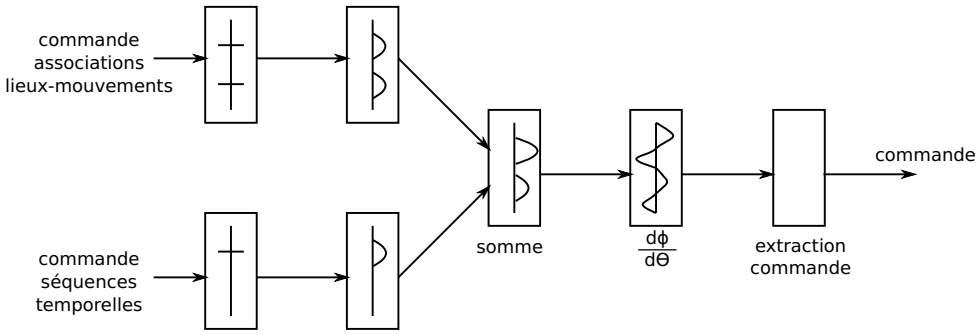


Fig. 5.9: Modèle de fusion/sélection des commandes de deux stratégies. Les stratégies d'associations lieux-mouvements et de séquences temporelles fournissent chacune une commande à exécuter. Les deux commandes sont des positions sur un champ de neurones. Elles sont ensuite sommées dans un champ de neurones dynamiques. Puis une dérivée spatiale est réalisée sur le champ et finalement un mécanisme d'extraction de la commande finale envoie la commande en vitesse à réaliser. La commande dépendra à la fois de l'activité du champ et de la position de l'effecteur dans l'espace de la commande.

avec en entrée $S(\phi, t)$ qui apporte de l'énergie au champ de neurones. Le champ réalise une diffusion spatiale avec un noyau d'interaction $W(\Delta\phi)$ et il intègre temporellement son activité grâce au premier terme. Dans mes travaux, le noyau d'interaction utilisé est une différence de Gaussiennes (figure 5.10.B). Ce noyau permet la fusion d'entrées proches grâce à sa partie positive et la sélection d'entrées éloignées grâce à ses parties négatives.

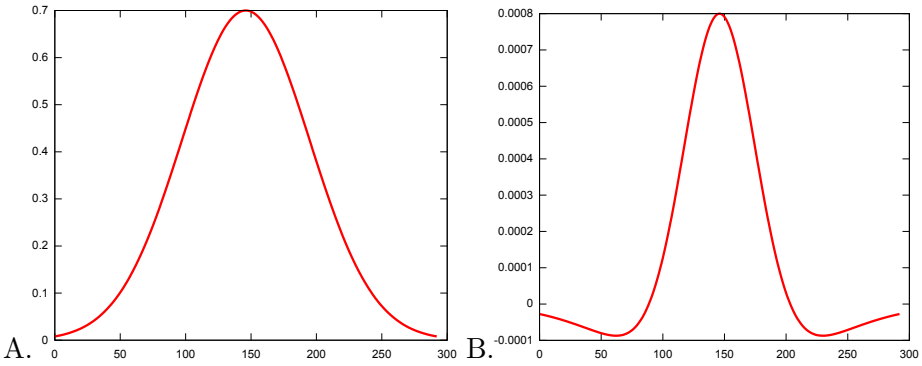


Fig. 5.10: A) Gaussienne utilisée pour créer un attracteur autour d'une position particulière. B) Différence de gaussienne utilisée comme noyau d'interaction dans le champ de neurones dynamiques.

5.6 Mécanisme d'extraction de la commande motrice

Pour obtenir la commande finale à appliquer à partir d'un champ de neurones, j'utilise un mécanisme d'extraction [Schöner *et al.*, 1995] dont la commande est une vitesse à appliquer au moteur correspondant. Ce mécanisme est exactement le même que celui utilisé dans le chapitre 2 avec le robot Aibo[3]. Une dérivée spatiale est réalisée sur la sortie du champ de neurones dynamiques (figure 5.11). Puis, à partir de la position actuelle du robot, l'activité du neurone

[3]Robot chien de Sony

correspondant sur le champ dérivé est extraite. Cette activité est alors une commande motrice en vitesse à réaliser.

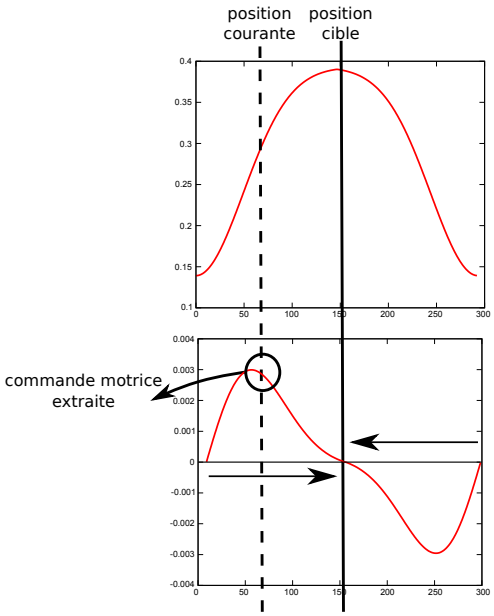


Fig. 5.11: Exemple d'extraction de la commande motrice à partir de la sortie du champ de neurones dynamiques. L'activité du champ de neurone (courbe à gauche) est centrée sur la position à atteindre (ligne noire continue). Ensuite, une dérivée spatiale (courbe à droite) est réalisée sur cette activité. A partir de la position courante (ligne noire en pointillés), alors la commande motrice est extraite (encerclée en noire) de la dérivée spatiale.

Le même mécanisme est donc utilisé lors de la présence de deux entrées (figure 5.12). Lorsque deux entrées proches sont présentes, alors comme je l'ai montré précédemment, elles sont fusionnées en un seul attracteur. Dans ce cas, le mécanisme d'extraction de la commande motrice est similaire à celui présenté au-dessus.
Quand les deux entrées sont suffisamment distantes, c'est à dire au-delà de Δ, alors deux attracteurs sont en compétition. De la même manière, une dérivée spatiale est réalisée sur l'activité du champ de neurones. Ensuite, à partir de la position courante, la commande motrice est extraite de cette dérivée. On remarque alors que la convergence ou non vers un attracteur est dépendante de la distance qui nous en sépare. Le robot convergera vers l'attracteur le plus proche. C'est donc cette distance par rapport aux attracteurs qui permet la sélection de l'action à réaliser.

5.7 Tests des champs de neurones dynamiques

Les tests sont réalisés en simulation avec les paramètres qui sont par la suite utilisés sur le robot. Le premier test permet de mettre en évidence le fonctionnement du champ de neurones dynamiques. Il y a une seule entrée active. L'activité d'entrée change de position toutes les 20 itérations (figure 5.13.A). Durant cette période, l'entrée est maintenue active. A chaque instant, l'entrée est convoluée avec une gaussienne (figure 5.10.A) qui permet de créer un attracteur autour de la position d'entrée (figure 5.13.B). L'activité de cet attracteur est ensuite traitée par le champ de neurones dynamiques (figure 5.13.C).
On observe alors que lorsque l'entrée change de position, alors l'activité de l'attracteur sur le champ de neurones dynamiques se déplace en conséquence. Grâce aux propriétés dynamiques,

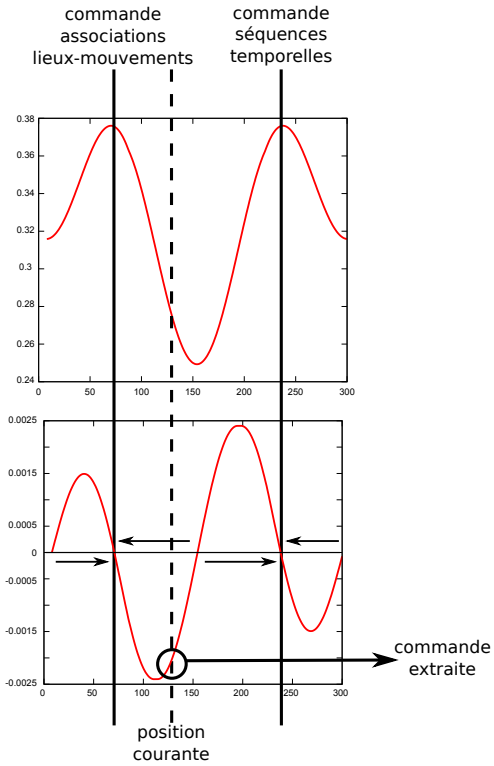


Fig. 5.12: Exemple d'extraction de la commande motrice à partir de la sortie du champ de neurones dynamiques. L'activité du champ de neurone (courbe à gauche) forme deux attracteurs centrés sur les commandes fournies par les deux stratégies association lieux-mouvements et séquences temporelles (lignes noires continues). Ensuite, une dérivée spatiale (courbe à droite) est réalisée sur cette activité. A partir de la position courante (ligne noire en pointillés) la commande motrice est extraite (encerclée en noire) de la dérivée spatiale. La direction vers laquelle la position converge est alors celle qui mène à l'attracteur le plus proche.

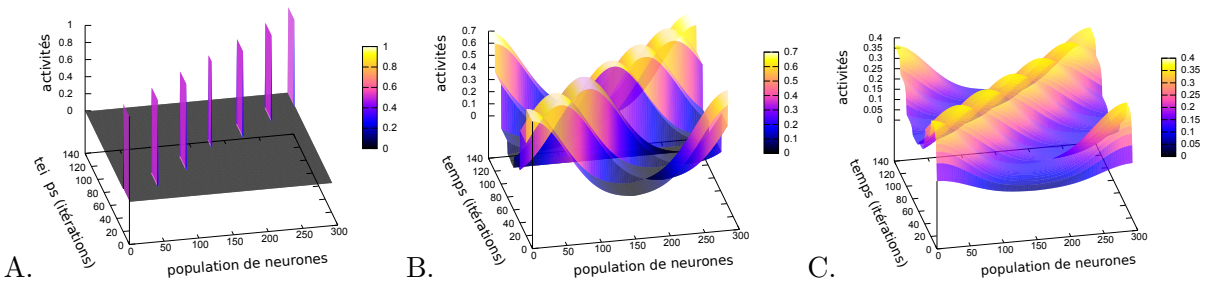


Fig. 5.13: A) L'activité d'entrée change de position toutes les 20 itérations. Durant cette période, l'entrée est maintenue active. B) A chaque instant, l'entrée est convoluée avec une gaussienne qui permet de créer un attracteur autour de la position d'entrée. C) L'activité de cet attracteur est ensuite traitée par le champ de neurones dynamiques. La "bulle" d'activité du champ de neurones suit le déplacement de l'entrée.

le déplacement ne se fait pas brusquement, mais l'attracteur "glisse" vers la nouvelle position. Dans le test réalisé ici, au départ les deux entrées sont à la même position. Puis elles s'éloignent progressivement chacune vers les extrémités opposées du champ de neurones (figures 5.14.A et 5.14.C). Comme le test précédent, les entrées sont convoluées par une gaussienne (figures 5.14.B et 5.14.D). Ces deux attracteurs sont alors ensuite sommés dans le champ de neurones

dynamiques (figure 5.14.E).

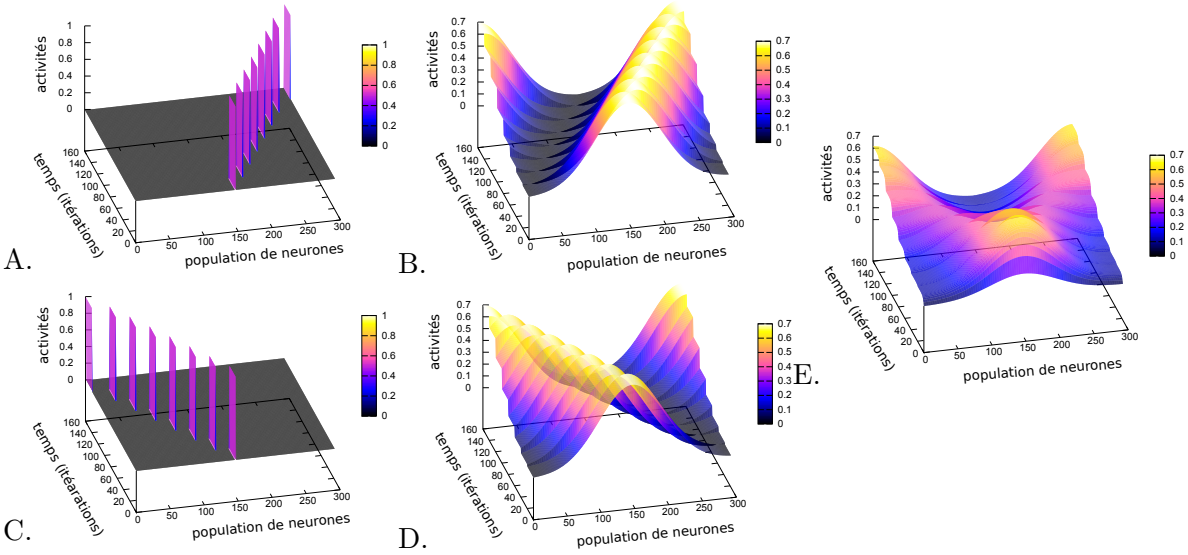


Fig. 5.14: A) et C) L'activité d'entrée de chaque stratégie change de position toutes les 20 itérations. Durant cette période, les entrées sont maintenues actives. B) et D) A chaque instant, les entrées sont convoluées avec une gaussienne qui permet de créer un attracteur autour de la position d'entrée. E) Les activités de chaque attracteur sont sommées dans un champ de neurones dynamiques.

Dans ce test, on observe comment deux entrées sont fusionnées ou séparées. Au début de la simulation, les deux entrées sont fusionnées en un seul attracteur sur le champ de neurones dynamiques. Malgré que les deux entrées commencent à se séparer, le champ garde toujours un seul attracteur d'activité. Quand les entrées deviennent distantes (bifurcation), alors l'attracteur sur le champ "éclate" en deux attracteurs qui correspondent aux deux entrées.
On a alors ici des propriétés de fusion et de sélection. En effet, lorsque deux entrées sont proches, alors le champ de neurones dynamiques les fusionne en un seul attracteur. Lorsque les entrées sont distantes, alors il y a deux attracteur. C'est finalement le mécanisme utilisé pour extraire la commande motrice qui défini vers quel attracteur le robot va converger.

5.8 Les actions du robot

Dans toutes les expériences présentées ici, les actions du robot sont des couples (directions, vitesse linéaire) (figure 5.15). La vitesse linéaire reste suffisamment faible pour permettre au robot de capturer son environnement avec la caméra en gardant des images nettes. Dans mes travaux, la vitesse linéaire est discrétisée sur trois neurones : vitesse positive, vitesse nulle et vitesse négative. De manière à ne pas avoir de conflit de commande en vitesse linéaire, l'architecture suit deux règles. La première est que la commande du professeur prend le dessus sur les deux autres stratégies. La seconde est qu'entre les deux stratégies, celle qui demande au robot d'avancer est prioritaire. Cette seconde règle est posée arbitrairement, mais ne perturbe pas l'étude, car notre travail porte essentiellement sur les changements de directions du robot.
Finalement, les actions apprises par le robot sont une matrice multiplicative entre la vitesse linéaire et les orientations du robot fournies par la boussole électronique (figure 5.16). Lorsque chacune des stratégies de navigation exprime une commande désirée, alors la commande effectuée

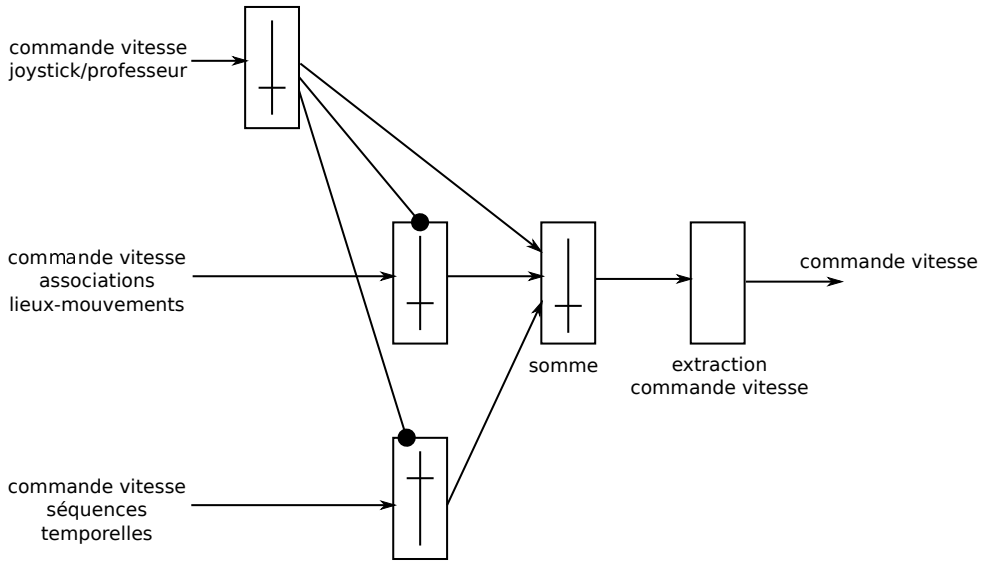


Fig. 5.15: Modèle de fusion des stratégies avec l'ajout de la gestion des commandes en vitesse linéaire. Les connexions se terminant par un cercle noir sont inhibitrices. L'architecture suit deux règles pour gérer les commandes conflictuelles. La première est que la commande du professeur prend le dessus sur les deux autres stratégies. La seconde est qu'entre les deux stratégies, celle qui demande au robot d'avancer est prioritaire.

est un élément de cette matrice qui est ensuite séparé en vecteur vitesse et vecteur d'orientations.

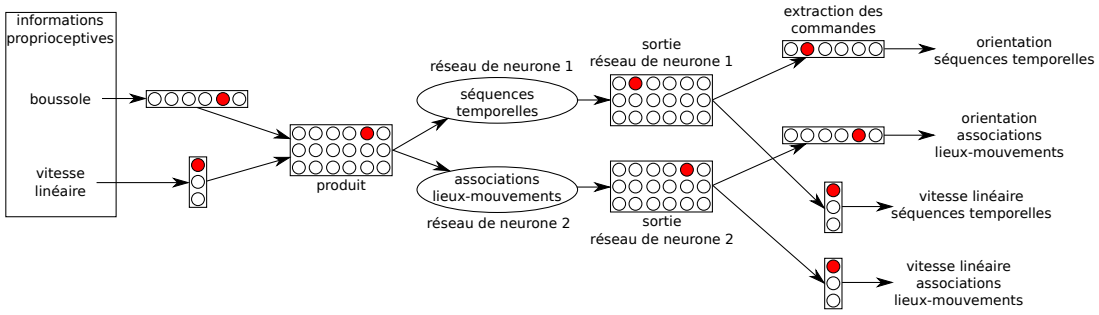


Fig. 5.16: Illustration du traitement des informations motrices. Les informations proprioceptives (boussole et vitesse linéaire) sont fusionnées dans une matrice multiplicative. Cette matrice est ensuite apprise dans les stratégies de séquences temporelles et d'associations lieux-mouvements. Lors de la restitution des commandes motrices par ces deux stratégies, chacune des matrices est séparée en deux vecteurs d'orientations et de vitesses linéaires. Ces différentes commandes sont alors traitées comme expliqué précédemment.

Dans mes travaux, la précision des orientations traitées par chacune des stratégies est différente. En effet, la stratégie de séquences temporelles sous échantillonne les orientations de manière plus importante que la stratégie d'associations lieux-mouvements. La raison de ce processus est le coût computationnel de la stratégie de séquences temporelles. En effet, la quantité de neurones codant les transitions d'états explose rapidement. Dans mon architecture, les orientations sur 360° sont codées sur 18 neurones multipliés par 3 neurones codant pour la vitesse linéaire. Donc ces 54 mouvements possible sont ensuite multipliés par le nombre d'états cachés permettant de lever les ambiguïtés des séquences, donc $54 * 3 = 162$ neurones codant les états cachés. Finalement, le groupe des transitions est le nombre d'états caché au carré, c'est à dire $162 * 162 = 26244$

transitions pour le groupe des transitions seulement. Il est alors difficilement imaginable de coder les orientations sur 360 neurones. Pour fusionner/sélectionner les commandes des deux stratégies sur un champ de neurones, les réponses doivent être au même format. C'est pour cela qu'en sortie de la stratégie de séquences temporelles les réponses sont suréchantillonnées.
Avant de tester l'execution en parallèle des deux stratégies de navigation, il reste à résoudre la question de la resynchronisation des dynamiques temporelles. Je propose dans la section suivante un mécanisme de "chunking" qui permet au robot de resynchroniser la séquence de ses déplacements par rapport à la détection d'événement complexe (le *chunk*).

5.9 Les *chunks*

Dans le chapitre 4, j'ai présenté un mécanisme de resynchronisation des dynamiques internes afin de permettre de retrouver les bons états cachés d'une séquence ambigüe. En plus de la resynchronisation, j'ai pu mettre en évidence que ce mécanisme permettait d'amorcer une séquence par un état intermédiaire et même d'apprendre et de restituer plusieurs séquences. Pour fonctionner correctement, ce mécanisme de resynchronisation dépend de signaux externes. Dans les simulations que j'ai présentées, ces signaux étaient fournis directement à l'architecture. Mais dans le cadre de la robotique autonome, quelle peut être la nature de ces signaux ?
Je me place ici dans le contexte de la navigation avec un robot mobile. Dans mes travaux, les états des séquences sont les orientations et la vitesse linéaire du robot. Ceci signifie que pour retrouver un état caché particulier, le robot doit être dans une orientation et une vitesse particulière au moment de la resynchronisation. Alors comment retrouver à quel état de la séquence correspond l'action en cours du robot ?
En exécutant en parallèle les deux stratégies de navigation (séquences et association lieux-mouvement), chacune des structures impliquées peut interagir avec l'autre. Je me suis alors concentré sur ce que pouvait apporter la stratégie d'associations lieux-mouvement aux séquences temporelles. Je propose qu'un mécanisme de *chunks* puisse coder l'état du robot à chaque instant. Ces *chunks* représentent alors un instantané de l'état du robot, qui joueront le role de signal de synchronisation.
En psychologie, le processus de "chunking" consiste à recoder une partie des états d'une séquence présente dans une mémoire à court terme en une seule unité [Simon, 1974]. Ici un *chunk* représente une sous séquence. Par exemple, considérons une série de chiffre "0745261998". On a ici une séquence de chiffre qui n'est pas forcément simple de mémoriser. Maintenant, représentons cette série d'une manière différente : "07 45 26 19 98". En regroupant deux à deux les chiffres, on obtient alors des nombres un peu plus simple à retenir. Ceci est surtout dû au fait que la séquence n'est plus une série de dix chiffres mais de cinq nombres. Ce regroupement de deux chiffres en un seul nombre correspond au processus de "chunking" et chaque nombre est alors un *chunk*. Dans [Grossberg, 1999], l'auteur utilise ce mécanisme pour encoder une sous-séquence d'entrées auditives présente en mémoire à court terme. D'une autre manière, dans [Luke *et al.*, 2005], les auteurs utilisent des *chunks* pour encoder un ensemble de points d'intérêts extrait de scènes visuelles dans la perspective de faire naviguer un robot en extérieur.
Dans les deux exemples cités ici, un *chunk* code soit un sous ensemble d'états d'une séquence, soit un sous ensemble de points d'intérêt. On peut alors définir un *chunk* comme une unité codant une situation particulière dans un comportement global.
Dans mes travaux, le robot doit naviguer d'un point de l'environnement à un autre. En utilisant la stratégie d'associations lieux-mouvements, le robot navigue de lieux en lieux en appliquant les commandes motrices qui y sont associées. Ces informations sont alors tout à fait pertinentes pour

définir la situation dans laquelle se trouve le robot par rapport à la tâche qu'il doit accomplir.

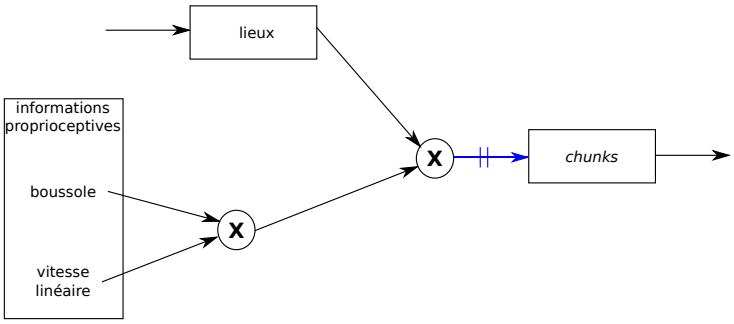


Fig. 5.17: Modèle d'apprentissage de *chunks* à partir des informations proprioceptives et du lieu courant. Chacun de ces *chunks* code pour une situation particulière du robot.

Les *chunks* sont donc construit à partir du lieu courant dans lequel se trouve le robot, ainsi qu'à partir des informations proprioceptives (figure 5.17). Ces informations sont fusionnées dans un tenseur d'états. Ce tenseur regroupe alors les différents états possibles dans lesquels le robot peut se trouver à chaque instant. Un *chunk* apprend alors ce tenseur suivant l'équation suivante :

$$\Delta\omega_{ec} = \Gamma_1\left(Act_e(t)\right) \cdot R_c(t) \tag{5.2}$$

avec Γ_1 une fonction rampe, ω_{ec} (initialisé à 0 et qui prend des valeurs binaire $0, 1$) le poids des connexions entre le e^{eme} neurone du tenseur d'états et le neurone c recruté qui code le *chunk*. Act_e est l'activité du e^{eme} neurone du tenseur d'état. R_c est un signal de recrutement du c^{eme} neurone. $R_c = 1$ *si* le neurone c est le neurone recruté, *sinon* 0.
L'activité C_c du c^{eme} *chunk* est exprimée comme suit :

$$C_c(t) = \sum_{e=1}^{nbetats} \omega_{ec}(t) Act_e(t) \tag{5.3}$$

avec $\omega_{ec}(t)$ le poids de la connexion entre le e^{eme} neurone du tenseur et le c^{eme} *chunk* et *nbetats* le nombre de neurones du tenseur d'états.
Les activités des *chunks* sont alors transmises au groupe de resynchronisation de séquence. En phase d'apprentissage, lorsqu'un *chunk* devient actif, il est alors associé à un neurone de resynchronisation. Cette association est alors réalisée suivant l'équation :

$$\Delta\omega_{cr} = \Gamma_1\left(Act_c\right) \cdot R_r \tag{5.4}$$

avec Γ_1 une fonction rampe, ω_{cr} (initialisé à 0 et qui prend des valeurs binaire $0, 1$) le poids des connexions entre le c^{eme} *chunk* et le neurone de resynchronisation r. Act_c est l'activité du c^{eme} *chunk*. R_r est un signal de recrutement du neurone de resynchronisation r. $R_r = 1$ *si* le neurone r est le neurone recruté, *sinon* 0.
Une fois l'apprentissage terminé, lorsqu'un *chunk* devient actif, le neurone de resynchronisation associé est alors actif et déclenche la resynchronisation de la séquence comme je l'ai décrit dans le chapitre 4.

5.10 Contrôle des stratégies

De manière à pouvoir étudier comment les différentes stratégies se comportent, une partie du modèle consiste à récupérer divers signaux, d'inhiber les réponses de l'une ou l'autre des stratégies et de déclencher ou non l'apprentissage sur une stratégie particulière. Cette partie de l'architecture a principalement pour entrée les boutons du joystick permettant ainsi au professeur d'agir directement sur le robot (figure 5.18).

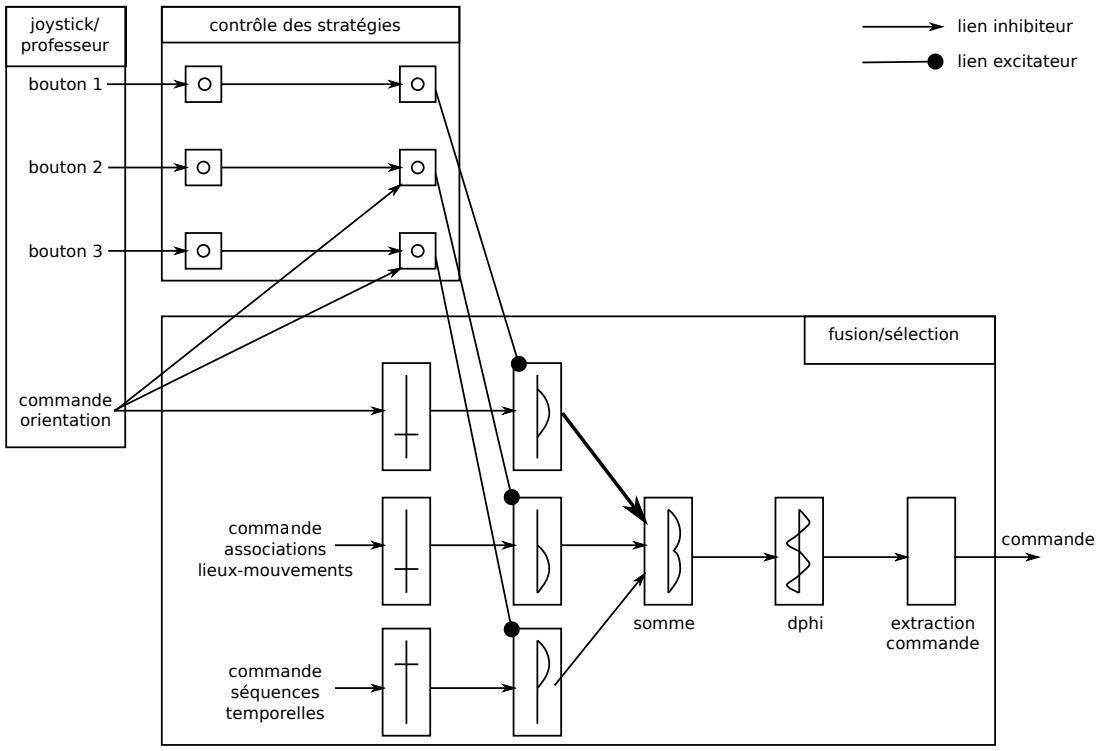


Fig. 5.18: Dispositif de contrôle ad hoc des différentes stratégies par l'utilisateur. Ce contrôle est réalisé par le professeur à partir de boutons permettant d'inhiber une ou plusieurs stratégies.

5.11 Tests de navigation avec deux stratégies en parallèle sur un robot mobile

Ces tests visent à mettre en évidence comment deux stratégies sensori-motrices peuvent coopérer et être en compétition dans une tâche de navigation. Le robot utilisé ici est une plateforme mobile Robulab10[4] équipée d'une caméra sur deux moteurs montés en *Pan-Tilt*. Le robot est également équipé d'une boussole électronique jouant le rôle d'information proprioceptive. Un joystick jouant le rôle d'une laisse est utilisé pour que le professeur apprenne au robot la trajectoire désirée (voir annexe 9.2 du dispositif complet). Le joystick agissant sur la dynamique du robot, il est fusionné dans le champ de neurones dynamiques avec un poids supérieur aux deux stratégies (figure 5.19).

[4]Plateforme mobile Robosoft

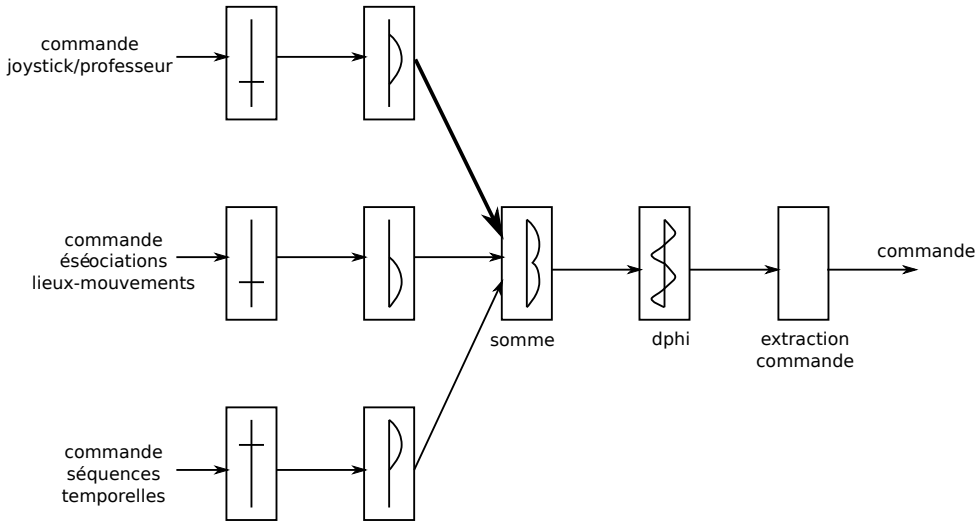


Fig. 5.19: Modèle de fusion des stratégies avec l'ajout du professeur (joystick) modifiant la dynamique du robot. La commande du professeur a un poids supérieur (flèche plus épaisse) de manière à ce que le professeur puisse imposer au robot la direction à prendre.

5.11.1 Test de la collaboration des stratégies de navigation spatiales et temporelles

Comme j'ai pu le montrer dans le chapitre 4, les stratégies spatiales et temporelles permettent à un robot corrigé par un professeur d'apprendre et de restituer une trajectoire. Nous voulons exécuter ces deux stratégies en parallèle dans une même architecture, nous faisons alors l'hypothèse qu'elles ont déjà été apprises. Mais ces stratégies peuvent-elles se développer en même temps ? La stratégie d'associations lieux-mouvements délivre une réponse quand un lieu appris est reconnu. Quant à la stratégie de séquences temporelles, elle prédit l'état suivant du robot. Nous avons alors une stratégie réactive (lieux-mouvements) et une proactive (séquences). On peut supposer que la stratégie réactive doit se développer en premier en apprenant les associations lieux-mouvements, puis que ces associations sont ensuite apprises sous forme de séquences. L'expérience qui suit va permettre de tester dans quelle mesure les deux stratégies peuvent être complémentaires pour une même tâche avec des conditions changeantes lorsqu'une stratégie est en défaut.
Dans l'expérience réalisée ici, le robot est kidnappé à différents endroits de l'environnement. A chaque endroit, le robot apprend alors un nouveau lieu qu'il associe à son action courante (figure 5.20). Dans ce test, le robot apprend quatre associations indépendantes les unes des autres. La figure 5.20.a montre les quatre associations apprises dans une salle. Les trajectoires représentent le mouvement (orientation et vitesse linéaire) associé à chacun des lieux (cercles noirs). Le cercle noir avec une croix signifie que l'action courante du robot est l'arrêt. La figure 5.20.b montre les activités des quatre cellules de lieux apprises.
Une fois les quatre associations apprises, le robot est alors kidnappé pour être placé sur le dernier lieu appris (lieux "D"). Le robot restitue alors la trajectoire de lieu en lieu "D B A C". Durant cette phase de rappel, le réseau apprend la succession temporelle des mouvements effectués par le robot à partir des informations proprioceptives (figure 5.21.a). Le robot apprend ainsi la séquence temporelle des actions (couple orientation, vitesse) qu'il est en train de réaliser. La figure 5.21.b montre les activités des cellules de lieux reconnues par le robot. Finalement, le

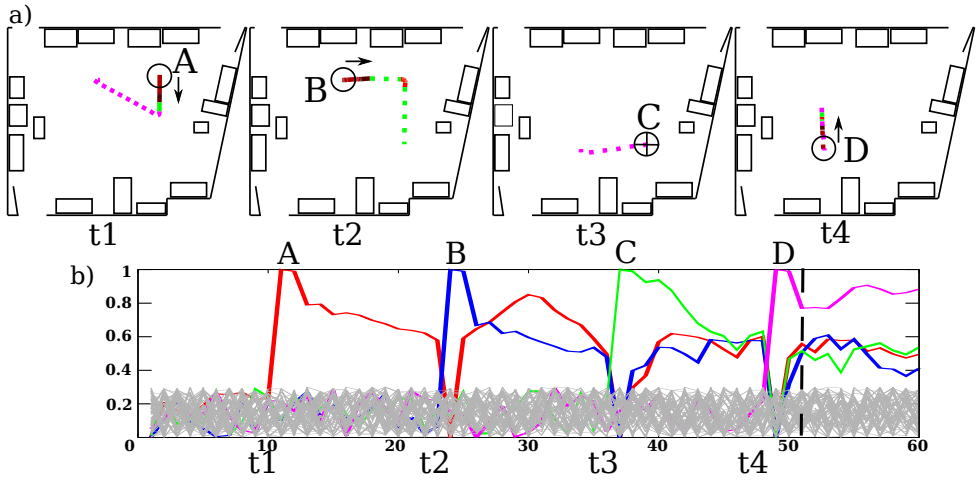


Fig. 5.20: Apprentissage de quatre associations lieux-mouvements indépendamment des unes des autres. A) Durant cette phase d'apprentissage, la stratégie de séquences temporelles n'apprend rien. Dans le lieu "A", le robot apprend à "avancer ver le bas", en "B" à "avancer vers la droite", en "C" à "s'arrêter" et en "D" à avancer vers le "haut". B) Activités des cellules de lieux correspondant aux quatre lieux appris.

robot est de nouveau kidnappé pour être remis sur le lieu "D". Un cache est placé sur la caméra du robot (figure 5.22).

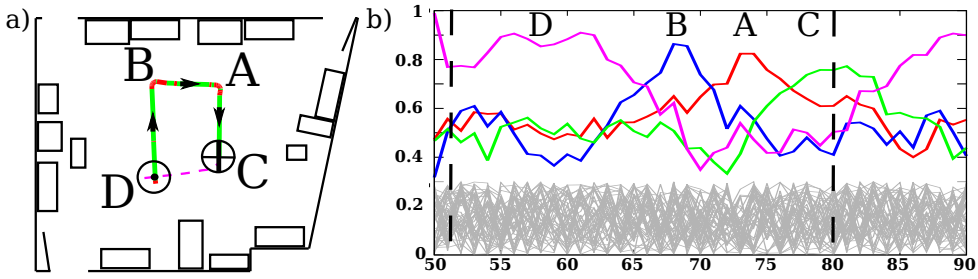


Fig. 5.21: Le robot est kidnappé pour être placé dans le lieu "D". A) Il exécute alors le mouvement associé à ce lieu "avancer vers le haut". Puis lorsqu'il reconnaît les lieux suivants, il exécute de la même manière les mouvements qui y sont associés ("avancer vers la droite", "avancer vers le bas" puis "s'arrêter"). Durant cette phase, le robot apprend à travers la stratégie de séquences temporelles, la succession des changements de mouvements grâce aux informations proprioceptives. B) Activités des cellules de lieux reconnues par le robot.

Ce cache empêche alors le robot de reconnaître visuellement les lieux. Par conséquent la stratégie d'association lieux-mouvements ne peut plus fournir de réponses. Le robot navigue en aveugle comme s'il était dans le noir (couper la lumière en pleine expérience aurait eu le même effet). La figure 5.23.a montre la trajectoire réalisé par le robot à partir de la stratégie de séquences temporelles seule. Les activités des transitions déclenchant les prédictions des mouvements suivants montrent que le robot réalise effectivement les quatre mouvements précédemment appris (figure 5.23.b).
Si cette expérience permet de tester comment une stratégie peut prendre le dessus sur l'autre, elle ne permet pas de voir ce qui va se passer en cas de conflit (stratégies fonctionnant en parallèle)

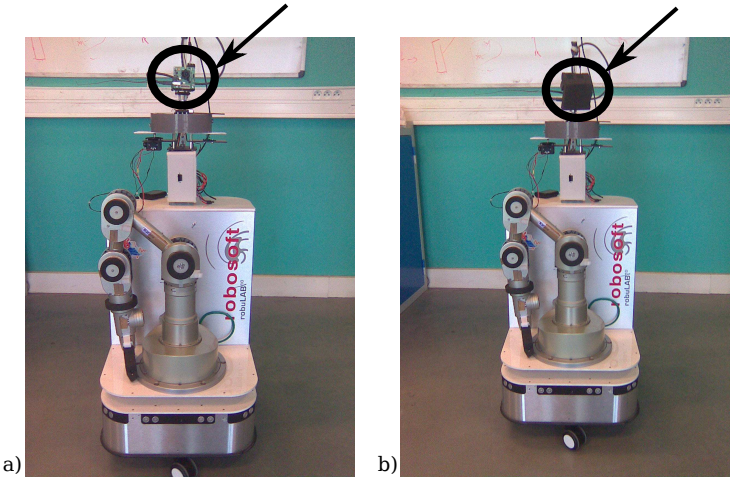


Fig. 5.22: Un cache est placé sur la caméra du robot de manière à ne pas permettre la reconnaissance de lieux. Le robot est donc aveugle, comme s'il devait se déplacer dans le noir. Par conséquent, seule la stratégie de séquences temporelles permet de restituer la trajectoire.

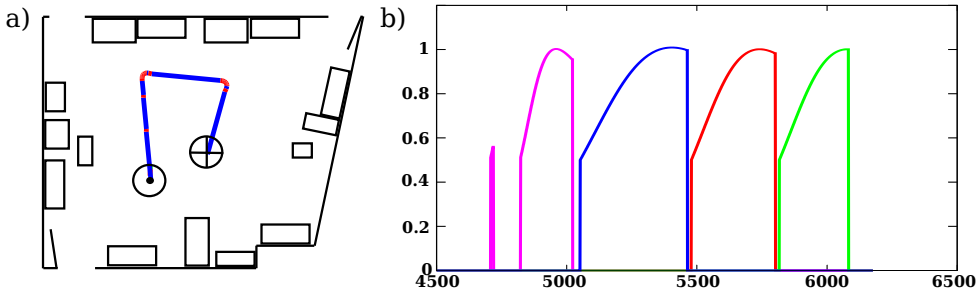


Fig. 5.23: Le robot est de nouveau kidnappé pour être placé dans le lieu "D". A) Trajectoire restitué par le robot avec la stratégie de séquences temporelles. B) Activités des transitions déclenchant (lorsque l'activité est à son maximum) successivement la prédiction des états suivants.

5.11.2 Test de la compétition des deux stratégies de navigation

Dans ce premier test, une approche complètement naïve a été adoptée face au comportement du robot. Durant l'apprentissage, la trajectoire est apprise aussi bien par la stratégie temporelle (séquences) que par la stratégie spatiale (associations lieux-mouvements). La trajectoire désirée est en forme de "U" (figure 5.24).
Dans un premier temps, le robot est guidé sur la trajectoire à apprendre. Le professeur modifie la dynamique sensori-motrice du robot à l'aide du joystick soit en le changeant d'orientation, soit en le changeant de vitesse. A chaque modification de sa dynamique sensori-motrice, le robot apprend alors le timing d'une nouvelle transition de la séquence de mouvements. En parallèle, cela déclenche également l'apprentissage d'un nouveau lieu, ainsi que d'une association lieu-mouvement. Dans le test présent, il y a au total quatre associations lieux-mouvements apprises. La figure 5.25 montre les différentes orientations prises par le robot durant la phase d'apprentissage. Ces informations sont directement extraites de la boussole électronique jouant le rôle d'informations proprioceptives et permettent de se rendre compte du mouvement réel du robot qu'il soit guidé par le professeur, qu'il navigue de manière autonome ou qu'il soit kidnappé.

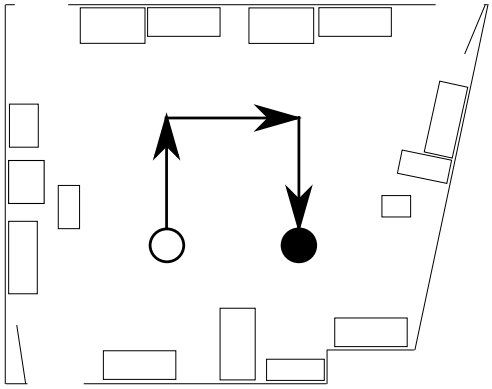
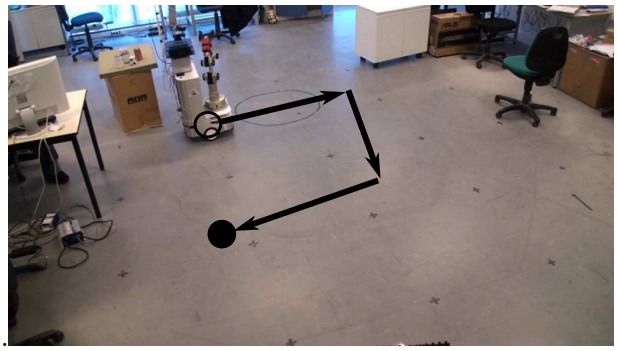
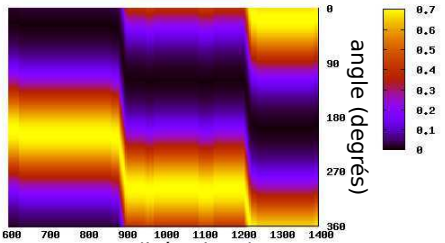


Fig. 5.24: Trajectoire désirée dans une salle d'expérience. Cette trajectoire est apprise par le robot. La trajectoire est composée d'un point de départ à l'arrêt (cercle noir vide), de trois directions (trois flèches) avec le robot qui avance et d'un point d'arrêt (cercle plein) dans lequel le robot doit s'arrêter.

A.



B.



C.

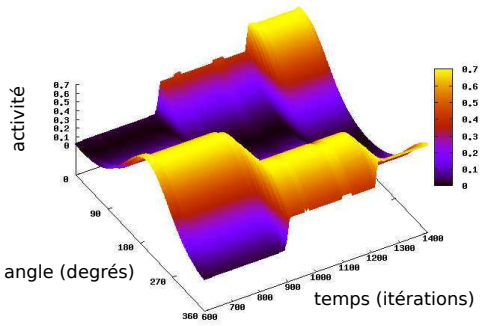


Fig. 5.25: A) Trajectoire (flèches noires) apprise par le robot. A chaque changement d'orientation, la stratégie de séquences temporelles apprend une nouvelle transition et la stratégie d'associations lieux-mouvement apprend un nouveau lieu où se trouve le robot et y associe le mouvement (orientation) courant. B) Carte des orientations prises par le robot durant l'apprentissage de la trajectoire. C) Activité du champ neuronal correspondant aux différentes orientations du robot.

Les deux derniers mouvements ne sont pas différenciable sur les orientations, car seule la vitesse linéaire a changée (le robot restant dans la même orientation). Par conséquent, dans les résultats que je présente dans cette étude, on pourra distinguer trois orientations et non quatre.
La figure 5.26 montre les différentes orientations associées à des lieux et restituées durant la phase d'apprentissage. On peut observer que les orientations prédites par les associations lieux-

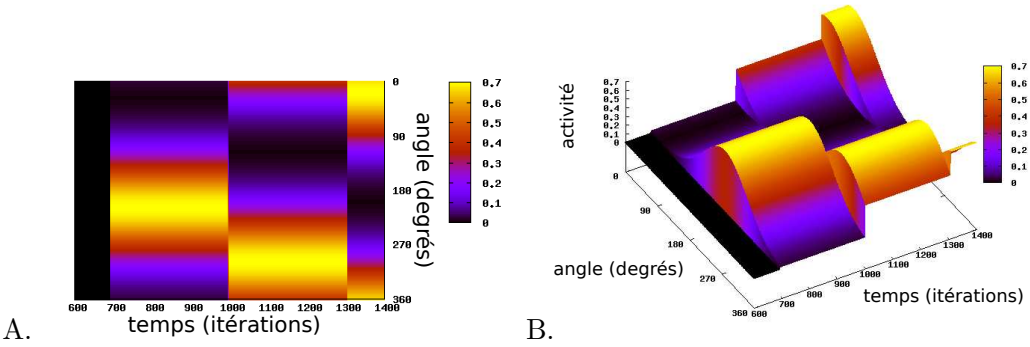


Fig. 5.26: Activités d'un champ de neurones ayant pour entrée les réponses délivrées par la stratégie d'associations lieux-mouvements. A) Carte des orientations délivrées par la stratégie d'associations lieux-mouvements durant l'apprentissage de la trajectoire. B) Activité du champ neuronal correspondant aux différentes orientations délivrées par la stratégie d'associations lieux-mouvements.

mouvements correspondent aux orientations du robot guidé par le professeur. Contrairement aux associations lieux-mouvements, la stratégies de séquences temporelles ne prédit pas de mouvement durant l'apprentissage, il n'y a donc aucune orientation prédite.
Pour tester la restitution de la trajectoire apprise, je kidnappe le robot du point d'arrivée au point de départ. Une fois à son point de départ dans le même état que lors de l'apprentissage, je laisse le robot libre de ses mouvements.

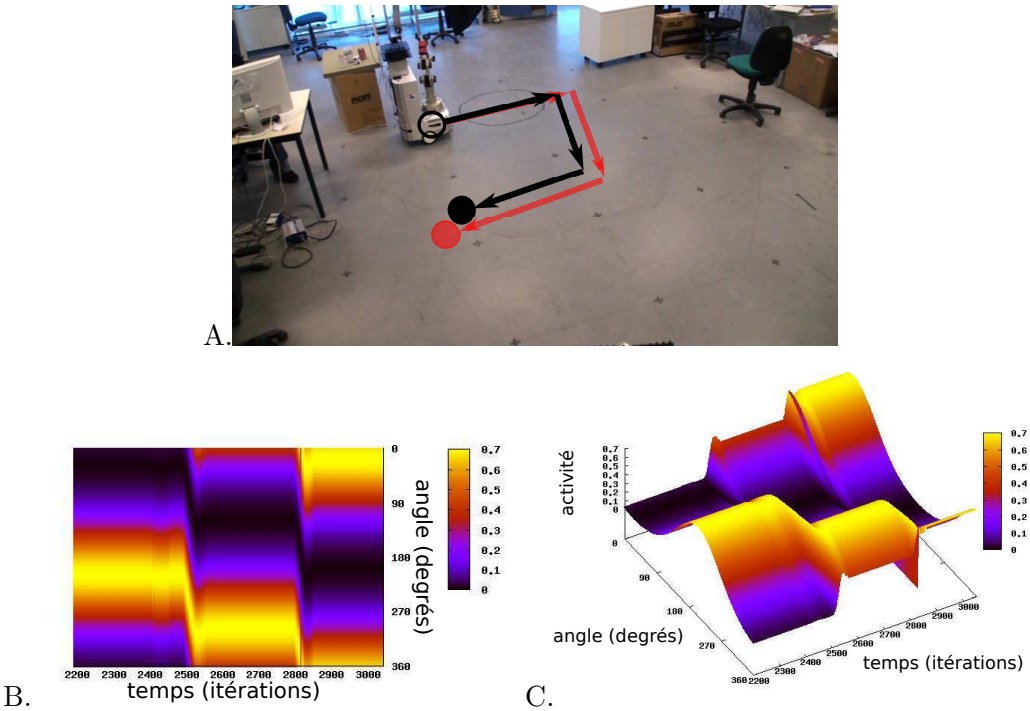


Fig. 5.27: A) Trajectoire restituée par le robot (flèches noires) superposée à la trajectoire apprise (flèches rouges) B) Carte des orientations (à partir de la boussole électronique jouant le rôle d'information proprioceptive du robot en terme d'orientations) prises par le robot durant la reproduction de la trajectoire. C) Activité du champ neuronal correspondant aux différentes orientations du robot.

En observant le comportement du robot, on remarque alors qu'il reproduit correctement la trajectoire telle qu'elle lui a été apprise, mais avec une certaine dérive. En effet, la trajectoire restituée par le robot est plus contractée que lors de l'apprentissage. Cette contraction est essentiellement due à la reconnaissance des lieux. En effet, le robot reconnaît un lieu lorsqu'il y entre, dès la frontière entre le précédent lieu et le suivant. Par conséquent, le déclenchement de l'action associée ne se fait pas au centre du lieu comme durant l'apprentissage de l'association, mais sur la frontière, donc plus tôt. Néanmoins, le robot reproduit correctement la forme de la trajectoire.

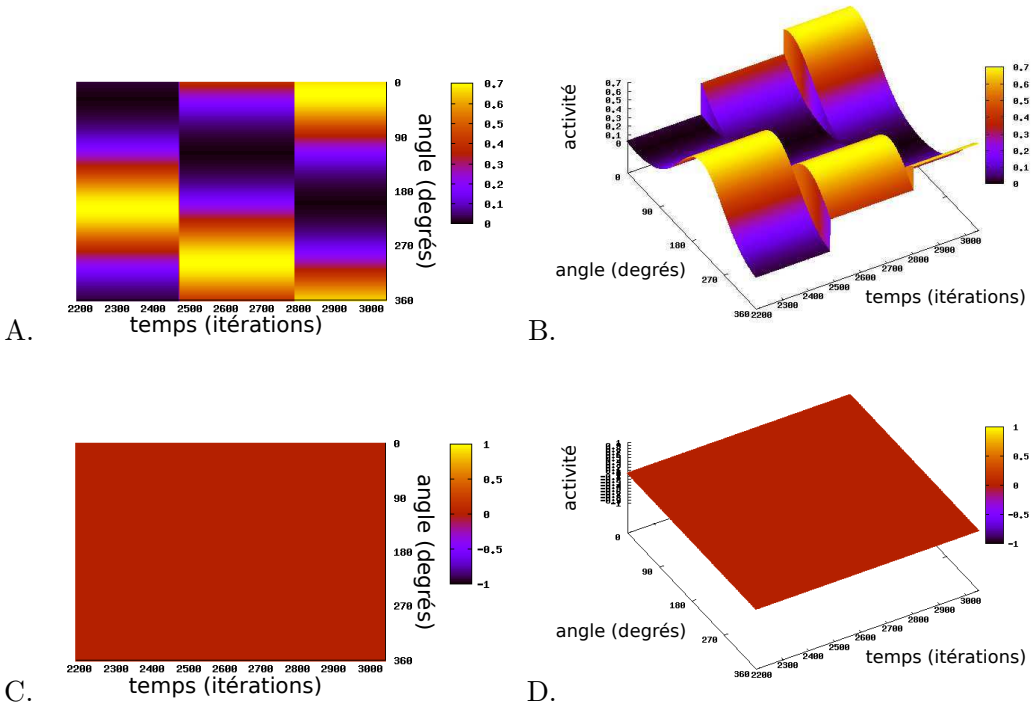


Fig. 5.28: Activités d'un champ de neurones ayant pour entrée les réponses délivrées par les stratégies d'associations lieux-mouvement (A. et B.) et de séquence temporelles (C. et D.) A) Carte des orientations délivrée par la stratégie d'associations lieux-mouvements durant la reproduction de la trajectoire. B) activité du champ neuronal correspondant aux différentes orientations délivrée par la stratégie d'associations lieux-mouvements. C) Carte des orientations délivrée par la stratégie de séquences temporelles durant la reproduction de la trajectoire. D) activité du champ neuronal correspondant aux différentes orientations délivrée par la stratégie de séquences temporelles.

La figure 5.28 montre les réponses (les orientations) de chacune des deux stratégies. On constate alors que seule la stratégie d'associations lieux-mouvements a fourni l'intégralité des commandes qui ont permis de restituer la trajectoire correctement.
Pour vérifier que la stratégie de séquences temporelles a bien appris la trajectoire et qu'elle permet de la restituer, je kidnappe une nouvelle fois le robot pour le remettre à son point de départ dans le même état que dans la phase d'apprentissage. J'inhibe alors les réponses de la stratégie d'associations lieux-mouvements sans pour autant en empêcher la reconnaissance de lieux. Par conséquent le robot n'a plus de commande à appliquer. Le professeur amorce le début du comportement de manière à fournir le premier élément de la séquence, puis laisse le robot libre de ses actions.
Grâce à la reconnaissance des différents lieux et des informations proprioceptives du robot (orien-

tation et vitesse linaire), les différents *chunks* correspondants s'activent permettant ainsi la re-synchronisation de la séquence. La figure 5.29 montre les orientations prises par le robot durant la reproduction de la séquence. On constate que le robot a bien réussi à reproduire la trajectoire avec une certaine dérive. En effet, la trajectoire reproduite est dilatée par rapport à celle qui a été apprise pour les raisons expliquées dans le chapitre 4 section 4.2.4.

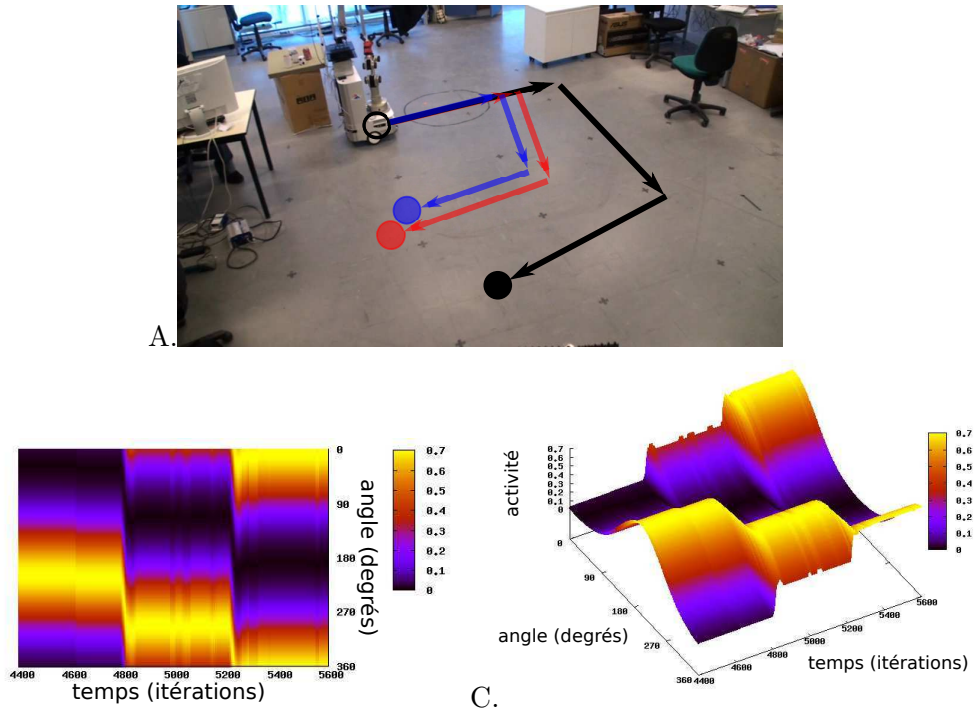


Fig. 5.29: A) Trajectoire restituée avec les réponses de la stratégie de séquences temporelles (flèches noires) superposée à la trajectoire restituée précédemment (flèches bleues) et à la trajectoire apprise (flèches rouges). B) Carte des orientations (à partir de la boussole électronique) prises par le robot durant la reproduction de la trajectoire avec la stratégie de séquences temporelles seule à fournir les commandes motrices. C) Activité du champ neuronal correspondant aux différentes orientations du robot.

La figure 5.30 montre les commandes (en orientation) prédites par la stratégie de séquences temporelles. On remarque alors que cette stratégie permet de restituer les bonnes commandes permettant de reproduire la trajectoire correctement. Mais alors, pourquoi cette stratégie n'a-t-elle pas répondue lors de la phase de reproduction précédente ?

5.11.3 Analyse du test de la compétition des deux stratégies de navigation

Pour chacune des phases de reproduction, la trajectoire n'était pas reproduite précisément comme elle avait été apprise, mais à chaque fois avec une certaine dérive. Lorsque la stratégie d'associations lieux-mouvements était seule à répondre, la trajectoire était alors contractée. Cette contraction est alors due à la généralisation des cellules de lieux dans l'espace. En effet, ces cellules codent pour une zone diffuse de l'environnement. Par conséquent, le robot reconnait un lieu dès qu'il dépasse la frontière de la zone correspondante et non quand il se trouve au centre où le lieu a été appris. Donc cette stratégie répond plus tôt. D'un autre coté, lorsque la stratégie d'apprentissage de séquence était la seule à répondre, la trajectoire était alors dilatée. Cette dilatation s'explique alors par deux raisons. La première est que durant la phase d'ap-

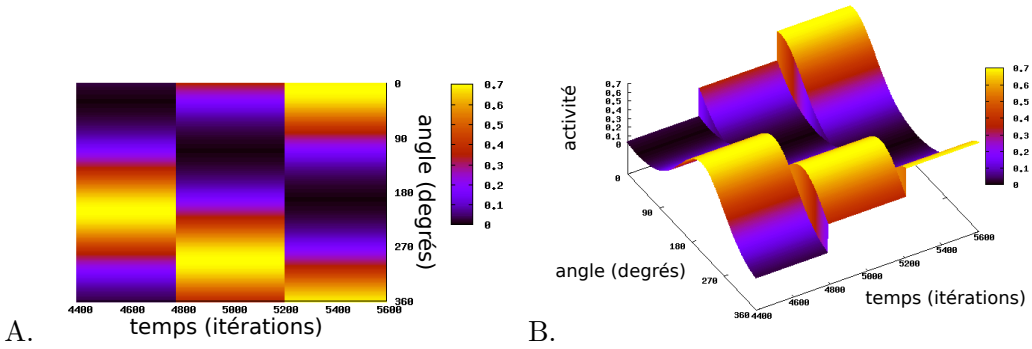


Fig. 5.30: Activités d'un champ de neurones ayant pour entrée les réponses délivrées par la stratégie de séquences temporelles. A) Carte des orientations délivrées par la stratégie de séquences temporelles durant la reproduction de la trajectoire. B) Activité du champ neuronal correspondant aux différentes orientations délivrées par la stratégie de séquences temporelles.

prentissage, lorsque le professeur corrige l'orientation du robot, le temps que le robot tourne jusqu'à détecter la nouvelle orientation est compris dans le timing de la transition apprise. La seconde raison pour origine le temps nécessaire à reconnaître des lieux. En effet, le robot ne peut se localiser dans un lieu qu'après avoir observé son panorama visuel. Dans mes travaux, ce temps est d'environ trois secondes. Comme décrit précédemment, la resynchronisation d'une séquence dépend du mécanisme de *chunking*. L'activité des *chunks* dépendent eux même du lieu dans lequel se trouve le robot. Par conséquent, durant cette période de temps, le robot continue son mouvement courant sans que la séquence soit resynchronisée. Ces deux raisons ont alors comme conséquence la dilatation de la trajectoire reproduite. Donc cette stratégie répond plus tard que le timing "observé" durant la phase d'apprentissage. Finalement, comme la stratégie d'associations lieux-mouvements répond plus tôt que celle de séquences temporelles, il est cohérent que le robot exécute essentiellement ces commandes. Ce changement précoce d'orientation est alors détecté en entrée de la stratégie de séquences temporelles qui par conséquent n'a pas eu le temps de déclencher de prédiction. Mais ici, le test réalisé ne permet pas de se rendre compte des propriétés de fusion/sélection du champ de neurones dynamiques.

5.11.4 Test de la fusion/sélection des réponses de différentes stratégies de navigation

Pour permettre de tester les propriétés de fusion et de sélection du champ de neurones dynamiques, il est alors nécessaire de forcer les situations de coopération et de compétition des deux stratégies de navigation.
Le robot est alors kidnappé pour être replacé au point de départ de la trajectoire. Le professeur inhibe alors les commandes fournies par la stratégie d'associations lieux-mouvements de manière à attendre la prédiction de la commande de la stratégie de séquences. Par cette manipulation, on force alors les deux stratégies à restituer la première commande (orientation) de la trajectoire. Par conséquent les deux commandes sont fusionnées dans un même attracteur dans le champ de neurones dynamiques (figure 5.31). Lorsque la stratégie d'associations lieux-mouvements reconnaît le second lieu, elle prédit la commande suivante. A partir de ce moment, les réponses de chacune des stratégies sont alors éloignées. Les attracteurs créés par les réponses des deux stratégies étant de grande taille, ils sont fusionnés en un seul attracteur sur le champ de neu-

rones dynamiques en moyennant les deux réponses. Par conséquent, le robot n'appliquera pas les commandes de l'une ou l'autre des stratégies, mais la moyenne des deux. Finalement, le robot part alors dans une nouvelle orientation qui n'a pas été apprise, la séquence temporelle est alors corrompue. La seule possibilité pour le robot de pouvoir récupérer la trajectoire sera alors de se retrouver dans un lieu connu et dans la même orientation que celle associée à ce même lieu pour réenclencher la stratégie temporelle.

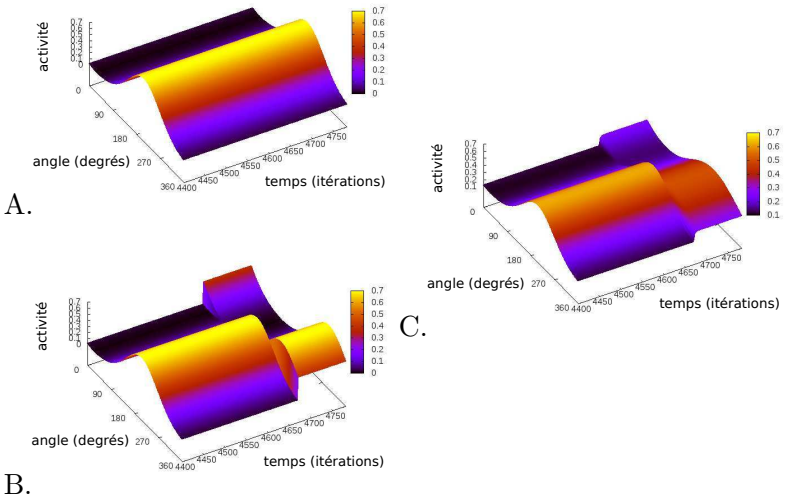


Fig. 5.31: A) Activités du champ neuronal correspondant aux réponses délivrées par la stratégie de séquences temporelles. B) Activités du champ neuronal correspondant aux réponses délivrées par la stratégie d'associations lieux-mouvements. C) Activités du champ neuronal dynamique dans lequel les réponses des deux stratégies sont fusionnées. On remarque alors que tant que les réponses des deux stratégies sont très proches (voir identique), les attracteurs de chacun sont fusionné en un seul. Lorsque la stratégie d'associations lieux-mouvements prédit la commande suivante, alors les réponses des chacune des stratégies sont distantes. L'attracteur du champ de neurones dynamiques est alors la moyenne des deux réponses.

Dans le précédent test, j'ai mis en évidence les propriétés de fusion et de moyennage des réponses fournies par les deux stratégies, mais pas leur sélection. En effet, ici même si les réponses sont distantes, les attracteurs sont suffisamment larges pour se fusionner. L'architecture travaille donc sur des attracteurs en basse résolution. Pour effectivement tester les propriétés de sélection du champ de neurones dynamiques, il faudrait que l'architecture travaille en haute résolution. Autrement dit, le noyau d'interaction (la différence de gaussiennes) doit alors permettre de créer des attracteurs de plus petite taille. Une conséquence de cette modification est que le robot peut être en dehors de la zone d'attraction et donc ne plus converger vers l'attracteur. Pour permettre au robot de rejoindre un attracteur même s'il n'est pas dans le champ d'attraction, il fera la sélection sur l'attracteur le plus fort et le plus proche.
Pour mettre en évidence la propriété de sélection du champ de neurones dynamiques, le robot apprend une trajectoire en "T". Pour permettre à chacune des stratégies d'apprendre deux trajectoires différentes, l'apprentissage de l'une est désactivée pendant que l'autre apprend. De cette manière, la stratégie d'associations lieux-mouvements apprend une première partie de la trajectoire (figure 5.32.A) et la stratégie de séquences temporelles apprend la seconde partie dont l'orientation finale est opposée à la précédente (figure 5.32.B). La figure 5.32.E montre les orientations prises par le robot durant l'apprentissage.

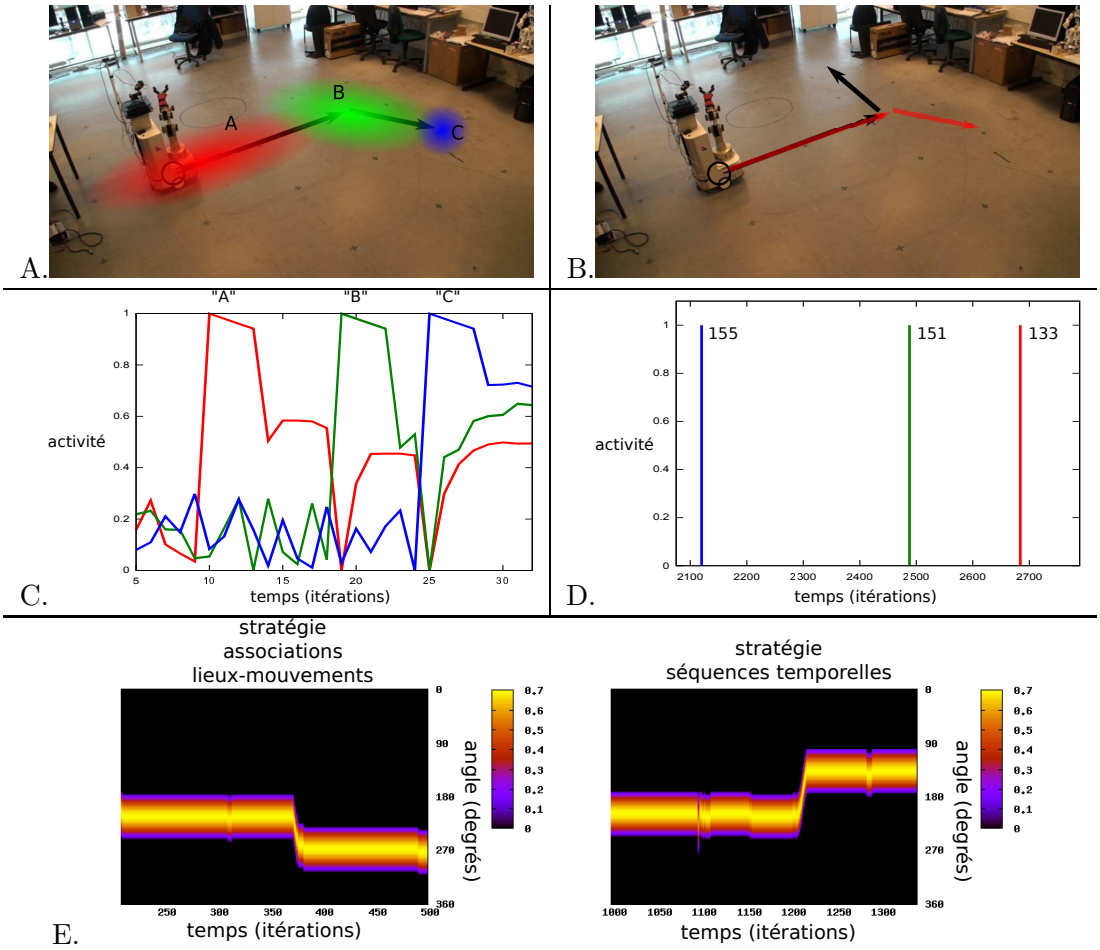


Fig. 5.32: Apprentissage d'une trajectoire en forme de "T". A) Le robot apprend une première partie du de la trajectoire avec la stratégie d'associations lieux-mouvements : "tout droit" puis "tourner à droite". B) Le robot apprend la seconde partie de la trajectoire avec la stratégie de séquences temporelles (flèche noires) : "tout droit" puis "tourner à gauche". Cette trajectoire est superposée avec la trajectoire apprise par la stratégie d'associations lieux-mouvements (flèches rouges). C) Activités des lieux appris durant l'apprentissage avec la stratégie d'associations lieux-mouvements. D. Etats cachés créés par la stratégie de séquences temporelles. E) Orientations (récupérée grâce à la boussole) sur un champ de neurones prises par le robot durant les deux phases d'apprentissage.

Lors de la reproduction de la trajectoire, le robot est kidnappé pour être remis au point de départ puis débute la reproduction de la trajectoire (figure 5.33.A). La première partie étant commune aux deux trajectoires, les deux stratégies proposent la même orientation. Les attracteurs sont donc fusionnés en un seul. Lors de la seconde partie de la trajectoire, on remarque que le robot a sélectionné celle de la stratégie d'associations lieux-mouvements, mais avec un certain retard. Regardons alors plus précisément les réponses de chacune des deux stratégies ainsi que les orientations effectivement prises par le robot grâce à la boussole électronique (figure 5.33.B). On observe que la stratégie de séquences temporelles répond en premier (figure 5.33.B ligne bleue), mais que le robot ne se dirige pas vers ce nouvel attracteur. En effet, la stratégie d'associations lieux-mouvements ayant toujours son attracteur à la même position angulaire, il est le plus proche de la position actuelle du robot, donc il y reste (figure 5.33.B entre la ligne bleue et la ligne verte). Ensuite, la seconde stratégie fournit sa réponse (figure 5.33.B ligne verte) pour le

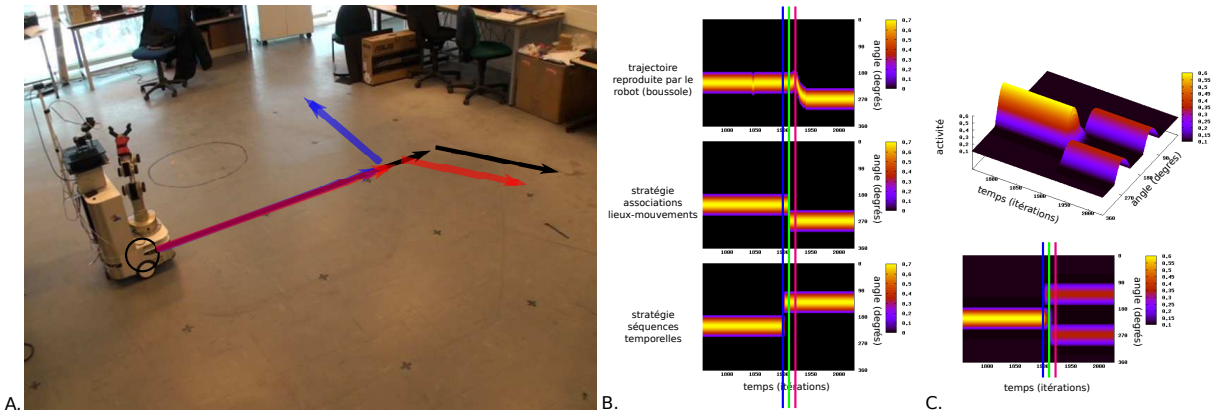


Fig. 5.33: A) Le robot reproduit la trajectoire (flèches noires). Cette trajectoire est superposés à celle apprise par la stratégie d'association lieux-mouvements (flèches rouges) et celle de séquences temporelles (flèches bleues). On observe alors que lorsque le robot arrive à l'endroit où il doit prendre la décision de soit tourner à droite soit tourner à gauche, le mouvement sélectionné est de tourner à droite. B) Orientations sur un champ de neurones prises par le robot durant la reproduction de la trajectoire (en haut). Ces mouvements réalisés par le robot sont comparés aux prédictions des stratégies d'associations lieux-mouvements (au centre) et de séquences temporelles (en bas). On remarque que la seconde stratégie prédit l'orientation suivante en premier (ligne bleue), mais que le robot reste sur son orientation actuelle. Lorsque la seconde stratégie fournie sa réponse (ligne verte), le robot se trouve durant une courte période avec un seul attracteur fort (celui de la première stratégie) ; il commence donc à se diriger vers cet attracteur. Puis lorsque le second attracteur à atteint une amplitude maximum (ligne rose), il est le plus proche de la position actuelle du robot. Par conséquent le robot sélectionne finalement cet attracteur comme orientation à atteindre. C) Activité du champ de neurones dynamiques durant la phase de reproduction avec les différentes phases de sélection décrites précédemment.

prochain état. Les propriétés dynamiques du champ de neurones ne permettent pas que le nouvel attracteur soit entièrement créé immédiatement, mais croît progressivement (figure 5.33.C entre la ligne verte et la ligne rose). Par conséquent, à ce moment précis, le robot va commencer à se diriger vers l'orientation prédite par la stratégie de séquences temporelles, car elle est à son activité maximale, même si elle est la plus éloignée. Une fois le second attracteur à son activité maximale (figure 5.33.C après la ligne rose), le robot prend finalement la décision de tourner à droite suivant la réponse de la stratégie d'associations lieu-mouvement. La règle définit a priori que le robot ira rejoindre l'attracteur le plus fort et le plus proche, permettant alors au robot de choisir l'attracteur créé par la stratégie d'associations lieux-mouvement. Finalement, le robot choisit de tourner à droite, mais avec le timing de la stratégie de séquences temporelles (figure 5.33.B ligne verte).

5.12 Discussion

Le premier test a permis de montrer comment deux stratégies sensori-motrices reposant sur des informations de sources différentes peuvent se compléter. Ce test repose sur une succession d'apprentissages et de restitutions des mouvements qui est déclenchée par le professeur. Mais ces résultats suggèrent aussi que notre architecture est capable de ré-apprendre un comportement sous un format différent (de le ré-encoder) sur la base de sa propre reproduction. Se posent alors des questions du point de vue développement. Il y a-t-il une stratégie qui se développe avant l'autre ? Se développent-elles en même temps ?
Le second test a permis de mettre en évidence les limitations de chacune des deux stratégies. La

stratégie d'associations lieux-mouvements répond d'une manière générale plus tôt que l'endroit où l'apprentissage a été réalisé. Comme expliqué précédemment, cette limitation est due à la généralisation dans l'espace des cellules de lieux. Quant à la stratégie d'apprentissage de séquences temporelles, elle répond généralement bien plus tard que le moment de l'apprentissage. Comme expliqué précédemment, ce retard est essentiellement dû au temps de resynchronisation de la séquence, mais aussi au fait que le temps durant lequel le robot tourne est compris dans le timing d'une transition lors de l'apprentissage. Pour que le robot reproduise plus précisément une trajectoire, il est alors nécessaire qu'il puisse corriger les apprentissages passés. Dans [Giovannangeli, 2007], l'auteur permet à un robot de rester sur une trajectoire avec une certaine précision. La trajectoire n'est plus une succession de lieux-mouvements (figure 5.34.A), mais un attracteur créé par un ensemble de lieux autour de la trajectoire dont les mouvements associés gardent le robot sur la trajectoire (figure 5.34.B). Ceci implique que la trajectoire soit apprise après plusieurs apprentissages.

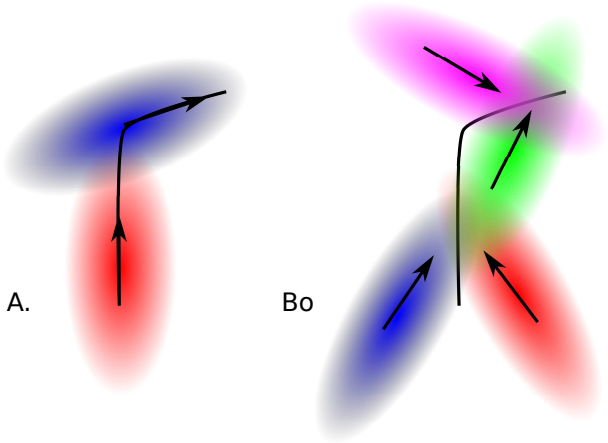


Fig. 5.34: Illustration de l'apprentissage d'associations lieux-mouvements. A) Cette illustration correspond au cas utilisé dans mes travaux. Le robot apprend un lieu sur la trajectoire à apprendre et y associe le mouvement à réaliser sur cette trajectoire. La trajectoire est donc encodée comme une succession de lieux-mouvements. B) Cette illustration consiste à encoder la trajectoire comme un attracteur construit à partir de cellules de lieux autour dont les mouvements associés permettent de faire converger le robot sur la trajectoire.

En ce qui concerne la stratégie de séquences-temporelles, des travaux sont en cours pour permettre l'adaptation du timing des transitions pour moyenner plusieurs itérations d'apprentissage. Néanmoins, une difficulté non négligeable est de conserver l'intégrité des séquences apprises. En effet, quand le robot est corrigé d'une démonstration sur l'autre, le réseau devra prendre en compte l'impact des corrections sur le timing de la séquence.
Le troisième test a permis de mettre en évidence les propriétés de fusion et de sélection du champ de neurones dynamiques. Ces propriétés dépendent de la taille du noyau d'interaction qui permet de créer des attracteurs plus ou moins grand. On a pu alors remarquer que lorsque les attracteurs sont très grands, il n'y avait pas de sélection possible, les attracteurs sont toujours fusionnés. Par conséquent le champ de neurones dynamiques ne fait que moyenner des attracteurs distants. Lorsque que les attracteurs sont de plus petite taille, le robot peut alors converger vers l'un ou l'autre des attracteurs. Pour permettre au robot de rejoindre un attracteur distant, une règle a été définie a priori permettant le choix de l'attracteur le plus proche. Pour retirer cette règle, une amélioration possible serait que le robot travaille en multi-échelles avec des attracteurs de différentes tailles. Les attracteurs seraient alors très grands (basse résolution)

au départ, puis rétréciraient (haute résolution). Une solution permettant un tel fonctionnement serait que les différences de gaussiennes (DoG) soient créées dynamiquement par l'architecture. Il est également imaginable que la résolution des attracteurs soient définie par un paramètre de vigilance. Dans mes travaux, les réponses de chacune des stratégies sont binaires, donc tous les attracteurs ont la même force d'attraction. Une autre amélioration possible serait que chacune des stratégies ait une saillance permettant alors la création d'attracteurs plus ou moins fort. La sélection de l'action pourrait se faire en fonction de la distance et de la force d'attraction. Les saillances des réponses des différentes stratégies pourraient être le résultat d'un apprentissage par renforcement. Ce renforcement pourrait alors provenir de structures de plus haut niveau tel que le cortex cérébral qui réaliserait alors un "priming" sur les structures de plus bas niveau. La saillance des réponses des stratégies spatiales et temporelles pourrait provenir du niveau d'activité de ces réponses. En effet, les réponses de la stratégie d'associations lieux-mouvements pourraient être pondérées par le niveau d'activité des cellules de lieux. L'activité des réponses de la stratégie de séquences temporelles pourraient être pondérées par la proximité au moment du déclenchement d'une prédiction.

Dans mes travaux, le modèle traite d'un coté les réponses d'une stratégie temporelle et de l'autre d'une stratégie spatiale. On peut alors se poser la question de la pertinence de fusionner/sélectionner des informations de nature différente. En effet, la stratégie d'association lieux-mouvements informe sur l'endroit où réaliser l'action alors que la stratégie de séquences temporelles informe sur quand réaliser l'action. Néanmoins, le modèle que je présente avec les deux boucles hippocampiques n'est pas complet. En effet, on remarque que pour la stratégie d'associations lieux-mouvement, il n'y a pas de transitions de lieux. Ces transitions coderaient alors des transitions de lieux permettant d'informer sur quand changer de lieu. Cette information serait alors de même nature que les séquences sensori-motrices permettant ainsi une fusion/sélection de réponses homogènes. Si ici ces transitions de lieux explicitent des aspects temporels sur les cellules de lieux, la même question se pose alors sur la stratégie de séquences temporelles : comment expliciter des aspects spatiaux de cette stratégie ? En effet, pour rester cohérent avec le modèle, l'information serait en amont des transitions, c'est à dire dans les états d'entrée et/ou dans les états cachés. A priori, cette information ne peut pas se trouver sur les états d'entrée, car à partir des informations proprioceptives seules, il paraît difficile de se localiser. En effet, le robot peut avoir une orientation de 30° aussi bien dans un couloir quand dans une salle. Par contre, les états cachés sont plus riches en informations. En étant associés à une dynamique interne dont l'état dépend de *chunks* qui répondent sur les lieux reconnus en plus des informations proprioceptives, ils peuvent permettre de fournir une information de nature spatiale dépendante des actions passées du robot. Alors, tout comme les lieux sont associés à des mouvements, les états cachés pourraient l'être également. Finalement, les réponses prédites par la reconnaissance de lieux et les réponses qui pourraient être prédites par la reconnaissance d'un contexte interne auraient une nature suffisamment homogène pour permettre leur fusion/sélection.

De la même manière, on peut également se poser la question de la pertinence de simuler l'hippocampe en deux tranches distinctes. En effet, ces deux tranches pourraient être fusionnées en une seule hippocampe permettant de traiter des informations multimodales plus riches. Cette structure ne coderait plus pour des informations précises comme des lieux ou des informations proprioceptives, mais pour une sorte de contexte sensori-moteur composé, entre autre, de lieux/informations proprioceptives représentant le contexte instantané dans lequel se trouve le robot par rapport à son environnement. Ces informations provenant des différentes modalités proviendraient alors des différents cortex (préfrontal, visuel, auditif, etc) et se fusionneraient tout ou partie dans le cortex ehtorinal en entrée de l'hippocampe (figure 5.35). Le cortex pré-

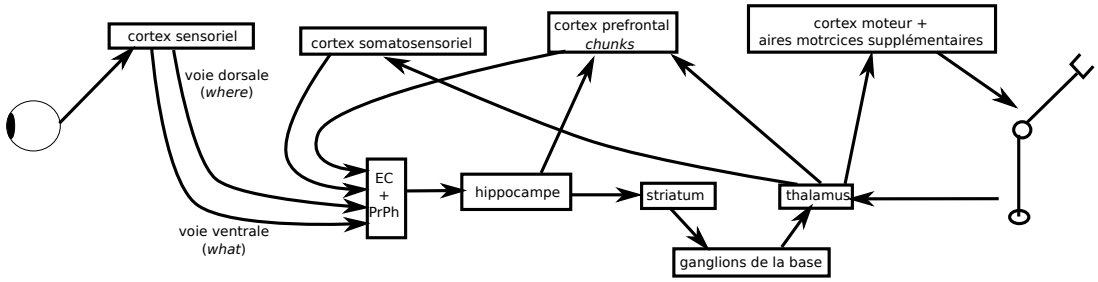


Fig. 5.35: Modèle de structures du cerveau pouvant être concernées par les différents mécanismes utilisés. Le cortex sensoriel traite les informations visuelles. Les informations visuelles sont séparées en deux voies *what* et *where*, puis fusionnées dans le cortex enthorinal (EC) permettant ainsi de coder des lieux. EC reçoit également les informations motrices du thalamus à travers le cortex somatosensoriel et les *chunks* du cortex préfrontal. EC transmet ses informations à l'hippocampe pouvant les mémoriser sous forme de transitions. Les prédictions de l'hippocampe sont envoyées aux ganglions de la base par l'intermédiaire du striatum. Puis une fois l'action finale sélectionnée par les ganglions de la base, elle est transmise au cortex moteur ainsi qu'aux aires motrices supplémentaires pour être appliquée sur les effecteurs.

frontal étant fortement interconnecté aux structures sous-corticales coderait alors l'équivalent de *chunks*. En effet, un *chunk* (sous forme de neurone ou de réseau de neurones) serait une sorte de contexte interne dans lequel l'action sélectionnée serait présente via les connexions provenant du thalamus. Ainsi par les connexions du cortex préfrontal vers les structures sous-corticales, les *chunks* seraient une modalité supplémentaire ; une sorte de senseur de l'état interne. Néanmoins, cela n'empêche pas de voir fusionner et/ou sélectionner les réponses de différentes structures.
L'execution en parallèle de deux stratégies de navigation nécessite l'utilisation d'outils adaptés. En effet, une telle architecure est composée de plusieurs centaines de milliers de neurones et elle est executée sur plusieurs unités de calcul en parallèle. De plus, la stratégie de navigation spatiale (associations lieux-mouvements) étant reprise dans le cadre de l'intégration de travaux, l'architecture doit être suffisament souple pour permette la suppression et l'ajout de nouvelles voies de catégorisation.

Chapitre 6

Réseaux de neurones temps réel distribués

Les réseaux de neurones que nous avons présenté dans cette thèse sont conçus comme des boucles perception-action (PerAc, [Gaussier *et al.*, 1998]). Chaque boucle est une combinaison d'une voie réflexe et d'une ou plusieurs voies de catégorisation. La décision finale est une commande envoyée aux moteurs du robot. Généralement, chaque voie correspond à une fonctionnalité propre (apprentissage temporel, apprentissage spatial, etc). Chaque voie possède sa "constante de temps" et sa propre vitesse d'apprentissage et d'exécution, les différentes voies doivent donc pouvoir fonctionner de manière asynchrone les unes par rapport aux autres. Néanmoins, différentes voies peuvent communiquer entre elles, par exemple le mécanisme de "chunking" vu précédemment, où l'information de la voie d'apprentissage spatial permet de resynchroniser les dynamiques de la voie d'apprentissage temporel. Par conséquent, les voies doivent pouvoir échanger des informations, généralement non bloquante. De plus, on souhaite pouvoir ajouter ou retirer facilement une voie entière (toute une chaîne de traitement) du réseau sans que la dynamique des autres voies ne soit altérée (sans parler d'un ajout ou retrait à chaud). On désire juste conserver une dynamique similaire indépendamment de la charge globale du processeur. Enfin, viennent s'ajouter les contraintes propres au contrôle d'un robot impliquant une utilisation du temps réel contrainte par le matériel embarqué, tout en gardant des commandes cohérentes malgré la répartition du traitement sur plusieurs unités de calculs.
Donc il est indispensable d'avoir un outil qui permette de répartir de très grands réseaux de neurones sur plusieurs unités de calculs. Paralléliser le traitement, soulève les questions du temps de traitement, ainsi que leur fusion/sélection lors de la décision finale. Comment simuler plusieurs structures en parallèle ?
Aujourd'hui, il existe plusieurs outils permettant de concevoir et simuler des réseaux de neurones artificiels. *SNNS* (Stuttgart Neural Network Simulator) [Zell *et al.*, 1993] sauvegarde de la description du réseau de neurones dans un fichier texte générée par une interface graphique. Le projet initialement écrit en langage C a évolué en langage Java sous le nom de *JavaNNS* depuis 2006. La différence avec les simulateurs cités précédemment est que *SNNS* permet de distribuer le calcul sur des machines de calcul distantes. Ceci est rendu possible par l'utilisation d'appel de procédures distantes, plus connu sous l'acronyme RPC (Remote Procedure Call).
L'un des outils les plus populaire est *Matlab* qui permet, à travers des boites à outils, de simuler des réseaux de neurones artificiels [Demuth et Beale, 2006]. *Matlab* est un outil mathématique performant qui permet entre autre de réaliser facilement les calculs matriciels. L'une des forces de ce logiciel est sa communauté active qui étend les fonctionnalités avec divers algorithmes de traitement d'images, d'apprentissages, etc. D'autres outils dédié à la conception et à la simulation de réseaux de neurones artificiels sont librement distribué comme *YANNS* (Yet Another Neural Network Simulator) [Boné *et al.*, 1998] et Aspirine/Migraine [Leighton, 1994] qui définissent leur propre langage de description de réseaux de neurones et fournissent le programme permettant de simuler le réseau décrit. D'une manière générale, ces outils imposent de se former à leur langage respectif pour permettre de les utiliser efficacement.
GENESIS [Beeman *et al.*, 2007] est un simulateur de réseaux de neurones artificiel créé pour simuler des neurones proche des neurones biologiques. Dans ce programme, le choix a été fait de simuler avec une granularité fine. Contrairement aux simulateurs vu précédemment qui assignent une fonction à un groupe de neurones, ici chaque neurone est divisé en plusieurs compartiments. La modélisation est réalisée par un langage propre au programme. Il est possible de distribuer un réseau de neurone sur plusieurs unités de calcul en utilisant *PGENESIS*. *PGENESIS* repose sur PVM (Parallel Virtual Machine) pour répartir le calcul sur plusieurs unités de calcul. Dans ce cas, le programme exécute plusieurs processus *GENESIS*, chacun simulant une partie du réseau de neurones. La communication entre chaque processus s'effectue par envoi de messages assuré

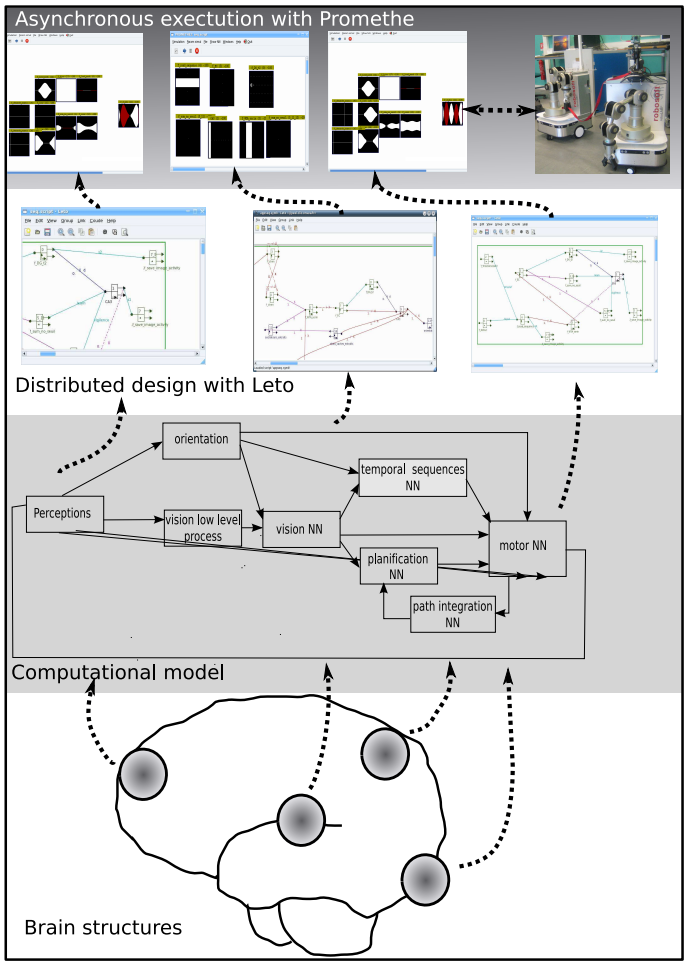


Fig. 6.1: Processus de développement d'un réseau de neurones artificiels pour le contrôle d'un robot. Le développement débute à partir de modèles de structures du cerveau (aires visuelles, boucle hippocampique, fonctions du cervelet, etc). Chaque structure est assimilée à un ensemble de fonctions et joue un rôle (apprentissage, filtrage, etc) dans le fonctionnement global du modèle. A partir de ce modèle, les fonctions (aussi appelé groupes de neurones) sont réparties en plusieurs réseaux de neurones artificiels conçus avec l'outil *Coeos*. Durant l'expérience sur le robot, chacun des réseaux de neurones artificiels est exécuté avec *Promethe* sur une unité de calcul. L'ensemble doit respecter des contraintes de temps de manière à assurer un contrôle correct du robot.

par PVM. Cette gestion de la répartition du calcul a pour conséquence d'alourdir le langage du logiciel et implique que l'utilisateur ait des connaissances particulières dans le domaine du calcul parallèle.

Ikaros [Balkenius *et al.*, 2009] est un simulateur dont l'objectif est de permettre la simulation de structures du cerveau. La modélisation se fait par modules. Un module est un morceau de code qui peut aussi bien définir un neurone, une région du cerveau ou toute autre chose. Chaque module possède des entrées et sorties lui permettant de communiquer avec d'autres modules. Les données transmises sont des tableaux de nombres flottants. Un premier point important de ce simulateur est de faciliter grâce aux modules, la répartition du calcul sur plusieurs unités de calcul. La communication entre les modules est gérée par une couche réseau sur des sockets standard de type *BSD*. Un autre point fort de ce programme est la possibilité de directement s'interfacer avec du matériel robotique. Ceci permet de directement tester un modèle sur un

robot dans l'environnement réel. *Ikaros* intègre également une gestion de contraintes temps réel reposant sur le standard *Posix* avec l'utilisation du multithreading via la libraire *pthread* (Posix thread). Cette gestion du temps réel permet d'assurer une certaine qualité sur le contrôle de robots. Par contre, *Ikaros* n'offre pas la possibilité de développer un modèle - des modules - via un outil particulier. La description du modèle est faite dans des fichiers au format *XML*.
Cette liste de simulateurs de réseaux de neurones est loin d'être exhaustive. Elle rend compte de la nécessité d'outils adaptés aux contraintes de la simulation (batch vs temps réel, modèles statistiques vs neurobiologiquement plausible) et des choix à faire a priori comme la granularité de la simulation. De plus, pour faire face à la consommation de calcul qui croît avec la complexité des simulations, on se rend compte qu'il est souvent nécessaire d'utiliser un grand nombre d'unités de calcul. Souvent, cette partie du simulateur n'est pas développée par les créateur du programme, mais repose sur des librairies existantes n'ayant pas de spécificités ni pour les réseaux de neurones, ni pour les flux de données qui y transitent.
Durant ses travaux, Philippe Gaussier à initié le développement d'un couple de programmes qui permettent la modélisation de réseaux de neurones avec *Leto* et leur simulation avec *Promethe* [Gaussier, 1992]. Les réseaux de neurones sont développés graphiquement via *Leto*. Il permet de créer des groupes de neurones et des liaisons entre ces groupes. Les groupes peuvent être configurés pour permettre de définir le nombre de neurones qu'ils contiennent ainsi que divers paramètres relatifs aux algorithmes qu'ils vont exécuter. Les liaisons permettent différentes connectivités entre les neurones de deux groupes (figure 6.2). Il y a quatre types de liaison : de un neurone vers tous ceux du groupe destinataire, de un neurone vers un voisinage, de un neurone vers un du groupe destinataire et aléatoirement de un neurone vers ceux du groupe destinataire.

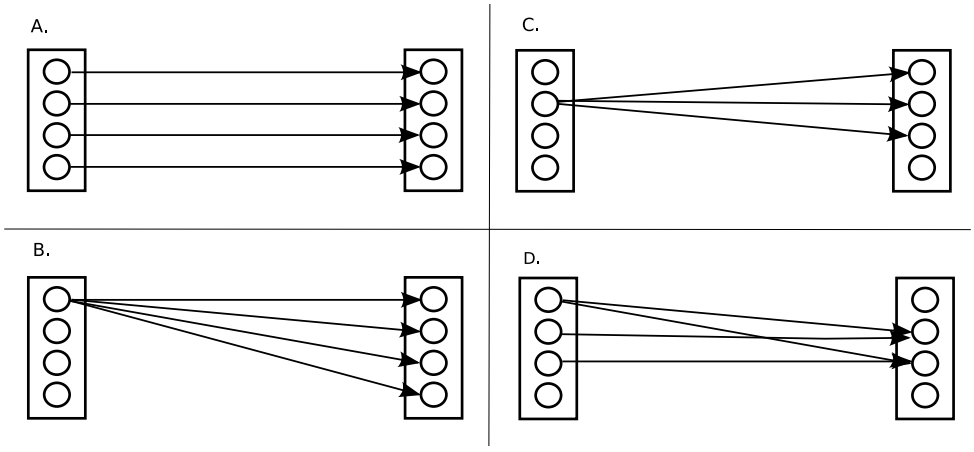


Fig. 6.2: Illustration des différents types de connexions entre deux groupes de neurones. A) Connexions de 1 vers 1. Les neurones du premiers groupes sont connectés a ceux du second groupe aux mêmes positions (mêmes indexes). B) Connexions de 1 vers tous. Chaque neurone du premier groupe est connecté à tous les neurones du second groupe. C) Connexions de 1 vers un voisinage Chaque neurone du premier groupe est connectés à celui du second groupe à la même position ainsi qu'à un certain nombre (paramétrable) de ses voisins. D) Connexions de 1 vers aléatoire. Chaque neurone du premier groupe est connecté aléatoirement à ceux du second groupe.

Un réseau de neurones compilé avec *Leto* est ensuite exécuté par *Promethe*. *Promethe* est le simulateur qui ordonnance et exécute les groupes de neurones (l'activité des neurones) et l'apprentissage (le poids des connexions entre les neurones des groupes). Avec la complexité croissante des architectures de contrôle, ces programmes ont évolué pour permettre la modélisation et la

simulation de plusieurs dizaines de milliers à quelques millions de neurones. Plusieurs *Promethe* peuvent être connectés grâce à des groupes particuliers de communication réseau. Une architecure est dans ce cas un macro réseau composé de réseaux de neurones communiquants les uns avec les autres.

6.1 Réseaux de neurones temps réel

Répartir une architecture implique une exécution sur des supports de calculs hétérogènes. Par exemple, un réseau de neurone de l'architecture peut être exécutée sur une machine mono processeur 32 bits et un autre sur une machine quadri voire octo-coeur 64 bits. Chaque processeur a le plus souvent des fréquences différentes et donc une vitesse d'exécution différente. Par conséquent, le temps de calcul d'une itération peut varier d'une machine à une autre. D'un point de vue purement optimisation des performances en temps de calcul, cela ne représente pas de réel problème. Par contre d'un point de vue contrôle et interaction avec un robot, le temps devient une contrainte critique aussi bien pour le matériel que pour la validation des algorithmes et des comportements. Si durant une phase d'interaction avec une tête expressive, le robot prend une minute de calcul entre chaque expression, très rapidement l'humain abandonnera l'interaction. De la même manière, si le système est beaucoup trop rapide, l'interaction est impossible. Dans le cadre de systèmes complexes interactifs, il est indispensable que les architectures respectent certaines contraintes de temps.
De plus, comme ces mêmes architectures sont massivement parallèles, la combinaison des contraintes temps réel avec la parallélisation soulèvent bien plus que des questions techniques et d'optimisations. Enfin, plus on parallélise une architecture, plus il y aura de temps consommé pour permettre aux différents réseaux de neurones d'un macro réseau de communiquer.
D'un point de vue purement informatique, on dégage deux types de temps réel : le temps réel "dur" et le temps réel "mou". Dans les deux cas, l'idée principale est d'imposer à un système de respecter des échéances prédéfinies par le développeur. Dans le cas du temps réel "dur", si la contrainte de temps n'a pas pu être respectée, alors le système est considéré en échec signifiant la fin de la mission ou des conséquences critiques pour la survie du système opérant. Dans le cas du temps réel "mou", le non respect du timing est moins critique. Si le système ne respecte pas la contrainte imposée, alors il peut continuer en considérant a priori que par la suite il la respectera. Ainsi, dans ce cas, on regarde plutôt sur une moyenne si la contrainte est respectée plutôt qu'à chaque instant.
Les outils de simulations étant executés sur le système d'exploitation *Linux*, les architectures de contrôle de robots interactifs développées sont soumises à des contraintes de temps réel "mou". Néanmoins, l'usage de contraintes de temps pour contrôler les moteurs d'un robot peut être critique à chaque instant, car il est indispensable de s'assurer qu'un robot mobile ne fonce pas dans un mur ou même sur des personnes ou même qu'un bras robotique n'aille par forcer contre un obstacle. Selon les besoins de l'architecture, il est donc plus approprié d'imposer des contraintes similaires au temps réel "dur". D'un autre coté, il n'est pas dramatique que durant une phase d'interaction le robot prenne de l'ordre de la seconde de retard. L'important ici est que le système soit suffisamment performant pour garder une interaction supportable avec un autre agent. Dans ce cas, il n'est pas indispensable de respecter un timing précis à chaque instant, mais plutôt sur une certaine période, en moyenne. Quelque soit le cas de figure, la gestion du temps réel est nécessaire pour permettre aux robots de remplir correctement leurs tâches, et mettre l'execution de nos modèles à l'abris de l'hétérogénéïté du matériel, en un mot de garder leurs performances reproductibles.

6.1.1 Ordonnancement des réseaux de neurones

L'ordonnancement consiste à organiser dans le temps la réalisation de tâches. La manière dont les processus vont être exécutés a un impact fort sur les services que vont pouvoir offrir les processus. L'ordonnancement permet de réaliser plusieurs tâches en parallèle sur un nombre limité de ressources : les processeurs, les mémoires, etc. Les processus doivent donc partager ces ressources. Comme il est impossible de pouvoir exécuter plusieurs processus en même temps, le système d'exploitation utilise un ordonnanceur permettant de partager le temps d'accès aux ressources.
L'algorithme d'ordonnancement certainement le plus connu est le *round-robin*. Cet algorithme partage de manière égale le temps pour chaque tâche sans aucune priorité. Les processus sont alors exécutés durant une courte période (un *quantum*) les uns après les autres. Il existe également des algorithmes d'ordonnancement plus complexes qui ajoute la notion de priorité. Chaque processus se voit attribuer une priorité qui lui permet un accès aux ressources plus important. Mais face à des besoins plus spécifiques, une notion importante est apparue dans les systèmes, le temps réel.

6.1.2 Les jetons

Dans le simulateur de réseaux de neurones artificiels *Promethe*, l'exécution est réalisée au niveau des groupes de neurones. L'algorithme d'ordonnancement utilise un mécanisme de jetons. Ces jetons circulent de groupe en groupe permettant ainsi leur exécution (figure 6.4). Il est important de noter que dans le cas d'une architecture répartie sur plusieurs unités de calcul, chacun des réseaux de neurones de l'architecture a son ordonnanceur. Il n'y a pas d'ordonnancement global. Au lancement du simulateur, des *threads* (processus légers) sont créés pour chaque groupe. Ces threads sont détruits seulement à la fin de la simulation. D'une manière générale, un groupe peut être exécuté si tous les groupes précédents avec lesquels il est connecté ont terminé leur exécution. Au début de la simulation, les groupes n'ayant pas de prédécesseurs sont exécutés. L'ordonnancement se déroule en plusieurs phases :

1. L'ordonnanceur recherche tous les groupes (threads) qui peuvent être exécutés. S'il n'en trouve pas, alors la simulation est bloquée.
2. Les groupes trouvés s'exécutent.
3. Une fois que les groupes ont terminés, alors l'ordonnanceur reprend la main et fait circuler les jetons sur les groupes successeurs en suivant les connexions.

Ces étapes sont répétées jusqu'à ce que tous les groupes du réseau de neurones aient été executés. Ce mécanisme de jetons qui se propagent de groupes en groupes et en parallèle selon l'architecture du réseau constitue une vague qui se propage du début à la fin du réseau de neurone. Cette vague est donc relancée cycliquement jusqu'à la fin de la simulation. Pour éviter les situations de blocages dues à des connexions en boucles (circuit dans lequel un groupe a en entrée un autre groupe qui utilise le résultat de sa sortie), on distingue deux types de connexions entre groupes : les connexions primaires et secondaires. La différence entre ces connexions est que celles qui sont secondaires ne sont pas prises en compte lors de la recherche des groupes qui peuvent être exécutés. En effet, sans cette solution, un groupe connecté à lui même ne pourrait jamais être exécuté, car il attendrait sans fin de s'être exécuté lui même.
Exécuter seulement les groupes dont les prédécesseurs ont terminé leur exécution permet d'obtenir un ordonnancement efficace. En effet, permettre à un groupe de s'exécuter alors qu'aucune nouvelle activité n'est disponible en entrée est peu pertinent pour l'architecture. Le mécanisme

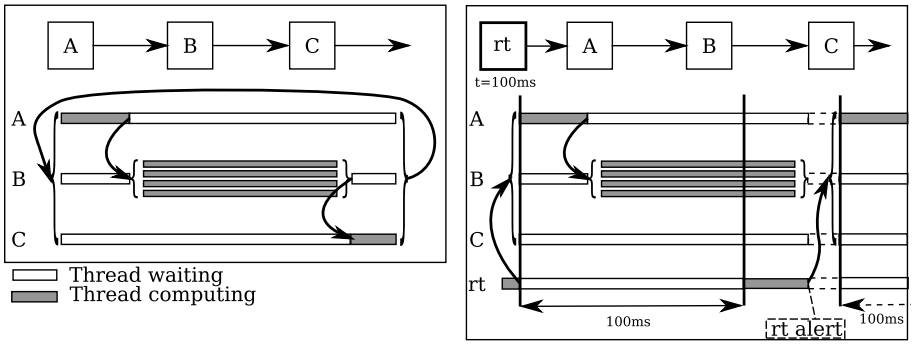


Fig. 6.3: A gauche : exemple d'ordonnancement des groupes de neurones sans contraintes de temps réel (sans jetons temps réel). Le groupe "B" attend la fin de l'exécution du groupe "A", puis débute son traitement. Lorsque le groupe "B" a terminé, le groupe "C" s'exécute à son tour. Une fois que le groupe "C" (en fin de chaîne de traitement) a terminé, une nouvelle vague de calcul est générée avec l'exécution du groupe "A". A droite : ordonnancement avec une contrainte de temps de 100ms : la contrainte n'est pas respectée durant l'exécution du groupe "B". Le groupe générant les jetons temps réel attend la fin de l'exécution du groupe "B", annule l'exécution du groupe "C" en détruisant ses jetons temps réel et génère de nouveaux jetons relançant ainsi une nouvelle vague de calcul (tâche A).

de jeton permet de tenir compte de ces dépendances intergroupes et d'offrir un ordonnancement efficace en terme de temps de calcul consommé.

6.1.3 Les jetons temps réel

Pour permettre aux architectures de respecter des contraintes de temps, l'ordonnanceur de *Promethe* a été modifié en ajoutant un type particulier de jetons, les jetons temps réel. Ces jetons sont générés par un groupe particulier appelé groupe temps réel. Comme les autres groupes, il s'exécute dans un thread créé au lancement du simulateur. Mais contrairement aux autres groupes, il a la capacité de se déclencher lui même après un certains temps. Ce temps correspond à la contrainte de temps qui doit être respectée. Le groupe "temps réel" génère des jetons temps réel avec un timing particulier et les propage dans le ou les groupes lui succédant avec lesquels il est connecté. Ensuite, c'est l'ordonnanceur qui va permettre la propagation des jetons dans les groupes suivant comme cela est fait pour les jetons normaux.
Lorsqu'un groupe temps réel s'exécute, il remet à zéro les jetons qu'il a précédemment générés. Si un lien avec l'option "-w" (pour "warning") a été spécifié, le groupe temps réel vérifie qu'il a bien reçu le jeton avec son propre identifiant (utilisé aussi comme niveau de priorité si plusieurs jetons temps réel circulent dans le réseau de neurones). S'il a reçu le jeton, alors c'est que tous les groupes ont terminé leur exécution. Dans ce cas, tout s'est bien déroulé dans le temps imparti. Si le groupe temps réel n'a pas reçu le jeton, alors il attend que les groupes terminent leurs executions, puis il détruit les jetons. Dans ce cas, la contrainte de temps n'a pas été respectée, puisqu'il reste des groupes en exécution. A l'image du temps réel "dur", ici l'exécution du reste du réseau est interrompue lorsque la contrainte de temps n'est pas respectée. Mais, de la même manière que le temps réel "mou", le simulateur permet également d'être plus souple sur le respect des contraintes de temps. En effet, il est possible de paramétrer un groupe temps réel pour qu'il ne détruise pas ses jetons même si le timing est dépassé (option "-c" pour "continue" sur le lien en entrée du groupe temps réel). Par contre, le groupe temps réel génèrera des jetons au timing prévu à l'origine. Il est alors possible que durant une certaine période il y ait deux vagues de calcul qui se déroulent en parallèle.

Comme évoqué précédemment, les jetons temps réel ne circulent que dans les groupes faisant parti d'une même "branche". Or, dans une architecture faisant cohabiter plusieurs stratégies en parallèle, il y a plusieurs branches (cf. les voies de catégorisation citées précédemment). Pour soumettre plusieurs branches à des contraintes de temps, il est possible de créer plusieurs groupes temps réel permettant ainsi à différentes branches de s'exécuter avec différentes fréquences. Avec un tel mode de fonctionnement, il est indispensable de faire un choix sur comment doit s'exécuter un groupe lorsque deux branches se fusionnent. Pour permettre de répondre à ce problème, *Promethe* offre une gestion de priorités sur les jetons temps réel. Ces priorités permettent de définir à quelle fréquence va être exécuté la suite du réseau lorsque deux branches de traitement se rejoignent et donc quelle va être la contrainte de temps à respecter.

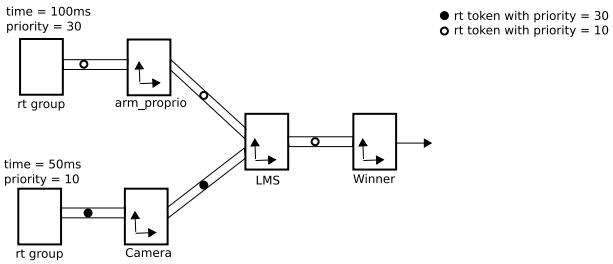


Fig. 6.4: Illustration montrant le fonctionnement de la gestion des jetons temps réel suivant leur priorité. Le groupe recevant les jetons de deux branches soumises à différentes contraintes de temps est soumis à la même contrainte que celle qui a la priorité la plus haute.

L'utilisation d'architectures réparties permet l'exécution en parallèle de plusieurs boucles sensori-motrices. Ce mode d'exécution permet de tester comment les réponses provenant de différentes boucles peuvent entrer en compétition ou se fusionner. En partant de l'hypothèse que chaque boucle s'exécute à des fréquences différentes, leurs réponses sont délivrées à des instants différents. Il est donc nécessaire d'avoir un mécanisme qui permet de fusionner et/ou sélectionner ces réponses asynchrones de manière à extraire une commande motrice cohérente et stable.

6.2 Réseaux de neurones distribués

En suivant l'idée que le même cerveau devrait être capable de s'adapter à différentes tâches, alors une même boucle sensori-motrice peut être partagée par différents réseaux de neurones. Par exemple, dans notre système, une boucle neuronale permet d'extraire des points d'intérêts d'une image prise avec une seule caméra vidéo. Cette boucle est aussi bien utilisée dans la reconnaissance d'expressions d'une tête de robot [Boucenna *et al.*, 2008] que dans la reconnaissance de lieux d'un robot mobile [Giovannangeli *et al.*, 2006].
La première version parallèle de *Promethe* utilisait PVM pour le calcul réparti sur plusieurs machines de calcul [Quoy *et al.*, 2000]. Cette machine virtuelle permet d'utiliser un ensemble d'unités de calcul sur lesquelles différents processus sont assignés. La communication entre processus est réalisée dans le code par des instructions de passage de messages permettant de définir quel type de message doit être envoyé ou reçu et par quel machine. De plus, cette première version parallèle de *Promethe* fournit des mécanismes de synchronisation grâce à la gestion de réceptions bloquantes ou non bloquantes de messages. Mais l'utilisation de PVM reste limitée à de petites architectures. En effet, quelque soit le nombre de processus, il y a qu'une file de message. En conséquence, cette file créé un goulot d'étranglement entrainant une chute des performances de

la simulation. Pour une execution temps réel, il fallait systèmatiquement parcourir la file d'attente, la vider pour ne récupérer que le dernier message valide. De plus, la déscription du macro réseau est spécifiée dans un fichier texte ayant pour extension "comm.prt" ("prt" pour "port", car le fichier contient les ports de connexion ou points d'entrées des différents réseaux de neurones). Ce fichier est écrit à la main par les développeurs. Il doit contenir deux sections : une qui permet de localiser chaque partie de l'architecture neuronale et une décrivant les liens réseaux (exemple 6.2).

```
# <partie de l'architecture> <adresse de la machine> <port de connexion>
begin network
nn1     localhost  1234
nn2     localhost  1235
nn3     localhost  1236
end

# <nom du lien> <source> <destination>
begin link
link1     nn1        nn3
link2     nn2        nn3
link3     nn3        nn1
end
```

Exemple 6.1: Exemple de fichier décrivant la répartition des différentes parties d'un réseau de neurones. La section "network" permet de localiser où se trouve les différentes parties de l'architecture. La section "link" permet de décrire les liens entre chacune de ces parties.

6.2.1 Coeos

L'augmentation régulière de la taille et le nombre de structures simulées a eu pour conséquence l'augmentation du nombre de réseaux de neurones à distribuer et le nombre de liens permettant de faire communiquer ces différents réseaux. Décrire la répartition à la main est donc devenu fastidieuse et source de nombreuses erreurs : liens réseaux manquant ou erronés induisant des erreurs difficiles à détecter. Dans le cadre d'expériences robotiques importantes, ces erreurs ont un impact sur le temps de développement très important. De la même manière que *Leto* permet de faciliter le développement de réseaux de neurones, *Coeos* permet de faciliter la répartition d'un macro réseau. D'une manière générale, *Coeos* permet de faciliter le passage à l'échelle (figure 6.5) en proposant une vue globale d'une architecture répartie (figure 6.6).

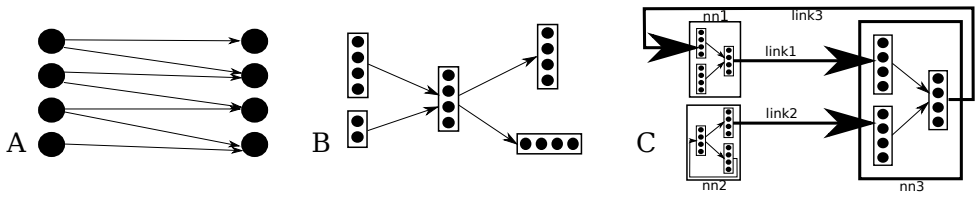
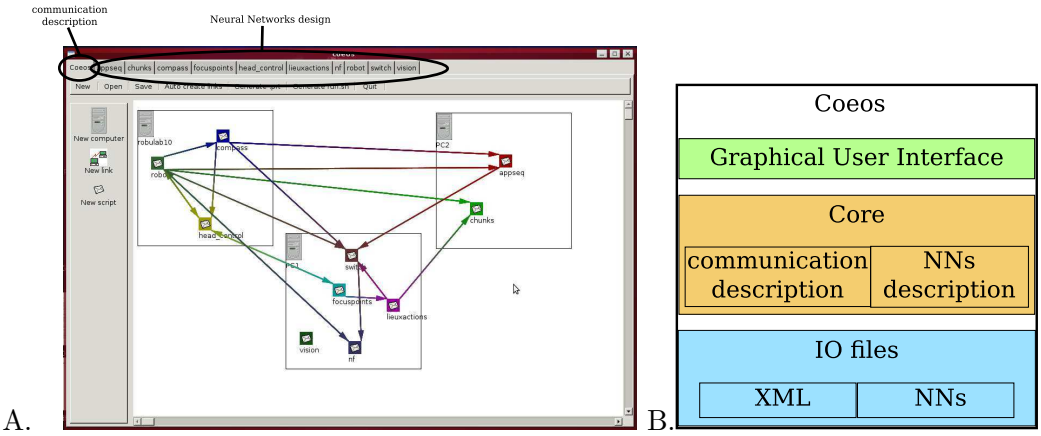


Fig. 6.5: Illustration de différents niveaux de modélisation de réseaux de neurones artificiels. A) Modélisation neurones par neurones. B) Modélisation par groupes de neurones (*Leto*). C) Modélisation de réseaux de neurones répartis (*Coeos*).

Pour spécifier la répartition d'un macro réseau, il suffit d'ajouter chaque réseau de neurones dans *Coeos*, de les assigner à une machine de calcul et de créer les liens entre ces parties. Les liens entre les groupes de deux réseaux de neurones différents sont appelés des liens réseaux. A partir de cette description graphique, il est alors possible de générer le fichier "comm.prt" qui sera donné au simulateur *Promethe*. *Coeos* permet de sauvegarder le macro réseau dans deux fichiers au format XML. Le premier fichier dont l'extension est ".net" permet de sauvegarder la liste des réseaux de neurones, à quelle machine de calcul ils sont assignés et les liens réseaux qui les lient. Le second fichier a pour extension ".cpt" et permet de sauvegarder la liste des machines de calcul sur lesquelles sont executés les réseaux de neurones.
Leto ne permettant le développement que d'un réseau de neurones mono-machine, il est rapidement devenu difficile de travailler avec une grande quantité de fenêtres. C'est pour quoi toutes les fonctionnalités de *Leto* ont été fusionnées dans *Coeos*. Grâce à un mécanisme d'onglets intégré dans *Coeos*, il est possible de rapidement naviguer dans les réseaux de neurones d'un macro réseau (figure 6.6).



A. B.

Fig. 6.6: A) Capture d'écran de l'interface de *Coeos*. L'outil permet l'affichage global d'une architecture neuronale distribuée sur le premier onglet. Chacun des onglets suivant permet l'affichage des différents réseaux de neurones. B) Architecture logicielle de *Coeos*. Le programme a été développé suivant différents niveau. Le niveau le plus bas permet de lire et écrire les fichiers décrivant les réseaux de neurones (format de fichier interne), ainsi que la description des connexions entre les réseaux de neurones (format *XML*). Le moteur (*Core*) de *Coeos* permet d'organiser les données lues en mémoire afin de les traiter (ajout, suppression, paramétrage, etc). Le niveau supérieur permet d'afficher graphiquement les données lues afin de permettre au développeur d'agir dessus.

Dans de grandes architectures ayant plusieurs dizaines de liens réseaux, il est alors rapide d'en oublier ou même de définir plusieurs liens avec le même identifiant ; ce qui peut poser de sérieux problèmes lors de l'exécution sans même le savoir. De manière à éviter ce genre de problème, *Coeos* propose une fonctionnalité qui permet de vérifier tous les réseaux de neurones d'un macro réseau qui ont été ajoutés. Durant cette vérification, le programme repère les groupes de communication et crée les liens réseaux correspondants. En cas de problème, *Coeos* ne crée pas les liens et renvoie un message d'erreur au développeur l'informant du problème rencontré (manque d'un groupe d'entrée ou de sortie ; groupe d'entrée ou de sortie en double pour un même lien réseau).
Finalement, avec l'utilisation de *Coeos*, il devient plus simple de gérer la répartition d'une architecture neuronale : il suffit d'ajouter les différentes réseaux de neurones qui composent le macro réseau, de générer automatiquement les liens et de sauvegarder. Il n'est alors plus indispensable d'être expert dans le développement d'applications réparties. Malgré tout, il est toujours néces-

saire que le développeur lance chaque réseau de neurones sur les machines de calculs spécifiées dans *Coeos*. L'intérêt de la solution actuelle est de pouvoir facilement arrêter ou relancer un réseau de neurones à chaud sans devoir redémarrer toute l'application (le robot pouvant par exemple continuer à rouler en évitant les obstacles en attendant que l'on modifie son réseau de neurones gérant la vision). Une fonctionnalité future de *Coeos* devrait alors permettre le déploiement automatique d'architectures distribuées.

6.2.2 Communications

Le simulateur *Promethe* intègre une couche de communication qui lui permet de communiquer à travers le réseau. Cette couche de communication réseau utilise des descripteurs réseaux (sockets) BSD avec les protocoles TCP/IP. Pour permettre la recopie de l'activité d'un groupe de neurones en entrée dans le réseau de neurones en sortie, un ensemble de groupes particuliers de communication réseau ont été ajoutés. Le passage des messages à travers le macro-réseau peut être synchrone ou asynchrone grâce à des liens bloquants ou non bloquants (reprenant les fonctionnalités disponibles sur le verrou PVM).
Le protocole doit non seulement permettre aux différents simulateurs de communiquer, mais également d'être un maximum robuste et d'informer l'utilisateur en cas de problème réseau tout en minimsant les perturbations induites par la parallélisation. La couche de communication doit être la moins consommatrice possible en ressources et en temps de calcul. Il est indispensable que les messages soient les plus simples et informatifs possibles.

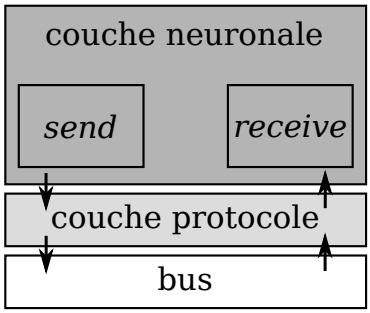


Fig. 6.7: Architecture de la gestion des communications réseau. Le réseau de neurone peut utiliser des groupes particulier afin d'envoyer ou recevoir des activités neuronales à travers le réseau à d'autres partie de l'architecture distribuée. Les données transmises par ces fonctions sont transmises suivant un protocole (défini en interne de la couche de communication) puis envoyées sur le bus de communication (TCP/IP).

Pour permettre une adaptation facile des communications réseau à d'autres protocoles (UDP, etc), la couche de communication a été abstraite. Elle comprend notamment trois niveaux. Le niveau le plus haut est la couche neuronale (fonctions visibles de l'utilisateur), puis il y a la couche protocole et la couche du bus de communication (figure 6.7).

6.2.2.1 La couche neuronale

Cette couche inclue principalement les groupes de communication nommés “f_send” et “f_recv”. Chacun de ces groupes peut être configuré grâce à différentes options spécifiées sur un lien en entrée du groupe. Ici, une simplification a été réalisée, car sur les versions précédentes il existait cinq groupes différents en réception et deux groupes différents en émission. Chacun de ces groupes était développé pour un fonctionnement particulier. Les nouveaux groupes de communication

permettent un meilleur paramétrage. Cette nouvelle manière de paramétrer les groupes permet aussi de facilement ajouter des options futures. Les options du groupe d'émission de message "f_send" sont :
- "-ack" : l'émission est bloquante. Le reste du réseau de neurone contenant ce groupe est bloqué tant qu'un message d'acquittement n'est pas reçu de l'émetteur. Il est possible de spécifier un temps d'attente limité en secondes avec l'option "-timeout"
- "-timeout=t" : cette option permet de spécifier combien de temps doit attendre le groupe avant de signaler une erreur de non réception de messages.

Par défaut, le groupe "f_send" est non bloquant, c'est à dire qu'il n'attend pas de confirmation de réception des messages. Le groupe en réception des messages réseaux offre plusieurs options :
- "-ack" : lorsque le groupe de réception a terminé de traiter le message, il envoie un message d'acquittement informant l'émetteur que tout s'est bien passé.
- "-block" : la réception est bloquante. Le réseau de neurones contenant ce groupe est bloqué tant qu'un message n'est pas reçu de l'émetteur. Il est possible de spécifier un temps d'attente limité en secondes avec l'option "-timeout"
- "-raz=x" : Au début de chaque exécution du groupe "f_recv", les neurones sont tous remis à zéro. Si "x" est précisé (entier naturel) alors la remise à zéro se fera après "x" exécutions du groupe. Cette option permet de limiter les effets de mémoire des activités précédentes en cas de non réception des messages pendant "x" itérations.
- "-next" : cette option permet de supprimer le dernier message reçu avant l'exécution du groupe et d'attendre le prochain. Ceci permet à l'application de traiter les activités neuronales les plus récentes. En effet, sans cette option, rien ne garantit que le message reçu ne date pas de plusieurs secondes ou plusieurs minutes, etc.
- "-timeout=t" : cette option permet de spécifier combien de temps doit attendre le groupe avant de signaler une erreur de non réception de messages.

Par défaut, un groupe de réception "f_recv" est non bloquant sans remise à zero et n'envoie pas de messages d'acquittement.

6.2.2.2 La couche protocole

La couche protocole permet lors de l'émission de messages d'encapsuler les données fournies par la couche neuronale et d'y ajouter des informations permettant de contrôler les communications (figure 6.8).

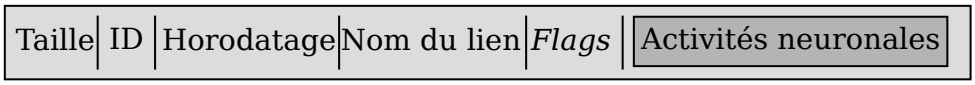


Fig. 6.8: Format des données transmises sur le réseau. En plus des données relatives aux activités neuronales, des informations sur la taille, l'identification du message, sur la date d'envoie du message, sur le nom de la connexion concernée et sur le comportement de la communication (à travers des *flags*) sont ajoutés afin de permettre de réaliser un contrôle d'intégrité de la communication (arrivée du message tardif ou non attendu) ainsi qu'un contrôle sur le temps dans la perspective d'utilisation de communications temps réel.

La couche protocole s'occupe des connexions avec les autres simulateurs. Pour permettre de spécifier quel est le groupe destinataire des activités de neurones artificiels transmises, une chaîne de caractères correspondant au nom du lien est ajoutée.
A l'initialisation des connexions, chaque simulateur tente de se connecter aux autres à partir des informations données dans le fichier "comm.prt". Que les connexions réussissent ou échouent,

chaque simulateur lance ses serveurs. En effet, comme il n'y a pas d'ordre de lancement des simulateurs, il est indispensable que chacun se mette en attente de connexions des autres. Par conséquent, chacun des simulateurs est en même temps client et serveur des autres simulateurs (fonctionnement *peer to peer*). Que ca soit en tentant de se connecter ou en réceptionnant une tentative de connexion d'un autre *Promethe*, la couche protocole réalise un échange de messages de connexion. Donc, en plus des échanges d'informations sur les activités, il y a des types de messages particuliers pour s'assurer du bon fonctionnement de la communication. Pour ceci, un champ "flags" est rempli à chaque échange entre deux simulateurs. Ce champ peut prendre plusieurs types qui permettent de définir le type de message dont il s'agit. Il peut s'agir d'une demande de connexion (MSG_CONNECT), d'une confirmation de connexion (MSG_CONNECT_OK), d'un refus de connexion (MSG_CONNECT_REFUSED), d'un message simple (MSG) ou d'un message avec accusé de réception (MSG_ACK).
Cette couche ajoute un identifiant unique pour chaque message envoyé. Cet identifiant permet au récepteur de vérifier qu'il attendait effectivement cet identifiant, et d'avertir l'utilisateur en cas de problème. Pour permettre de répondre à des besoins tenant compte de contraintes de temps, on retrouve parmis les informations de cette couche un "timestamp". Cette information est le temps machine en secondes et microsecondes au moment ou le message est construit et émis. Ceci a pour but de prendre en compte le temps de circulation des flux d'activités entre les différentes réseaux de neurones. En effet, il peut être inapproprié qu'un réseau de neurone traite des messages trop vieux qui n'ont plus de sens par rapport au contrôle du robot dans un environnement changeant.

6.3 Cas pratique : un robot mobile qui classe et range des objets selon leur taille

Le développement de *Coeos* a permis de répondre à un nouveau besoin qui est celui des architectures neuronales de plus en plus importantes. Mais si la répartition permet de gagner en performances, elle perme également de simplifier l'intégration de divers travaux. Dans la perspective de permettre toujours plus de comportements et de comportements plus complexes, dans le cadre du projet européen *Feelix Growing*, une partie de mes travaux ont été intégré, en collaboration avec Florient d'Halluin, avec les travaux du laboratoire LASA (Learning Algorithms and Systems) de l'EPFL (Ecole Polytechnique Fédérale de Lausanne) portant sur l'apprentissage de gestes avec un bras robotique, sous la direction de Aude Billard.
Dans ce contexte, nous voulons que le robot soit capable de naviguer d'un lieu "A" vers un lieu "B" ou "C". Le choix du lieu de destination est alors réalisé en fonction de la taille de l'objet pris par le robot. En chaque lieu, le robot doit également faire un geste particulier. Ce scénario implique alors que le robot (figure 6.9) soit capable de naviguer, d'attraper un objet et de faire des gestes.

6.3.1 La navigation

L'architecture (figure 6.10) qui a fait l'objet de cette intégration est composée de la gestion des entrées/sorties avec le matériel robotique ("robot"), du contrôle des mouvements de la caméra et de la caméra ("head_control", de la gestions de la boussole électronique ("compas"), du traitement bas niveau de la vision ("focuspoints"), de la stratégie d'associations lieux-mouvements ("lieuxactions"), du champs de neurones dynamique ("nf") ainsi que du contrôle sur les mouvements du robot ("pf").

Fig. 6.9: Dispositif expérimental. Le robot est composé d'une plateforme mobile, d'un bras robotique, d'une caméra montée sur deux moteurs en configuration *Pan-Tilt* et d'une boussole électronique. Le robot embarque un ordinateur ainsi qu'un routeur sans fil WiFi pour communiquer avec les trois machines de calcul distantes.

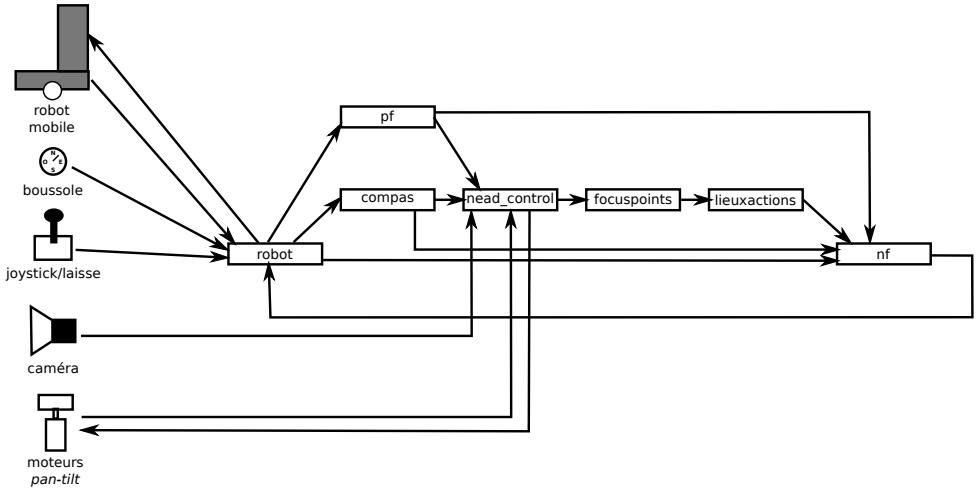


Fig. 6.10: Architecture telle qu'elle a été découpée. Le réseau de neurones "robot" lit les valeurs de la boussole électronique, du joystick et les informations odométriques du robot mobile. Ce réseau à également le rôle d'envoyer les commandes motrices au robot mobile. Le réseau "compas" à pour fonction de traiter les valeurs de la boussole électronique et de renvoyer ces informations à différents autres réseaux de neurones. "pf" pour "préfrontal" reçoit essentiellement les entrées du joystick et renvoie des signaux permettant le déclenchement de l'apprentissage et le contrôle des boucles sensori-motrices. Les signaux permettant de déclencher les apprentissages sont transmis à "head_control" qui récupère les informations proprioceptives des moteurs *Pan-Tilt* ainsi que les images de la caméra. Elle reçoit également les valeurs de la boussole à travers la partie "compas". "focuspoints" reçoit les informations proprioceptives (*Pan-Tilt* et boussole), le signal d'apprentissage et les images de la caméra. Ce réseau de neurones réalise un traitement bas niveau consistant à extraire les informations *what* et *where* et les fusionner dans un tenseur. Ce dernier est transmis au réseau "lieuxactions" avec les informations de la boussole et odométriques du robot pour apprendre un nouveau lieu et l'associer au mouvement courant. Lorsque ce réseau répond un mouvement a réaliser, celui-ci est envoyé à "nf" qui après fusion/sélection, envoie la commande motrice finale à "robot" pour être appliquée sur les moteurs. "nf" reçoit des signaux inhibiteurs permettant d'inhiber certaines boucles sensori-motrices.

Le réseau de neurones "robot" lit les valeurs de la boussole électronique et du joystick du robot mobile. Ce réseau à également le rôle d'envoyer les commandes motrices au robot mobile. Le réseau "compas" à pour fonction de traiter les valeurs de la boussole électronique et de renvoyer ces informations à différents autres réseaux de neurones. "pf" reçoit essentiellement les entrées du

joystick et renvoies des signaux permettant le déclenchement de l'apprentissage et le contrôle des boucles sensori-motrices. Les signaux permettant de déclencher les apprentissages sont transmis à "head_control" qui récupère les informations proprioceptives des moteurs *Pan-Tilt* ainsi que les images de la caméra. Elle reçoit également les valeurs de la boussole à travers la partie "compas". "focuspoints" reçoit les informations proprioceptives (*Pan-Tilt* et boussole), le signal d'apprentissage et les images de la caméra. Ce réseau de neurones réalise un traitement bas niveau consistant à extraire les informations *what* et *where* et les fusionner dans un tenseur. Ce dernier est transmis au réseau "lieuxactions" avec les informations de la boussole du robot pour apprendre un nouveau lieu et l'associer au mouvement courant. Lorsque ce réseau répond un mouvement à réaliser, celui-ci est envoyé au réseau "nf" qui après fusion/sélection, envoie la commande motrice finale au réseau "robot" pour être appliquée sur les moteurs. "nf" reçoit de "pf" des signaux inhibiteurs permettant d'inhiber certaines boucles sensori-motrices (associations lieux-actions, joystick).

6.3.2 L'objet

Une première phase du travail d'intégration était de permettre l'intégration du contrôle d'un bras robotique Katana[1] pour la préhension d'objets. Ce travail à également été développé sous la forme d'architecture neuronale avec *Coeos*. Si techniquement cela facilite l'intégration, il y a une cohérence globale à garder. La cohérence globale de l'architecture intégrée repose sur le modèle des boucles sensori-motrices *PerAc*.
L'objectif de l'intégration de ces travaux est de permettre à un robot d'apprendre et restituer des comportements mêlant la navigation, le geste et l'objet. Cette intégration pose la question de l'enchainement de ces différents comportements. Quand déclencher la prise ou le dépot d'un objet ? Pour simplifier le problème, il a été décidé que le robot détectera un objet à partir de son diamètre mesuré par proprioception lors de la prise. Par défaut le robot a sa pince grande ouverte. des capteurs infrarouges permettent de détecter la présence d'objet dans la pince, puis de déclencher sa fermeture. Le robot peut détecter deux types d'objets : ceux de grande taille (pince moyennement ouverte) et ceux de petite taille (pince presque fermée).
Dans l'architecture, le comportement de la pince est pré câblé : si les capteurs de la pince ne détectent rien, alors la pince est grande ouverte. Lorsque les capteurs détectent la présence d'un objet, alors la pince se ferme jusqu'à ce que les capteurs de pression soient actifs. La prise et le dépot des objets se fait dans des zones particulière de l'environnement. Pour matérialiser cette nouvelle possibilité dans l'architecture, les associations lieux-mouvements permettant la navigation se voient enrichies par l'ouverture de la pince en lieu-pince-mouvement. De cette manière, dans une zone de l'environnement, plusieurs associations peuvent être apprises, mais seule l'ouverture de la pince peut permettre la différenciation entre ces associations. Ce mécanisme a permis de réaliser un test intermédiaire afin de vérifier le comportement de navigation du robot avec des informations supplémentaire destiné à un autre type de comportement.

6.3.3 Le geste

Répartir une architecture ne signifie pas que ce sont des modules indépendants qui sont exécutés. En effet, même si la répartition d'une architecture permet une grande modularité, la répartition se fait principalement sur la parallélisation des boucles sensori-motrices. Dans le cadre du projet européen Feelix Growing, mes travaux et ceux du LASA de l'EPFL ont été intégré. Le défi ici

[1] bras robotique Neuronics

est d'intégrer des travaux qui sont par nature complètement différents. En effet, d'un coté mon architecture repose sur des réseaux de neurones développés en langage C, de l'autre les travaux reposent sur un programme séquentiel développé en C++. Ces travaux gèrent leurs propres boucles de calcul pour apprendre plusieurs démonstrations d'un geste. De plus, pour reproduire le geste, le programme pré-calcul les mouvements à réaliser directement avec les librairies du constructeur du bras robotique. Dans mes travaux, les boucles de calculs sont la succession des vagues de calculs du réseau de neurones et à chaque vague, de nouvelles commandes sont envoyées aux moteurs du bras robotique. Alors comment intégrer ces travaux sans en dénaturer l'intérêt scientifique ?
Les travaux du LASA consistent à permettre à un bras robotique d'apprendre par démonstration (manipulation passive du bras) un geste désiré. Dans un premier temps, l'objectif était de permettre la communication des deux programmes. Comme je l'ai expliqué précédemment, le simulateur de réseaux de neurones *promethe* embarque une librairie de communication réseau (*libcomm*). Cette librairie a alors été utilisée pour réaliser cette communication. Le programme du LASA n'ayant pas été fait pour communiquer en réseau avec d'autres processus, il a fallu développer une interface logicielle permettant la communication réseau. Cette interface à pour objectif d'être intégré avec le programme du LASA afin de permettre la communication entre les travaux de chaque partie. Par conséquent, cette interface a été développée en langage C++ intégrant le protocole de communication de la *libcomm*. Une fois ce travail terminé, il a été alors possible de déclencher les apprentissages de différentes démonstrations et la reproduction d'un geste à partir d'activités de neurones.
Dans un second temps, l'objectif était d'intégrer ces travaux avec mon architecture neuronale. Le choix qui a été fait est que le programme du LASA ne soit pas un simple module, mais considéré comme l'équivalent d'un réseau de neurone géré par *Promethe*. Pour réaliser cette intégration, le programme du LASA ne communique plus directement avec le bras robotique, mais seulement avec le reste de l'architecture neuronale qui se charge d'envoyer les ordres aux moteurs du bras. Cette intégration a impliqué une modification profonde du programme du LASA qui, d'une part, ne gère plus ces propres boucles de calcul et d'autre part, ne pré-calcule plus les mouvements du bras ; le programme devait fonctionner de manière itérative. Les boucles ont donc été retirées pour se conformer aux vagues de calcul du simulateur. Le programme à également été modifié pour permettre un calcul de commandes en vitesse à chaque vague à partir des informations motrices du bras robotique reçues du réseau de neurones.
Une autre partie du travail a été de permettre l'apprentissage et la reproduction de plusieurs gestes et non d'un seul. Pour ce faire, l'approche la plus simple a été adoptée : instancier plusieurs fois la même classe (dans le sens objet C++) où chaque instance permet d'apprendre plusieurs apprentissage par démonstration d'un même geste et pouvoir le reproduire plusieurs fois. Pour identifier chacune des instances, un neurone particulier était associé à une instance. Par conséquent, pour permettre l'apprentissage de trois geste, un groupe de trois neurones été ajouté.

6.3.4 Test de la navigation en fonction de l'objet

Dans le test réalisé, le robot apprend différentes associations lieu-pince-mouvement afin d'apporter dans une zone particulière un objet particulier. A partir de la taille de l'objet, le robot part soit dans la zone de gauche soit dans celle de droite (figure 6.12). Ici, le professeur simule l'ouverture de la pince en fournissant directement l'information grâce à trois boutons de joystick (pince complètement ouverte : pas d'objet, pince moyennement ouverte : objet de grosse taille, pince peu ouverte : objet de petite taille.). A chaque changement de direction (corrigée par le

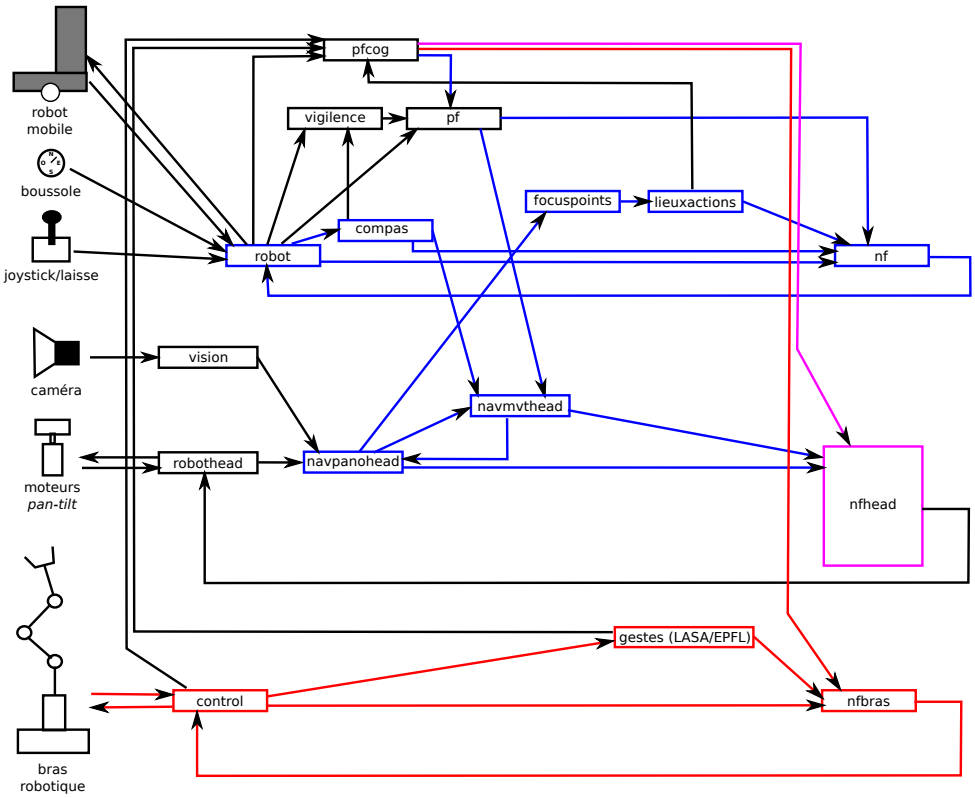


Fig. 6.11: Architecture de contrôle qui intègre les travaux de navigation et ceux du contrôle du bras pour la réalisation de gestes (LASA). Les travaux du LASA sont alors intégrés comme une partie de l'architecture neuronale dans une boucle sensori-motrice.

professeur) ou à l'arrivée devant un carton (détecté grâce aux ultrasons frontaux du robot), le robot apprend une nouvelle association lieu-pince-mouvement.

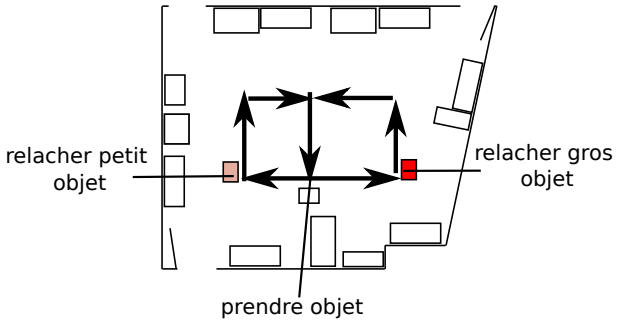


Fig. 6.12: Le robot apprend à se rendre vers le carton du centre. Lorsqu'il y arrive, le professeur lui donne un gros objet. Le professeur apprend au robot la direction dans laquelle il doit partir (vers le carton de droite) pour aller déposer l'objet. Une fois arrivé devant le carton de droite, le professeur retire l'objet de la pince du robot. Ensuite, le professeur apprend au robot à retourner à son point de départ et réalise un apprentissage similaire avec un petit objet à aller déposer sur le carton de gauche.

Dans l'expérience (figure 6.13), le robot évolue dans une zone d'environ 5 mètre de longeur par 3 mètre de largeur. Le robot apprend à venir dans la zone du carton du centre, il y apprend une

nouvelle association avec la pince grande ouverte. Le professeur simule la prise d'un gros objet et dirige le robot vers l'endroit où il doit apporter l'objet. Le robot apprend alors une nouvelle association avec la pince moyennement ouverte. Le robot arrive dans la zone du carton de gauche et il y apprend une nouvelle association avec pince moyennement ouverte. Le professeur simule alors le dépot de l'objet puis dirige le robot sur la suite de la trajectoire ; le robot apprend alors une nouvelle association et ainsi de suite jusqu'à la fin de l'apprentissage.

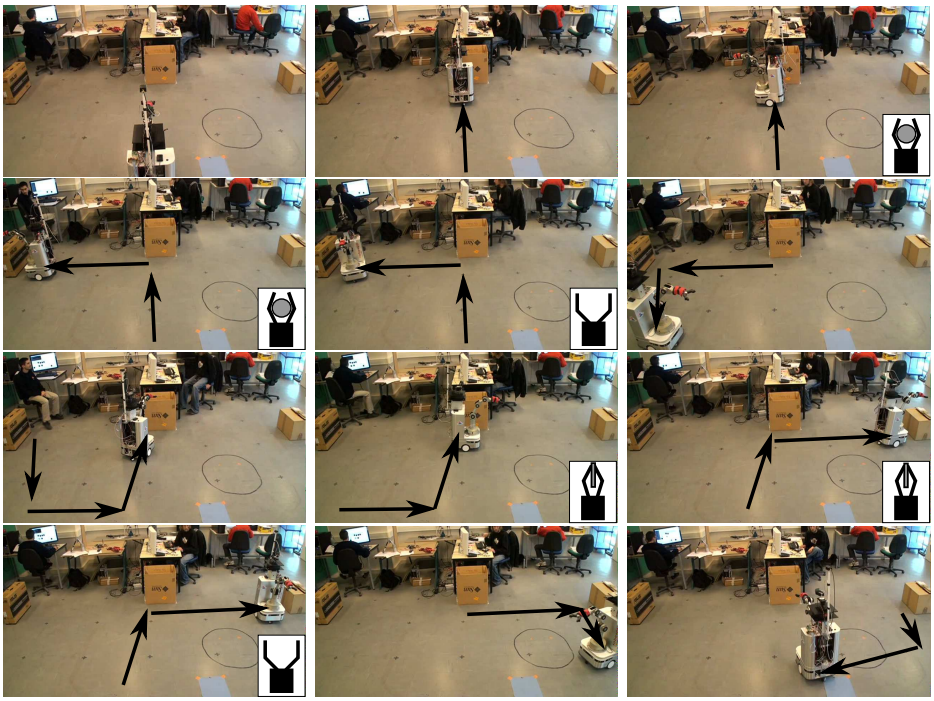


Fig. 6.13: Le robot apprend à attraper un objet sur le carton du centre et le relâche sur le carton de droite ou de gauche en fonction de la taille d'un objet. Le robot détecte la taille de l'objet à partir des informations proprioceptives de la pince (plus l'objet est gros, plus la pince est ouverte). Dans ce test, l'ouverture de la pince est simulée et fournie par le professeur. Lorsque le robot n'a pas d'objet dans sa pince, alors la pince est ouverte à son maximum. Une fois l'objet relâché, le robot apprend à revenir à son point de départ. A chaque changement de direction (corrigée par le professeur) ou à l'arrivée devant un carton (détecté grâce aux ultrasons frontaux du robot), le robot apprend une nouvelle association lieu-pince-mouvement. Devant le carton du centre, le robot apprend alors trois associations (quand il arrive devant pince ouverte, quand il a attrapé un objet de petite taille et quand il a attrapé un objet de grosse taille). Devant les cartons de gauche et de droite, il a appris devant chacun deux associations (quand il arrive devant et quand il a déposé l'objet).

Une fois l'apprentissage réalisé, le robot est alors capable de restituer le comportement (figure 6.14). Le robot se rend alors vers le carton du centre. A cet endroit, trois associations avaient été apprises, mais une seule avec la pince grande ouverte, donc il ne bouge pas. Lorsque le robot attrape un objet (le professeur fournit un signal simulant l'objet), une autre association parmi les trois est gagnante. En fonction de la taille de l'objet, le robot se dirige vers le carton de gauche ou de droite. A chacun de ces endroits, deux associations avaient été apprises, mais une seule avec un objet dans la pince. Le robot dépose alors l'objet (le professeur simule l'ouverture de la pince), la seconde association est gagnante. Puis le robot continue la tâche en revenant vers le carton du centre pour y prendre un nouvel objet.

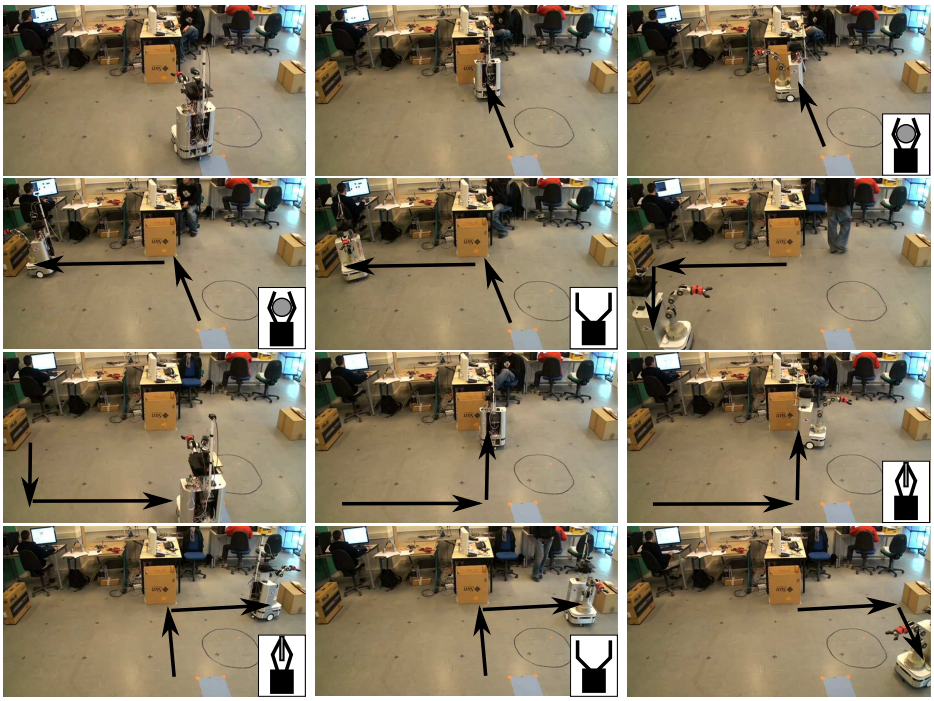


Fig. 6.14: Le robot reproduit seul la tâche apprise. Il arrive devant le carton central. A cet endroit, trois associations avaient été apprises, mais seul une avec la pince grande ouverte, donc il ne bouge pas. Lorsque le robot attrape un objet de petite taille (le professeur fournit alors un signal simulant l'objet), une autre association parmi les trois est gagnante. Il se dirige alors vers le carton de gauche. A cet endroit, deux associations avait été apprises, mais seul une avec la pince faiblement ouverte. Le robot dépose alors l'objet (le professeur simule l'ouverture de la pince), la seconde association est gagnante. Puis le robot continue la tâche en revenant vers le point de départ. Il retourne donc vers le carton du centre pour attraper un nouvel objet de grosseur plus importante (le professeur envoie alors le signal simulant l'objet), le robot se dirige alors vers le carton de droite où il relâche l'objet.

6.3.5 Test sur un robot mobile rangeant des objets

Les tests réalisés ont consisté à permettre à un robot d'apprendre et restituer des comportements mêlant la navigation, l'objet et le geste. De la même manière que précédemment, les objets sont différenciés en fonction de la taille à partir des informations proprioceptives de la pince. Par contre, grâce à l'intégration des travaux du LASA, le robot est capable d'apprendre des gestes avec le bras. Les gestes sont appris dans un premier temps en dehors du reste de l'expérience. Un répertoire de deux gestes a été appris. Cet apprentissage se déroule en manipulant le bras en mode passif. Pour un même geste, trois démonstrations sont ainsi réalisées pour que le robot soit capable de le restituer. Dans un second temps, le robot doit apprendre à se déplacer entre différentes zones de l'environnement dans lesquelles il doit prendre un objet donné par un professeur (carton du centre) et où il doit relâcher l'objet (carton de gauche et de droite). En fonction de l'objet, le robot doit alors le déposer soit à gauche (gros objet) ou à droite (petit objet). De plus, le robot apprend à réaliser un geste particulier avant ou après avoir pris ou relâché un objet.
Le robot apprend à se rendre vers le carton du centre (figure 6.15). Lorsqu'il y arrive, le professeur lui donne un objet de grosse taille, puis lui apprend qu'il doit réaliser un premier geste. Le professeur apprend au robot la direction dans laquelle il doit partir (vers le carton de droite) pour aller déposer l'objet. Une fois arrivé devant le carton de droite, le professeur apprend au robot qu'il doit réaliser un second geste, puis retire l'objet de la pince du robot pour lui apprendre

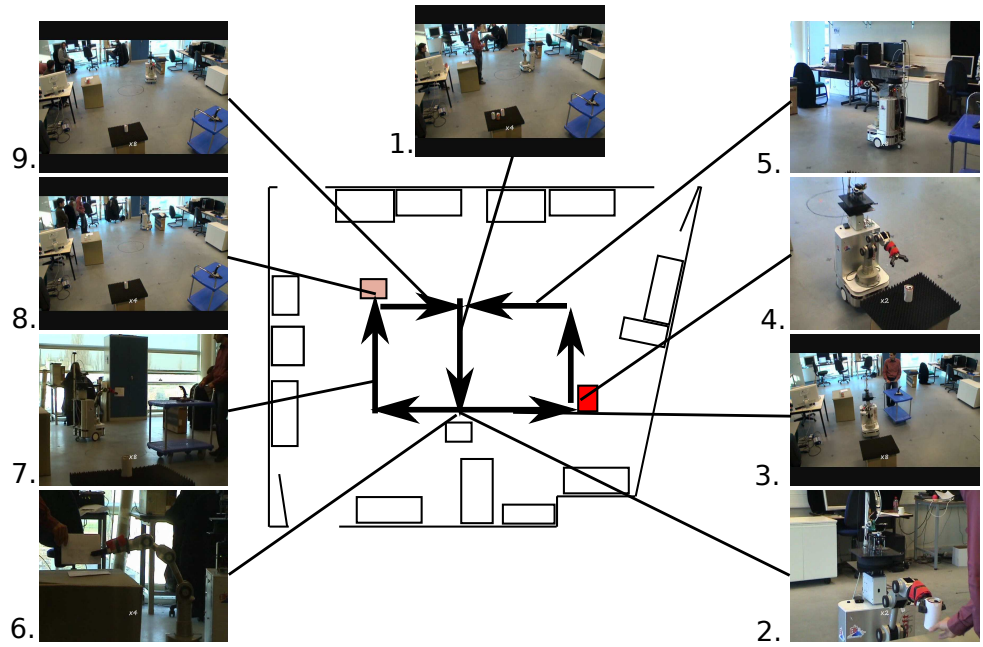


Fig. 6.15: Le robot apprend à se rendre vers le carton du centre (1). Lorsqu'il y arrive, le professeur lui donne un objet de grosse taille, puis lui apprend qu'il doit réaliser un premier geste (2). Le professeur apprend au robot la direction dans laquelle il doit partir (vers le carton de droite) pour aller déposer l'objet (3). Une fois arrivé devant le carton de droite, le professeur apprend au robot qu'il doit réaliser un second geste, puis retire l'objet de la pince du robot pour lui apprendre à le relâcher (4). Ensuite, le professeur apprend au robot à retourner à son point de départ (5) et réalise un apprentissage similaire avec un petit objet à aller déposer sur le carton de gauche (6, 7, 8, 9).

à le relâcher. Ensuite, le professeur apprend au robot à retourner à son point de départ et réalise un apprentissage similaire avec un petit objet à aller déposer sur le carton de droite. Une fois l'apprentissage terminé, le robot est alors capable d'aller chercher un objet, de faire un geste particulier, d'aller le déposer sur un carton en fonction de la taille de l'objet, puis de revenir vers le carton du centre, de prendre un objet et ainsi de suite. Finallement, le robot a appris à ranger des objets en fonction de leur taille tout en réalisant des gestes. Cette expérience à durée environ 1 heure sans que le robot échoue dans sa mission.

6.4 Conclusion

Dans ce chapitre j'ai présenté les outils de conception et de simulation de réseaux de neurones artificiels temps réel. Ces outils permettent de concevoir et d'exécuter de très grand réseaux de neurones composées de plusieurs centaines de milliers de neurones.
En plus de modéliser des architectures neuronales, *Coeos* permet de définir graphiquement la répartition des différents réseaux de neurones d'une architecture sur plusieurs unités de calculs. Ce processus de répartition s'effectue par de simples *glisser-déposer*. *Coeos* propose donc une vision globale d'une architecture distribuée. Cette vue globale permet de fournir au concepteur des informations sur la charge (en nombre de réseaux de neurones de l'architecture) de chacune des unités de calcul, mais également la charge en communication (en nombre de canaux) entre les différents réseaux de neurones. Ces informations visent non seulement à apporter des informations purement technique sur une architecture, mais également à permettre de tester comment

plusieurs boucles sensori-motrices peuvent être exécutées en parallèle. L'apport du parallélisme de plusieurs boucles permet également de tester leurs exécutions respectives à différentes fréquences.
Pour permettre de contrôler la fréquence d'exécution d'une boucle sensori-motrice, le simulateur *Promethe* fournit un mécanisme de jetons temps réel. Ce mécanisme permet ainsi d'exécuter différents réseaux de neurones d'une architecture sous différentes contraintes de temps. Le respect de ces contraintes de temps peut être imposé par les possibilités/limites du matériel lui même, mais également pour économiser du temps de calcul. En effet, il est inutile de prendre mille valeurs par seconde si le matériel ne délivre que dix valeurs par seconde.
Un élément important pour fusionner les données venant de deux réseaux de neurones différents réside dans les propriétés dynamiques du champ de neurones qui permettent de filtrer le bruit présent sur les entrées et de tenir compte des différences de phase et de fréquence d'acquisition. La forme du noyau d'interaction permettent de fusionner (coopération) ou de sélectionner (compétition) des entrées proches ou éloignées. De plus, la mémoire intrinsèque du système permet de réaliser un contrôle stable des moteurs sous réserve que les contraintes de temps du matériel soient respectées. Un tel système permet une prise de décision à partir des réponses asynchrones des boucles sensori-motrices.
Les outils développés ont finalement permis de concevoir et d'exécuter une architecture composée d'environ cinq cent mille neurones découpés en vingt deux réseaux répartis sur quatre ordinateurs. Une telle architecture a permis à un robot d'apprendre un comportement complexe mêlant la navigation, le geste et l'objet.
Le développement et l'exécution de très grandes architectures neuronales réparties soulève de nouveaux problèmes liés à la parallélisation. *Coeos* apporte une solution pour développer et visualiser facilement de telles architectures. Des améliorations peuvent être apportée pour faciliter le développement, le déploiement, l'exécution et le débogage des différents réseaux de neurones composant le macro réseau.
Même si *Coeos* permet de gérer les liens de communication pour des architectures neuronales parallèles, le découpage de l'architecture doit toujours être réalisé par le concepteur et doit être soigneusement pensé dès le début de la conception. Une voie d'amélioration serait que *Coeos* soit capable de détecter les différentes boucles sensori-motrices d'une architecture et de réaliser le découpage automatiquement. Le découpage pourrait s'effectuer en fonction de la consommation de calcul nécessaire par l'architecture. Il est souvent nécessaire de répartir une même boucle sensori-motrice en plusieurs réseaux de neurones pour en améliorer les performances. De plus, le découpage et le choix du mode de communication (bloquant, non bloquant) est toujours définis par le développeur, il est toujours possible de rencontrer des problèmes d'inter blocage entre les différents réseaux de neurones d'une architecture. Pour apporter une réponse à ce type de problème, *Coeos* pourrait intégrer des fonctionnalités de vérification de la circulation des flux d'informations dans les réseaux de neurones.
La répartition d'une architecture pose la question de l'ordonnancement des groupes qui la composent. En effet, si dans un même réseau de neurones les groupes sont exécutés par vagues et de manière synchrone (un groupe ne peut s'exécuter que si ses prédécesseurs ont terminé), les communications bloquantes ou non bloquantes permettent d'autres modes de fonctionnement. Par exemple, quand une communication est bloquante, doit-elle bloquer tout le réseau de neurone concerné ou doit-elle seulement bloquer la branche réceptionnant les données ? Si la communication est non bloquante, l'architecture doit elle continuer de calculer à partir des dernières informations reçues ou doit-il y avoir une remise à zéro ? Selon la solution choisie, le comportement d'une architecture peut changer, induisant un changement de comportement du

robot pouvant entrainer l'échec de la tâche. La distribution d'une architecture a donc un impact non négligeable sur les algorithmes et sur leur validation. L'utilisation de communications non bloquante, permet d'exécuter à différentes fréquences plusieurs boucles sensori-motrices en parallèle et d'étudier comment plusieurs réponses asynchrones peuvent être fusionnées. Là où l'on peut voir un problème (modification du comportement d'une architecture), on peut également voir de nouvelles possibilités (exécution de boucles sensori-motrices en parallèle).

Chapitre 7

Conclusion et perspectives

Cette thèse m'a permis de développer à la fois de nouveaux modèles pour l'apprentissage de comportements - dans le temps et dans l'espace -, de nouveaux outils pour maîtriser des réseaux de neurones de très grande taille. A travers les limitations du système actuel, ces travaux m'ont permis de discuter les éléments importants pour un système de sélection de l'action.

7.1 Conclusion et principaux apports de la thèse

J'ai proposé des outils permettant la conception, l'exécution et la communication de réseaux de neurones répartis et temps réel. J'ai développé un outil logiciel, *Coeos* qui permet la conception et le développement de très grands réseaux de neurones massivement parallélisés sur plusieurs unités de calcul. Une nouvelle librairie de communication a été développée pour améliorer la communication entre les différents réseaux de neurones d'une même architecture neuronale. Cette libraire offre de meilleures performances ainsi que d'avantage d'informations sur l'état des connexions réseaux. Ces outils sont aujourd'hui utilisés dans tous les travaux de l'équipe en cours avec des réseaux de neurones. Ils ont été pleinement utilisés dans l'intégration de plusieurs travaux dans une même architecture neuronale découpée en une vingtaine de réseaux et composée d'environ cinq cents mille neurones.
Je suis parti de comment l'imitation pouvait émerger de mécanismes plus bas niveau. J'ai ensuite appliqué et testé un modèle qui permet l'émergence de comportement d'imitation de bas niveau. Ce modèle est construit comme un homéostat qui tend à équilibrer par l'action ses informations perceptives frustres (détection du mouvement ou de couleur). Ce modèle implique que le robot ait au préalable associé les positions visuelles de son effecteur avec les informations proprioceptives de ses moteurs.
J'ai ensuite présenté et testé deux modèles permettant l'apprentissage de séquences temporelles. Le premier apprend en ligne le timing de séquences temporelles simples (séquences n'ayant pas d'éléments répétés). Le second modèle repose sur les propriétés d'un réservoir de dynamiques, il apprend en ligne des séquences complexes (séquences ayant des éléments répétés). Finalement, une architecture apprenant le timing d'une séquence complexe a été proposée. Avec l'ajout de dynamiques internes, l'architecture crée des états cachés. Ceux-ci permettent de lever les ambiguïtés des séquences. Cette architecture a permis d'apprendre des séquences de gestes sur un robot.
Un comportement n'étant pas seulement le contrôle d'un bras manipulateur mais également le déplacement dans l'environnement, j'ai présenté deux architectures permettant d'apprendre et de restituer une même tâche de navigation. La première encode la tâche de navigation sous forme d'associations lieux-mouvements. Les lieux sont appris à partir d'informations visuelles extraites d'une caméra balayant le panorama du robot. Les mouvements qui y sont associés sont fournis par une boussole électronique jouant le rôle d'information proprioceptive. La seconde architecture encode les comportements sous forme de séquences temporelles de mouvements. Les éléments des séquences sont les informations proprioceptives du robot. En appliquant cette seconde architecture non plus sur l'apprentissage de gestes avec un bras robotique, mais sur l'apprentissage d'une trajectoire avec un robot mobile, le temps des transitions de mouvements n'est plus négligeable. J'ai alors proposé un mécanisme de resynchronisation des dynamiques internes à partir de signaux externes. Ce mécanisme permet alors de retrouver les états cachés précédemment appris afin de correctement restituer le comportement de navigation. Ce mécanisme ajouté au contrôle de l'apprentissage des transitions a permis d'amorcer une séquence par un état intermédiaire, d'apprendre plusieurs séquences, où même de les combiner.
A partir de ces deux stratégies, j'ai mené une étude sur leur exécution en parallèle dans une

même architecture. Dans un premier temps, j'ai montré comment une stratégie peut en compléter une autre lorsqu'une modalité (la vision) est en défaut. J'ai ensuite étudié comment peut être réalisée la fusion et la sélection de l'action avec un champ de neurones dynamique. Les tests ont permis de mettre en évidence les limitations de chacune des stratégies. La principale limitation est l'apprentissage non adaptatif des stratégies. Les travaux en cours visent à améliorer en ce sens les algorithmes. J'ai mis en évidence que les propriétés de fusion et de sélection dépendent principalement de la taille des attracteurs générés par les réponses des différentes stratégies. Je discute alors de rendre dynamique la taille des attracteurs. Elle pourrait varier à partir d'un paramètre de vigilance. Ce paramètre proviendrait de structures de plus haut niveau régulant le comportement du robot à partir de motivations/buts particuliers. De plus, la force d'attraction pourrait être pondérée par la saillance des réponses des différentes stratégies représentant alors un niveau de confiance sur chacune des réponses. J'ai discuté également de la pertinence de traiter des informations hétérogènes (temporelles vs spatiales) et de considérer l'hippocampe en deux tranches distinctes. Je discute alors d'une fusion des deux tranches en une seule. L'ensemble des informations (visuelles, motrices, dynamiques internes) pourraient se fusionner en entrée de l'hippocampe dans le cortex enthorinal. Ce nouveau code pourrait alors être associé à un mouvement comme cela est fait avec les associations lieux-mouvements. Ceci constituerait alors un comportement réactif (réponses rapides). L'hippocampe permettrait l'apprentissage de séquences temporelles de ces codes prédisant alors la suite du comportement du robot. Ceci constituerait un comportement proactif (réponses à plus long terme). Néanmoins, cette fusion ne résout pas la question de la sélection de l'action.

7.2 Perspectives

Les tests réalisés sur l'émergence d'un comportement d'imitation de bas niveau a montré que l'apprentissage de la coordination visuo-motrice ne convergeait pas complètement. En effet, l'association des informations proprioceptives du bras robotique est réalisée uniquement pour une position visuelle précise codée sous forme d'un neurone. D'une position visuelle à l'autre, il est alors possible de ne pas retrouver une posture proche du bras. Cette partie du travail fait actuellement l'objet de travaux avec l'utilisation de nouveaux algorithmes utilisant des attracteurs dans l'espace moteur activés par des sensations visuelles.
Les tests réalisés sur la fusion/sélection de l'action ont permis de mettre en évidence les limitations des deux stratégies utilisées (associations lieux-mouvements et séquences temporelles). La première limitation provient de l'apprentissage des séquences temporelles. En effet, celui-ci est réalisé en un coup et n'est plus modifiable par la suite. Il est alors indispensable que cette stratégie soit adaptative de manière à prendre en compte les mouvements finaux du robot après une prise de décision. Cette adaptation implique alors de faire converger les apprentissages sur le bon timing. Un tel algorithme d'apprentissage est alors très complexe. En effet, s'il peut paraître trivial de modifier des poids de connexions, il faut tout d'abord que le système soit capable de détecter ce qu'il doit modifier et quand. Par exemple, l'architecture a appris la transition "1 - 2" et on veut corriger le robot pour qu'il apprenne "1 - - - 2". Le robot détecte "1". Après un certain temps, le robot va déclencher "2", il applique alors la commande correspondante. Le professeur voit alors le robot faire "2", mais trop tôt alors il le corrige et le remet en "1". Comment l'architecture peut elle détecter que la correction du professeur correspond à l'état précédent et pas à la suite de la séquence ? Cette détection peut alors être en partie traitée par le mécanisme de resynchronisation qui permet de retrouver les états cachés. Alors en faisant l'hypothèse que le robot est capable de retrouver l'état dans lequel il était, comment retrouve-t-il le timing duquel

il doit repartir ?
Les dynamiques internes utilisées pour la création des états cachés des séquences temporelles sont générées par un ensemble d'oscillateurs. Ces oscillateurs permettent d'apporter un contexte interne suffisamment riche pour lever les ambiguïtés des séquences, mais il en résulte néanmoins une dynamique cyclique. Une amélioration possible serait de remplacer ces oscillateurs par un réseau exhibant une dynamique chaotique. Une telle dynamique à un état différent à chaque instant ; ceci apporterait donc la richesse suffisante pour la création des états cachés. De plus, une dynamique chaotique étant déterministe, elle permettrait alors d'être utilisée avec le mécanisme de resynchronisation.
Les dynamiques internes permettant la resynchronisation des séquences temporelles reposent sur un mécanisme de création de *chunks*. Dans mes travaux, ces *chunks* sont engrammésp ar des neurones. Une évolution du modèle serait alors d'étudier dans quelle mesure un tel mécanisme est neurobiologiquement plausible, sous quelle forme ces *chunks* peuvent exister : sont-ils codés par des neurones élémentaires ou sous forme de réseaux plus complexes ?
La stratégie d'associations lieux-mouvement ne permet pas de reconnaître rapidement un lieu. En effet, l'apprentissage et la reconnaissance d'un lieu sont calculés après chaque panorama visuel balayé par la caméra. Or, ce temps de balayage est important puisqu'il est d'environ sept secondes. Il serait intéressant de ne plus faire la reconnaissance de lieux après chaque panorama, mais après chaque prise de vue. Si l'amélioration de la vitesse du balayage du panorama permet de reconnaître plus rapidement un lieu, elle ne permet pas au robot d'avancer beaucoup plus vite. En effet, lorsque la vitesse du robot est trop importante, la caméra restitue des images floues. Il y a alors ici un problème technologique. Il serait alors intéressant d'avoir des caméras permettant de capturer plusieurs centaines d'images par secondes contrairement au matériel actuel qui permet trente images par seconde au maximum.
Du coté de la reconnaissance de lieux, une fois qu'un lieu est appris, il n'est plus modifié. Si l'environnement change nettement (objets déplacés ou même lumière du jour changeante), alors le lieu n'est plus reconnu et le robot doit en apprendre un nouveau. Il serait intéressant que cette stratégie puisse adapter un lieu déjà appris aux changements environnementaux.
La stratégie d'associations lieux-mouvements permet l'adaptation des mouvements dans les différents lieux déjà appris. Mais un robot devant apprendre plusieurs tâches (aller manger, aller boire) peut être mené à associer plusieurs mouvements à un même lieu. Un tel mécanisme entraîne une ambiguïté sur le mouvement à réaliser. Il est imaginable de lever cette ambiguïté à partir de motivations permettant de rejoindre un but particulier. Se pose alors la question de l'influence des motivations. Biaisent-elles la reconnaissance des lieux ? En effet, il ne paraît pas insensé que les cellules de lieux menant à de la nourriture aient une activité amplifiée par la sensation de faim du robot. Cela suppose qu'en chaque lieu les mouvements associés sont en direction des lieux menant à la nourriture. Les motivations biaisent-elles les mouvements prédits ? Cela revient alors à ne plus seulement associer un lieu avec un mouvement, mais à associer un lieu et une motivation avec un mouvement. En faisant l'hypothèse qu'en un lieu il est possible d'associer plusieurs mouvements, les motivations pourraient être utilisées pour amplifier sélectivement l'activité du mouvement correspondant.
Une perspective interessante de mes travaux est le développement d'un modèle pour la sélection de l'action. Comme les tests l'ont montrés, la fusion seule des réponses des différentes stratégies permet une prise de décision, mais sans permettre au robot d'atteindre un objectif particulier. Il y a alors deux améliorations qui pourraient permettre la prise de décision. La première est l'ajout d'une valeur de confiance sur les réponses des différentes stratégies. Cette valeur de confiance modulerait les activités des différentes réponses à partir de renforcements passés. Mais

ce renforcement ne peut pas seulement "noter" des réponses motrices, car dans l'absolu, "aller à gauche" n'est jamais un mauvais mouvement. Alors, on sent bien qu'ici il manque d'autres informations permettant l'évaluation du comportement. Les stratégies prédisent les mouvements à réaliser à partir d'informations sensorielles et/ou proprioceptives. Ces informations forment alors un contexte plus ou moins riche qui permet au robot de se localiser dans son comportement actuel. Le renforcement d'un mouvement prédit pourrait aussi se faire par rapport à ce contexte sensoriel. Ceci reviendrait à rendre les différentes stratégies adaptatives comme évoqué précédemment en modulant leurs poids à partir d'un signal de renforcement. Mais cela pose la question de définir proprement ce signal de renforcement. En effet, on peut imaginer au moins deux réponses à cette question. La première, est que le renforcement est fourni par le professeur soit directement en donnant un signal "c'est bien" ou "ce n'est pas bien", soit en corrigeant le robot (le tirer avec une laisse). Dans les deux cas, le robot évaluerait chacune des stratégies avec la correction fournie. La seconde solution serait que le robot détermine seul le renforcement à partir des informations (sensorielles) à sa disposition lui permettant d'évaluer s'il se rapproche ou non de son objectif. Cette seconde solution implique alors d'introduire une notion de but [Hasson et Gaussier, 2010].

Dans mes travaux, la sélection de l'action est réalisée par une règle donnée a priori (aller vers l'attracteur le plus proche). Une première amélioration serait de ne plus donner de règle sur comment sélectionner le bon mouvement. La taille des attracteurs pourrait être modifiée dynamiquement. Une première implémentation consisterait à faire grandir les attracteurs jusqu'à ce qu'ils attirent le robot. Dans un second temps, on pourrait introduire un paramètre de vigilance permettant de contrôler la taille des attracteurs. Plus le robot serait vigilent, plus les attracteurs rétréciraient et réciproquement. De cette manière, un attracteur permettant au robot d'aller satisfaire un manque de nourriture pourrait être grossi par une motivation particulière, alors que des attracteurs ne répondant pas à une satisfaction particulière pourraient être diminués. De cette manière, ce ne serait plus directement le plus proche attracteur/but qui serait choisi, mais le plus pertinent vis à vis d'une motivation.

Au cours de mes travaux, j'ai développé des outils permettant la modélisation, la communication et l'exécution de réseaux de neurones répartis sur plusieurs unités de calcul. Ces outils ont permis de concevoir de très grandes architectures neuronales réparties en une vingtaine de réseaux de neurones. Si maintenant les outils permettant leur développement existe, il manque des outils permettant leur déploiement, leur monitoring/debug et le traitement des résultats obtenus. Il faudra faire particulièrement attention que de tels outils ne perturbent pas l'exécution des architectures, car ils impliquent des communications réseaux (monitoring distant) et des accès disque dur (sauvegarde de résultats). De plus, les différents outils utilisés fonctionnent sur le système d'exploitation Linux en environnement utilisateur (et non en mode noyau), ce qui implique que ces différents accès font des appels systèmes qui consomment un temps non négligeable sur les exécutions. Il faut alors non seulement faire attention aux communications, mais aussi au respect des contraintes de temps réel des architectures. Aujourd'hui, les différents réseaux d'une architecture sont déployés manuellement par les chercheurs. Ceci demande alors un effort non négligeable pour ne pas oublier de déployer tel ou tel réseau de neurones après l'avoir modifié. De plus, lors de l'exécution de l'architecture, le lancement de chaque réseau en mode "debug" (avec visualisation des activités de neurones sur une interface graphique) ou en mode aveugle doit être décidé a priori. Il devient de plus en plus difficile de visualiser efficacement les activités d'une architecture neuronale lorsqu'il y a une fenêtre pour chaque réseau distribué. Pour visualiser vingt réseaux de neurones répartis sur quatre machines de calcul, il faudrait cinq grands écrans par machine ; donc vingt écrans. Ceci n'est absolument pas réaliste

surtout quand l'une des machines se trouve sur le robot même. Il est alors nécessaire de mettre en place un nouvel outil de monitoring à distance. Cet outil, de la même manière que *Coeos* pourrait fonctionner sur un mécanisme d'onglets. Un premier onglet serait une vue globale de l'architecture permettant de visualiser la bonne exécution de chacun des réseau à travers un jeu de couleurs (un réseau de neurones en rouge serait en défaut alors qu'en vert pour signifier que tout va bien. On peut imaginer des couleurs intermédiaires pour d'autres niveaux d'erreur). D'autres onglets pourraient permettre de visualiser en même temps des activités de différents groupes de neurones répartis sur plusieurs machines. D'autre part, pour traiter correctement les résultats, les différentes machines de calculs utilisées pour exécuter une même architecture doivent être synchronisées manuellement sur un serveur de temps (service *ntp* : net time protocol). Cette phase manuelle est indispensable pour s'assurer au maximum de pouvoir recouper les différentes activités des neurones entre elles pour pouvoir les traiter de manière cohérente. Finalement, de tel outils peuvent paraître secondaires, car ils ne sont pas nécessaires au bon fonctionnement du robot, mais ils sont vitaux pour le chercheur qui doit s'en assurer et sortir des résultats le démontrant. La parallélisation d'une architecture neuronale soulève la question de la resynchronisation des boucles sensori-motrices. En effet, l'utilisation d'une horloge externe est acceptable pour la publication de résulats et le monitoring. Mais pour le fonctionnement même d'une architecture, c'est la dynamique des neurones qui doit permettre la resynchronisation des boucles sensori-motrices sans horloge extérieure.

Chapitre 8

Références bibliographiques

Bibliographie personelle

Chapitre de livre

[Lagarde et al., 2010] Lagarde, M., Andry, P., Gaussier, P., Boucenna, S., and Hafemeister, L. (2010). Proprioception and imitation : on the road to agent individuation. In Sigaud, O. and Peters, J., editors, *From Motor Learning to Interaction Learning in Robots*, volume 264, pages 43–63. Springer.

Publications internationnales avec actes

[Lagarde et al., 2007a] Lagarde, M., Andry, P., and Gaussier, P. (2007). The role of internal oscillators for the one-shot learning of complex temporal sequences. In de Sa, J. M., Alexandre, L. A., Duch, W., and Mandic, D., editors, *Artificial Neural Networks – ICANN 2007*, volume 4668 of *LNCS*, pages 934–943. Springer.

[Lagarde et al., 2008b] Lagarde, M., Andry, P., and Gaussier, P. (2008). Distributed real time neural networks in interactive complex systems. In *proceedings of the IEEE International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST 08)*, pages 95–100.

[Lagarde et al., 2008c] Lagarde, M., Andry, P., Gaussier, P., and Giovannangeli, C. (2008). Learning new behaviors : Toward a control architecture merging spatial and temporal modalities. In *Workshop on Interactive Robot Learning - International Conference on Robotics : Science and Systems (RSS 2008)*.

[Lagarde et al., 2009] Lagarde, M., Andry, P., and Gaussier, N. (2009). Learning paths as a sequence of sensori-motor associations. In *Proceedings of the ninth international conference on Epigenetic robotics, EPIROB09*, pages 217–218 Lund University Cognitive Studies.

Participation à une publication présentée en conférence internationnale avec actes

[Andry et al., 2008a] Andry, P., Gaussier, P., Lagarde, M., and Boucenna, S. (2008). Proprioception and imitation : on the road to agent individuation. In *IEEE/RSJ International Conference on Robots and Systems (IROS 2008) Session : From motor to interaction learning robots*.

Présentation orale en conférence nationnale sans actes

[Lagarde et al., 2008d] Lagarde, M., Andry, P., and Gaussier, P. (2008). Apprentissage par imitation de nouveaux comportements en robotique autonome. JNRH08 : Journées Nationales de la Robotique Humanoïde

Posters en conférences nationales sans actes

[Lagarde et al., 2007b] Lagarde, M., Andry, P., and Gaussier, P. (2007). Le rôle d'oscillateurs internes pour l'apprentissage en un coup de séquences temporelles complexes. JNRR07 : Journées Nationales de la Recherche en Robotique

[Lagarde et al., 2008e] Lagarde, M., Andry, P., and Gaussier, P. (2008). Learning by imitation of complex behaviors in autonomous robotics. Doctoriales de Cergy-Pontoise

[Lagarde et al., 2008f] Lagarde, M., Andry, P., Gaussier, P., Boucenna, S., and Hafemeister, L. (2008). Learning from social interaction. DIGITEO

Références

[Albus, 1975] Albus, J. S. (1975). A new approach to manipulator control : the cerebellar model articulation controller (cmac. *Journal of Dynamic Systems, Measurement, and Control*, 97:220–227.

[Alexander *et al.*, 1986] Alexander, G. E., DeLong, M. R. et Strick, P. L. (1986). Parallel organization of functionally segregated circuits linking basal ganglia and cortex. *Annual Review of Neuroscience*, 9:357–381.

[Amari, 1977] Amari, S. (1977). Dynamic of pattern formation in lateral-inhibition type by neural fields. *Biological Cybernetics*, 27:77–87.

[Andry, 2002a] Andry, P. (2002a). Thèse : Apprentissage et interactions via imitation : application d'une approche développementale à la robotique autonome.

[Andry, 2002b] Andry, P. (2002b). *Thèse : Apprentissage et interactions via imitation : application d'une approche développementale à la robotique autonome*. Thèse de doctorat, University of Cergy-Pontoise.

[Andry *et al.*, 2002] Andry, P., Gaussier, P. et Nadel, J. (2002). From visuo-motor coordination to imitation : an autonomous robot perspective. *In Workshop on Dynamic Motor representations*, Institut Henri Poincaré, IHP, France. Conférence invitée.

[Arrowsmith et Place, 1990] Arrowsmith, D. et Place, C. (1990). *An Introduction to Dynamical Systems*. Cambridge University Press.

[Bailly, 2007] Bailly, D. (2007). Apprentissage non supervisé d'association visuo-motrices pour un robot autonome. Rapport technique, ETIS.

[Bakker et Kuniyoshi, 1996] Bakker, P. et Kuniyoshi, Y. (1996). Robot see, robot do : An overview of robot imitation. *In In AISB96 Workshop on Learning in Robots and Animals*, pages 3–11. springer.

[Balkenius *et al.*, 2009] Balkenius, C., Morén, J., Johansson, B. et Johnsson, M. (2009). Anticipatory models in gaze control : a developmental model. *Adv. Eng. Informat.*

[Bandura, 1969] Bandura, A. (1969). Social learning theory of identificatory processes. *Handbook of socialization theory and research*, pages 213–262.

[Bandura, 1971] Bandura, A. (1971). *Psychological Modeling : Conflicting Theories*.

[Banquet *et al.*, 1997] Banquet, J. P., Gaussier, P., Dreher, J. C., Joulain, C., Revel, A., Günther, W. et Modélisation, N. E. (1997). Space-time, order, and hierarchy in fronto-hippocampal system : A neural basis of personality. *In In Mattews, G., (Ed.), Cognitive Science Perspectives on Personality and Emotion. Elsevier Science BV*, pages 123–189. Elsevier Science BV.

[Beeman *et al.*, 2007] Beeman, D., Wang, Z., Edwards, M., Bhalla, U., Cornelis, H. et Bower, J. (2007). The genesis 3.0 project : a universal graphical user interface and database

for research, collaboration, and education in computational neuroscience. *BMC Neuroscience*, 8(Suppl 2).

[Beer, 1994] Beer, R. D. (1994). On the dynamics of a continuous hopfield neuron with self-connection. Rapport technique CES-94-1, Dept. of Computer Engineering and Science & Dept. of Biology, Case Western Reserve University, Cleveland, OH.

[Berthouze et Tijsseling, 2006] Berthouze, L. et Tijsseling, A. (2006). A neural model for context-dependent sequence learning. *Neural Processing Letters*, 23(1):27–45.

[Billard *et al.*, 1998] Billard, A., Dautenhahn, K. et Hayes, G. (August 1998). Experiments on human-robot communication with robota, an imitative learning and communicating robot. *Proceedings of "Socially Situated intelligence" Workshop, part of the Fifth International Conference on Simulation of Adaptive Behavior*.

[Boné *et al.*, 1998] Boné, R., Crucianu, M. et Asselin de Beauville, J. (1998). *Yet Another Neural Network Simulator*.

[Boucenna *et al.*, 2008] Boucenna, S., Gaussier, P. et Andry, P. (2008). What should be taught first : the emotional expression or the face ? *In 8th International conference on Epigenetic Robotics, EPIROB*. Lucs.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

[Bullock et Grossberg, 1989] Bullock, D. et Grossberg, S. (1989). *VITE and FLETE : neural modules for trajectory formation and postural control*. Elsevier Science Publishers.

[Buonomano et Mauk, 1994] Buonomano, D. V. et Mauk, M. D. (1994). Neural network model of the cerebellum : temporal discrimination and the timing of motor responses. *Neural Comput.*, 6(1):38–55.

[Byrne et Russon, 1998] Byrne, R. et Russon, A. (1998). Learning by imitation : a hierarchical approach. *Behavioral and Brain Science*, 21:667–721.

[Calinon, 2007] Calinon, Sylvain. ; Billard, A. (2007). What is the teacher's role in robot programming by demonstration ? - toward benchmarks for improved learning. *In Interaction Studies. Special Issue on Psychological Benchmarks in Human-Robot Interaction*, volume 8.

[Calinon et Billard, 2007] Calinon, S. et Billard, A. (2007). Active teaching in robot programming by demonstration. *In Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 702–707.

[Chaumette et Hutchinson, 2006] Chaumette, F. et Hutchinson, S. (2006). Visual servo control, part i : Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90.

[Chaumette et Hutchinson, 2007] Chaumette, F. et Hutchinson, S. (2007). Visual servo control, part ii : Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118.

[Cheng et Kuniyoshi, 2000] Cheng, G. et Kuniyoshi, Y. (2000). Complex continuous meaningful humanoid interaction : A multi sensory-cue based approach. *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA2000)*, pages 2235–2242.

[Cohen *et al.*, 1990] Cohen, A., Ivry, R. I. et Keele, S. W. (1990). Attention and structure in sequence learning. *Journal of Experimental Psychology : Learning, Memory, and Cognition*, 16(1):17–30.

[Cohen et Frank, 2009] Cohen, M. et Frank, M. (2009). Neurocomputational models of basal ganglia function in learning, memory and choice. *Behavioural Brain Research*, (199):141–156.

[Corke, 1994] Corke, P. I. (1994). Visual control of robot manipulators – a review. *In Visual Servoing*, pages 1–31. World Scientific.

[Daucé et Doyon, 1998] Daucé, E. et Doyon, B. (1998). Novelty learning in a discrete time chaotic network. *In Proceedings of International Conference on Artificial Neural Networks (ICANN)*, volume 2, pages 1051–1056.

[Daucé *et al.*, 2002] Daucé, E., Quoy, M. et Doyon, B. (2002). Resonant spatiotemporal learning in large random recurrent networks. *Biological Cybernetics*, 87(3):185–198.

[Dautenhahn, 1995] Dautenhahn, K. (1995). Getting to know each other - artificial social intelligence for autonomous robots. *Robotics and Autonomous System*, 16(2-4):333–356.

[Degallier *et al.*, 2006] Degallier, S., Santos, C., Righetti, L. et Ijspeert, A. (2006). Movement generation using dynamical systems : a humanoid robot performing a drumming task. *In IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS06)*.

[Demuth et Beale, 2006] Demuth, H. et Beale, M. (2006). *Neural Network Toolbox : For Use with MATLAB*. Mathworks.

[Dolle *et al.*, 2008] Dolle, L., Khamassi, M., Girard, B., Guillot, A., et Chavarriaga, R. (2008). Analyzing interactions between navigation strategies using a computational model of action selection. volume Spatial Cognition VI, pages 71–86. Springer.

[Dominey, 2005] Dominey, P. F. (2005). Emergence of grammatical constructions : evidence from simulation and grounded agent experiments. *Connection Science*, 17(3-4):289–306.

[Dominey *et al.*, 1995] Dominey, P. F., Arbib, M. A. et Joseph, J.-P. (1995). A model of cortico-striatal plasticity for learning oculomotor associations and sequences. *Journal of Cognitive Neuroscience*, 7(3):311–336.

[Dominey et Ramus, 2000] Dominey, P. F. et Ramus, F. (2000). Neural network processing of natural language : I. sensitivity to serial, temporal, and abstract structure of language in the infant. *Language and Cognitive Processes*, 15(1):45–85.

[Doya, 2000] Doya, K. (2000). Complementary roles of the basal ganglia and the cerebellum in learning and motor control. *Current opinion in neurobiology*, 10(6):732–739.

[Duran *et al.*, 2007] Duran, B., Metta, G. et Sandini, G. (2007). Emergence of smooth pursuit using chaos. *Self-Adaptive and Self-Organizing Systems, International Conference on*, 0:269–272.

[Durbin et Rumelhart, 1989] Durbin, R. et Rumelhart, D. E. (1989). Product units : a computationally powerful and biologically plausible extension to backpropagation networks. *Neural Comput.*, 1(1):133–142.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[Gaussier, 1992] Gaussier, P. (1992). *Simulation d'un système visuel comprenant plusieurs aires corticales : Application à l'analyse de scènes.* Thèse de doctorat, University of Paris Sud Centre d'Orsay.

[Gaussier *et al.*, 1997] Gaussier, P., Moga, S., Banquet, J. et Quoy, M. (1997). From perception-action loops to imitation processes : A bottom-up approach of learning by imitation. *In Socially Intelligent Agents*, pages 49–54, Boston. AAAI fall symposium.

[Gaussier *et al.*, 1998] Gaussier, P., Moga, S., Banquet, J.-P. et y, M. Q. (1998). From perception-action loops to imitation processes. *Applied Artificial Intelligence (AAI)*, 1(7):701–727.

[Giovannangeli, 2007] Giovannangeli, C. (2007). Thèse : Navigation autonome bio-inspirée en environnement intérieur et extérieur : Apprentissages sensori-moteurs et planification dans un cadre interaction.

[Giovannangeli et Gaussier, 2007] Giovannangeli, C. et Gaussier, P. (2007). Human-robot interactions as a cognitive catalyst for the learning of behavioral attractors. *In 16th IEEE International Symposium on Robot and Human Interactive Communication 2007*, pages 1028–1033, Jeju, South Korea.

[Giovannangeli et Gaussier, 2008] Giovannangeli, C. et Gaussier, P. (2008). Interactive teaching for vision-based mobile robot : a sensory-motor approach. *IEEE Transactions on Man, Systems and Cybernetics, Part A : Systems and humans*.

[Giovannangeli *et al.*, 2006] Giovannangeli, C., Gaussier, P. et Banquet, J.-P. (2006). Robustness of visual place cells in dynamic indoor and outdoor environment. *International Journal of Advanced Robotic Systems*, 3(2):115–124.

[Girard, 2003] Girard, B. (2003). *Intégration de la navigation et de la sélection de l'action dans une architecture de contrôle inspirée des ganglions de la base*. Thèse de doctorat, LIP6/AnimatLab, Université Pierre et Marie Curie.

[Girard *et al.*, 2002] Girard, B., Cuzin, V., Guillot, A., Gurney, K. N. et Prescott, T. J. (2002). Comparing a bio-inspired robot action selection mechanism with winner-takes-all. *In* Hallam, B., Floreano, D., Hallam, J., Hayes, G. et Meyer, J.-A., éditeurs : *From Animals to Animats 7. Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*, pages 75–84. The MIT Press.

[Girard *et al.*, 2005] Girard, B., Filliat, D., Meyer, J.-A., Berthoz, A. et Guillot, A. (2005). Integration of navigation and action selection functionalities in a computational model of cortico-basal ganglia-thalamo-cortical loops. *Adaptive Behavior*, 13(2):115–130.

[Grossberg, 1999] Grossberg, S. (1999). How hallucinations may arise from brain mechanisms of learning, attention, and volition. *JOURNAL OF THE INTERNATIONAL NEUROPSYCHOLOGICAL SOCIETY*, 6:579–588.

[Hasson et Gaussier, 2010] Hasson, C. et Gaussier, P. (2010). Generical frustration as a regulatory mechanism for motivated navigation. *In International Conference on Intelligent Robots and Systems (submitted)*. soumis.

[Hayes et Demiris, 1994] Hayes, G. M. et Demiris, J. (1994). A robot controller using learning by imitation. *In* Borkowski, A. et Crowley, J. L., éditeurs : *Proceedings o the second international symposium on intelligent robotic systems*, pages 198–204.

[Hersch et Billard, 2006] Hersch, M. et Billard, A. (2006). A biologically-inspired model of reaching movements. *In Proceedings of the First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 1067–1072, pisa.

[Heyes, 2001] Heyes, C. (2001). Causes and consequences of imitation. *TRENDS in Cognitive Sciences*, 5(6):253–261.

[Hill et Park, 1979] Hill, J. et Park, W. T. (1979). Real time control of a robot with a mobile camera. *In 9th International Symposium on Industrial Robot*, pages 233–246, Washington, DC.

[Hopfield, 1984] Hopfield, J. (1984). Neurons with graded response properties have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81:3088–3092.

[Hubel et Wiesel, 1965] Hubel, D. H. et Wiesel, T. N. (1965). Binocular interaction in striate cortex of kittens reared with artificial squint. *Neurophysiology*, 28(6):1041–1059.

[Hutchinson *et al.*, 1996] Hutchinson, S., Hager, G. et Corke, P. (1996). A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12:651–670.

[Ijspeert *et al.*, 2002a] Ijspeert, A., Nakanishi, J. et Schaal, S. (2002a). Learning Attractor Landscapes for Learning Motor Primitives. *In* Becker, S., Thrun, S. et Obermayer, K., éditeurs : *Advances in Neural Information Processing Systems 15 (NIPS2002)*, pages 1547–1554.

[Ijspeert *et al.*, 2002b] Ijspeert, A., Nakanishi, J. et Schaal, S. (2002b). Movement imitation with nonlinear dynamical systems in humanoid robots. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2002)*, pages 1398–1403. (received the ICRA2002 best paper award).

[Ijspeert *et al.*, 2003] Ijspeert, A., Nakanishi, J. et Schaal, S. (2003). learning attractor landscapes for learning motor primitives. *In advances in neural information processing systems 15*, pages 1547–1554. cambridge, ma : mit press.

[Iossifidis et Schöner, 2006] Iossifidis, I. et Schöner, G. (2006). Dynamical systems approach for the autonomous avoidance of obstacles and joint-limits for an redundant robot arm. *In Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 580–585.

[Ito et Tani, 2004] Ito, M. et Tani, J. (2004). Joint attention between a humanoid robot and users in imitation game. *In 3rd Int. Conf. on Development and Learning (ICDL'04), Proceedings.*

[Ivry *et al.*, 2002] Ivry, R., Spencer, R., Zelaznik, H. et Diedrichsen, J. (2002). The cerebellum and event timing. *The Cerebellum : Recent Developments in Cerebellar Research*, 978:302–317.

[Jaeger, 2001] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science.

[Joel *et al.*, 2002] Joel, D., Niva, Y. et Ruppin, E. (2002). Actor-critic models of the basal ganglia : new anatomical and computational perspectives. *Neural Networks*, 15(4–6):535–547.

[K. Dautenhahn, 2002] K. Dautenhahn, A. B. (2002). *Games children with autism can play with robota a humanoid robotic*, chapitre 18. Springer-Verlag (London).

[Khamassi *et al.*, 2005] Khamassi, M., Lachèze, L., Girard, B., Berthoz, A. et Guillot, A. (2005). Actor-critic models of reinforcement learning in the basal ganglia : From natural to artificial rats. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 13(2):131–148.

[Khamassi *et al.*, 2006] Khamassi, M., Martinet, L.-E. et Guillot, A. (2006). Combining self-organizing maps with mixture of experts : Application to an actor-critic of reinforcement learning in the basal ganglia. *In* Nolfi, S., Baldassare, G., Calabretta, R., Hallam, J., Marocco, D., Meyer, J.-A., Miglino, O. et Parisi, D., éditeurs : *From Animals to Animats : Proceedings of the 9th International Conference on the Simulation of Adaptive Behavior (SAB)*, pages 394–405, Rome, Italy.

[Knierim *et al.*, 1995] Knierim, J. J., Kudrimoti, H. S. et Mcnaughton, B. L. (1995). Place cells, head direction cells, and the learning of landmark stability. *Journal of Neuroscience*, 15:1648–1659.

[Kragic et Christensen, 2002] Kragic, D. et Christensen, H. I. (2002). Survey on visual servoing for manipulation. Rapport technique, Computational vision and active perception laboratory.

[Kuang et Tan, 2000] Kuang, J. et Tan, S. H. (2000). Chaotic attitude motion of satellites under small perturbation torques. *Sound and Vibration*, 235(2):175–200.

[Kuniyoshi, 1994a] Kuniyoshi, Y. (1994a). Learning by watching : extracting reusable task knowledge from visual observation of human performance. *IEEE transactions on robotics and automation*, 10(6):799–822.

[Kuniyoshi, 1994b] Kuniyoshi, Y. (1994b). The science of imitation - towards physically and socially grounded intelligence. Special Issue TR-94001, Real World Computing Project Joint Symposium, Tsukuba-shi, Ibaraki-ken.

[Kuperstein, 1991] Kuperstein, M. (1991). Infant neural controller for adaptive sensory-motor coordination. *Neural Networks*, 4(2):131–145.

[Leighton, 1994] Leighton, R. (1994). Aspirin/migraines. `http://www.elegant-software.com/software/aspirin`.

[Li *et al.*, 2008] Li, Y., Kurata, S., Morita, S., Shimizu, S., Munetaka, D. et Nara, S. (2008). Application of chaotic dynamics in a recurrent neural network to control : hardware implementation into a novel autonomous roving robot. *Biol. Cybern.*, 99(3):185–196.

[Luke *et al.*, 2005] Luke, R. H., Keller, J. M., Skubic, M. et Senger, S. (2005). Acquiring and maintaining abstract landmark chunks for cognitive robot navigation. *In In IEEE/RSJ IROS*, pages 3770–3775.

[Lukosevicius et Jaeger, 2009] Lukosevicius, M. et Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

[Maass *et al.*, 2002] Maass, W., Natschläger, T. et Markram, H. (2002). Real-time computing without stable states : a new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

[Maskara et Noetzel, 1993] Maskara, A. et Noetzel, A. (1993). Forced simple recurrent neural networks and grammatical inference. *In In Proc. the Fifteenth Annual Conference of the Cognitive Science Society*, pages 420–425.

[Mataric, 2000] Mataric, M. (2000). *Imitation in Animals and Artifacts*, chapitre Sensory-Motor Primitives as a Basis for Imitation : Linking Perception to Action and Biology to Robotics. The MIT Press.

[McHaffie *et al.*, 2005] McHaffie, J. G., Stanford, T. R., Stein, B. E., Coizet, V. et Redgrave, P. (2005). Subcortical loops through the basal ganglia. *Trends in Neurosciences*, 28(8):401–407.

[Meltzoff et Decety, 2003] Meltzoff, A. et Decety, J. (2003). What imitation tells us about social cognition : a rapprochement between developmental psychology and cognitive neuroscience. *Philosophical Transactions of the Royal Society B : Biological Sciences*, 358(1431):491–500.

[Meltzoff et Moore, 1977] Meltzoff, A. N. et Moore, M. K. (1977). Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):75–78.

[Meyer et D., 2003] Meyer, J.-A. et D., F. (2003). Map-based navigation in mobile robots - ii. a review of map-learning and path-planning strategies. *Cognitive Systems Research.*, 4(4):283–317.

[Moga, 2001] Moga, S. (2001). *Imiter : une nouvelle voie pour l'apprentissage de robots autonomes*. Thèse de doctorat, Thèse de l'Université Cergy-Pontoise.

[Moga et Gaussier, 1999] Moga, S. et Gaussier, P. (1999). A neuronal structure for learning by imitation. *In* Floreano, D., Nicoud, J.-D. et Mondada, F., éditeurs : *Lecture Notes in Artificial Intelligence - European Conference on Artificial Lif e ECAL99*, pages 314–318, Lausanne.

[Muller *et al.*, 1996] Muller, R. U., James B Ranck, J. et Taube, J. S. (1996). Head direction cells : Properties and functional significance. *Current Opinion in Neurobiology*, 6(2):196–206.

[Münch *et al.*, 1994] Münch, S., Kreuziger, J., Kaiser, M. et Dillmann, R. (1994). Robot programming by demonstration (rpd) - using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. *In Proceedings of the 24th International Symposium on Industrial Robots*, pages 685–693.

[Nadel, 1986] Nadel, J. (1986). *Imitation et communication entre jeunes enfants*. Presse Universitaire de France, Paris.

[Nadel et Potier, 2002] Nadel, J. et Potier, C. (2002). Imitez, imitez, il en restera toujours quelque chose : le statut developpemental de l'imitation dans le cas d'autisme. *ENFANCE PARIS*, 54:76–85.

[Nehaniv et Dautenhahn, 2002] Nehaniv, C. L. et Dautenhahn, K. (2002). The correspondence problem. pages 41–61.

[O'Keefe et Dostrovsky, 1971] O'Keefe, J. et Dostrovsky, J. (1971). The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Res*, 34(1):171–175.

[Pardowitz *et al.*, 2007] Pardowitz, M., Knoop, S., Dillmann, R. et Zollner, R. (2007). Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *SMC-B*, 37(2):322–332.

[Pardowitz, 2007] Pardowitz, M. ; Dillmann, R. (2007). Towards life-long learning in household robots : the piagetian approach. *In 6th IEEE International Conference on Development and Learning, Proceedings*.

[Pearce *et al.*, 1998] Pearce, J. M., Roberts, A. D. L. et Good, M. (1998). Hippocampal lesions disrupt navigation based on cognitive maps but not heading vectors. *IN*, 396:75–77.

[Piaget, 1945] Piaget, J. (1945). *La formalisation du symbole chez l'enfant. Imitation, jeu et rêve. Image et représentation*. Neuchatel ; Paris : Delachaux et Niestlé.

[Piaget, 1970] Piaget, J. (1970). Structuralism. page 153.

[Pollack, 1990] Pollack, J. B. (1990). Recursive distributed representation. *Artificial Intelligence*, 46:77–105.

[Quoy *et al.*, 2001] Quoy, M., Banquet, J.-P. et Daucé, E. (2001). Learning and control with chaos : From biology to robotics. *Behavioral and Brain Sciences*, 24(05):824–825.

[Quoy *et al.*, 2000] Quoy, M., Moga, S., Gaussier, P. et Revel, A. (2000). Parallelization of neural networks using PVM. *Lecture Notes in Computer Science*, 1908:289–296.

[Rizzolatti *et al.*, 1996] Rizzolatti, G., Fadiga, L., Gallese, V. et Fogassi, L. (1996). Premotor cortex and the recognition of motor actions. *Cognitive Brain Research*, 3:131–141.

[Roberts, 1941] Roberts, D. (1941). Imitation and suggestion in animals. bulletin of animal behaviour. 1:11–19.

[Schaal, 1999] SCHAAL, S. (1999). Is imitation learning the route to humanoid robots ? *Trends in cognitive sciences*, 3(6):232–242.

[Schaal *et al.*, 2007] SCHAAL, S., MOHAJERIAN, P. et IJSPEERT, A. (2007). *dynamics systems vs. optimal control - a unifying view*, pages 425–445. Numéro 165.

[Schaal *et al.*, 2001] SCHAAL, S., VIJAYAKUMAR, S., D'SOUZA, A., IJSPEERT, A. et NAKANISHI, J. (2001). real-time statistical learning for robotics and human augmentation. *In international symposium on robotics research*.

[Schöner *et al.*, 1995] SCHÖNER, G., DOSE, M. et ENGELS, C. (1995). Dynamics of behavior : Theory and applications for autonomous robot architectures. *Robotics and Autonomous Systems*, 16(4):213–245.

[Schultz *et al.*, 2000] SCHULTZ, S., PANZERI, S., ROLLS, E. et TREVES, A. (2000). *Quantitative analysis of a Schaffer collateral model*, chapitre 14, pages 257–272. Cambridge University Press.

[Schultz, 1998] SCHULTZ, W. (1998). Predictive reward signal of dopamine neurons. *J Neurophysiol*, 80(1):1–27.

[Servan-Schreiber *et al.*, 1989] SERVAN-SCHREIBER, D., CLEEREMANS, A. et MCCLELLAND, J. L. (1989). Learning sequential structure in simple recurrent networks. pages 643–652.

[Siapas et Wilson, 1998] SIAPAS, A. G. et WILSON, M. A. (1998). Coordinated interactions between hippocampal ripples and cortical spindles during slow-wave sleep. 21(5):1123–1128.

[Simon, 1974] SIMON, A. H. (1974). How big is a chunk ? *Science*, 183(4124):482–488.

[Spence, 1937] SPENCE, K. W. (1937). Experimental studies of learning and the higher mental processes in infra-human primates. *Psychological Bulletin*, 34(10):806–850.

[Thorpe, 1963] THORPE, W. (1963). *Learning and instinct in animals*. Cambridge, MA : Harvard University Press.

[Tritton et Gollub, 1978] TRITTON, D. J. et GOLLUB, J. P. (1978). Physical Fluid Dynamics. *American Journal of Physics*, 46:441–441.

[Tyrrell, 1993] TYRRELL, T. (1993). The use of hierarchies for action selection. *Adapt. Behav.*, 1(4):387–420.

[Vanni-Mercier *et al.*, 2009] VANNI-MERCIER, G., MAUGUIÈRE, F., ISNARD, J. et DREHER, J. (2009). The hippocampus codes the uncertainty of cue-outcome associations : an intracranial electrophysiological study in humans. *Neuroscience*, 29(16):5287–5294.

[Waelbroeck, 1995] WAELBROECK, H. (1995). Deterministic chaos in tropical atmospheric dynamics. *the Atmospheric Sciences*, 52(13):2404–2415.

[Whiten et Ham, 1992] WHITEN, A. et HAM, R. (1992). On the nature and evolution of imitation in the animal kingdom : reappraisal of a century of research. *In* SLATER, P., ROSENBLATT, J., BEER, C. et MILINSKI, M., éditeurs : *Advances in the study of behavior*, pages 239–283, San Diego, CA. Academic Press.

[Widrow et Hoff, 1960] WIDROW, B. et HOFF, M. (1960). Adaptive switching circuits. *In IRE WESCON Convention Record*, volume 4, pages 96–104.

[Zazzo, 1957] ZAZZO, R. (1957). Le problème de l'imitation chez le nouveau-né. *Enfance*, 2:135–142.

[Zell *et al.*, 1993] ZELL, A., MAMIER, G., HÜBNER, R., SCHMALZL, N., SOMMER, T. et VOGT, M. (1993). Snns : An efficient simulator for neural nets. *In MASCOTS '93 : Proceedings of the*

International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems, pages 343–346, San Diego, CA, USA. Society for Computer Simulation International.

Chapitre 9

Annexes

9.1 Annexe A : Les robots

Dans le cadre de mes travaux, j'ai utilisé essentiellement deux robots. Les travaux qui ont porté sur l'émergence d'un comportement d'imitation immédiate, ainsi que ceux sur l'apprentissage de séquences temporelles complexes de gestes ont été appliqués sur un robot Aibo[1]. Les travaux portant sur la navigation ont été appliqués sur un robot mobile Robulab10[2]. Pour permettre l'utilisation de ces robots avec l'outil d'exécution d'architectures neuronales (*Promethe*), les fonctions de contrôle bas niveau ont été intégrées dans la couche d'abstraction matérielle de l'outil. Cette couche d'abstraction matérielle a pour but de permettre le contrôle de n'importe quel matériel robotique sans avoir à modifier les architectures neuronales (figure 9.1).

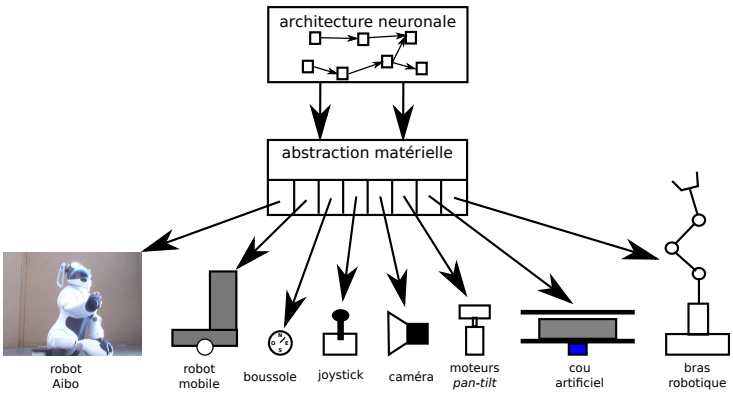


Fig. 9.1: Schémas mettant en évidence le rôle de la couche d'abstraction matériel. Cette couche permet le contrôle de tout type de matériel robotique avec une même architecture neuronale

9.1.1 Sony Aibo/URBI

A mon arrivée, le robot chien Sony Aibo été intégré au simulateur *promethe* avec des fonctions qui lui étaient propre. Ceci impliquait de modifier l'architecture neuronale lors de l'exécution sur d'autre robots. De plus, ces fonctions ne supportaient que difficilement mes travaux pour l'apprentissage de gestes. Pour gérer la communication, un thread (processus léger) a été créé ayant pour objectif de recevoir et traiter les évènements provenant du robot. Ces évènements peuvent être l'image de la caméra, les informations motrices, des valeurs de capteurs, des retours d'erreurs, le niveau de la batterie, etc. Lorsque ce thread reçoit des données (image, informations motrices), alors il les stocke dans des variables ou mémoires appartenant aux structures (dans le sens langage C) correspondant aux différents matériels (moteur, caméra, capteur, etc). Ces structures sont créées à partir d'informations fournies au simulateur par l'intermédiaire de fichiers de configuration "hardware". Il y a alors un fichier de configuration par moteur, capteur, camera. Au niveau de l'architecture neuronale, il existe des groupes de neurones particuliers qui permettent l'accès au matériel :

– *f_joint* permet d'envoyer une commande sur un moteur particulier défini par une chaîne de caractère. Cette fonction récupère la valeur d'un neurone du groupe de neurone précédent (entre 0 et 1) et la transmet à la couche d'abstraction matérielle.

[1]Robot chien de Sony
[2]Plateforme mobile Robosoft

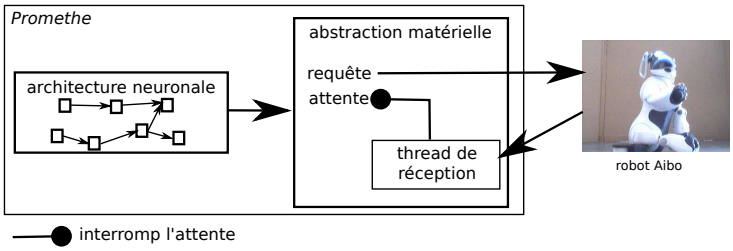


Fig. 9.2: Schémas détaillant le fonctionnement de la communication avec le robot Aibo utilisant le langage URBI. La couche neuronale fait appelle à la couche d'abstraction matérielle pour accéder au robot. La couche matérielle envoie alors une requête au robot puis se met en attente. Lorsque le thread de réception a reçue les données, il interrompt l'attente, débloquant ainsi la suite de l'exécution de l'architecture neuronale.

- *f_speed_joint* permet d'envoyer une commande en vitesse à un moteur particulier défini par une chaîne de caractère. Cette fonction récupère la valeur d'un neurone du groupe de neurone précédent (entre -1 et 1, le signe définissant le sens de rotation) et la transmet à la couche d'abstraction matérielle.
- *f_joint_get_proprio* permet de récupérer la valeur proprioceptive d'un moteur particulier défini par une chaîne de caractère. La valeur récupérée (entre 0 et 1) est alors l'activité du neurone de ce groupe.
- *f_sensor* permet de récupérer la valeur d'un capteur particulier défini par une chaîne de caractère. La valeur récupérée (entre 0 et 1) est alors l'activité du neurone de ce groupe.
- *f_grabimages* permet de récupérer une image d'une caméra particulière définie par une chaîne de caractère.

Toutes ces fonctions font uniquement des appels à la couche d'abstraction matérielle du simulateur. C'est cette couche qui se charge d'envoyer et recevoir les requêtes au matériel même. Dans le cas d'Aibo, la communication asynchrone implique que la couche matérielle envoie la requête puis attende sur un sémaphore (figure 9.2). Lorsque le thread de réception reçoit la donnée attendu, alors il termine l'attente. Ce mécanisme est indispensable pour répondre au fonctionnement synchrone des groupes de neurones tout en s'assurant que l'information attendue soit effectivement disponible.

Dans le cadre du projet européen *Feelix Growing*, nous devions adapter nos algorithmes pour un robot humanoïde Nao[3]. Pour faciliter l'intégration de *Nao* avec le simulateur, le choix a été fait de réutiliser les fonctions développées pour le robot *Aibo*. De la même manière que le robot Aibo, Nao communique avec le simulateur avec le langage *URBI* (en cours de développement à cette période, la caméra avait de faibles performances). De manière à permettre au constructeur d'améliorer la gestion de la caméra de *Nao*, j'ai développé un programme permettant de tester les performances de la caméra du robot avec différents paramètres. Ce programme a alors permis de comparer les performances de la caméra d'*Aibo* avec celle de *Nao*, puis d'améliorer les performances de la caméra de *Nao*. Ce programme a été mis sous licence *GPL* (General Public License) (téléchargeable à l'adresse http ://www-etis.ensea.fr/Equipes/Neuro/telechargement/AiboCam.tar.gz/view).

[3]Robot humanoïde Aldebaran

9.1.2 Le Robulab10 de Robosoft

Au cours de mes travaux, de nouveaux robots ont été acquis par le laboratoire. Ces robots sont un Robuoroc4[4] et trois Robulab10[5], ainsi que trois bras robotique Katana II[6] pouvant être montés sur les Robulab10. Dans le cadre de mes travaux, j'ai essentiellement utilisé une plateforme mobile d'intérieur Robulab10 pour l'apprentissage de tâches de navigation. Ce robot intègre un ordinateur embarqué *Robubox* fonctionnant sous Windows XPe[7]. Cet ordinateur permet de gérer la communication avec les différents matériels composant le robot (moteurs, capteurs ultrason) et avec d'autres ordinateurs via le réseau. Le Robulab10 embarque un programme serveur recevant les requêtes sur le réseau suivant le protocole UDP/IP. Ce programme permet le contrôle de la plateforme mobile par des programmes extérieurs. Lors de la recette de cette nouvelle plateforme, j'ai développé un programme (langage C) sous Linux[8] visant à tester la communication avec l'ordinateur embarqué, le contrôle du robot et la réception de diverses informations comme les capteurs et le niveau de la batterie. Ce programme a ensuite était mis sous licence *GPL* (General Public License) à la demande du fournisseur afin de le redistribuer à ses clients (téléchargeable à l'adresse http ://www-etis.ensea.fr/Equipes/Neuro/telechargement/client_robubox.zip/view). Mais l'ordinateur embarqué dans le *Robulab10* ne permet pas d'exécuter le simulateur. En effet, *Promethe* fonctionne sous Linux alors que l'ordinateur est sous Windows. De plus, la puissance de calcul ne permet pas d'exécuter de grandes architectures neuronales. Il a alors été décidé d'embarquer dans le coffre du robot un nouvel ordinateur permettant l'exécution du simulateur. Le choix devait respecter deux contraintes principales. La première est l'encombrement. En effet, l'espace est limité dans le coffre du robot et doit pouvoir accueillir aussi bien l'ordinateur que la batterie l'alimentant ainsi que le matériel permettant la communication avec des machines de calcul distantes. La seconde contrainte est que ce nouveau matériel ne doit pas créer une charge de travail supplémentaire au niveau du développement du simulateur et des architectures neuronales. Le choix s'est finalement porté sur un ordinateur composé d'une carte mère au format mini ITX (17cmx17cm) *Commell LV679D2C* sur laquelle un processeur *Intel Core2Duo* a été ajouté, deux giga octets de mémoire vive, ainsi que d'une *DOM*(Disk On Module) de quatre giga octets pour stocker les exécutables des outils *Coeos* et *Promethe* et les architecture neuronales. Les avantages de l'utilisation d'une *DOM* sont son très faible encombrement et les performances égales à celles d'un disque dur classique par sa connectique *SATA*.
Cet ordinateur étant embarqué dans le robot, il doit permettre d'envoyer des requêtes au serveur du robot. Le protocole étant de l'UDP, il n'y a donc pas de mécanisme s'assurant de l'intégrité des données transmises sur le réseau ou si celles-ci sont effectivement reçues. Par conséquent, pour éviter des situations indésirables, le choix a été fait de connecter en direct l'ordinateur au robot avec un câble Ethernet full duplex (figure 9.3). Cette connexion directe permet alors d'éviter toute collision de paquets Ethernet qui aurait pu entraîner la perte des données transmises. D'un autre coté, l'ordinateur doit pouvoir communiquer avec d'autres machines de calculs distantes. Cette communication est réalisée par des connexions sans-fil (WiFi) grâce à deux routeurs sans fil *ASUS WL-500g* permettant de créer un pont WiFi. De plus, ces routeurs intègrent chacun un commutateur cinq ports permettant de connecter plusieurs machines sur le même sous réseau. Le robot embarque donc un routeur sans-fil en plus de l'ordinateur. Ces deux composants représentent alors un seul block qui se retrouve dans toutes les plateformes mobiles acquises. Une

[4]Plateforme mobile d'extérieur Robosoft
[5]Plateforme mobile d'intérieur Robosoft
[6]Bras robotique Nueronics
[7]Système d'exploitation embarqué Microsoft
[8]Système d'exploitation Unix

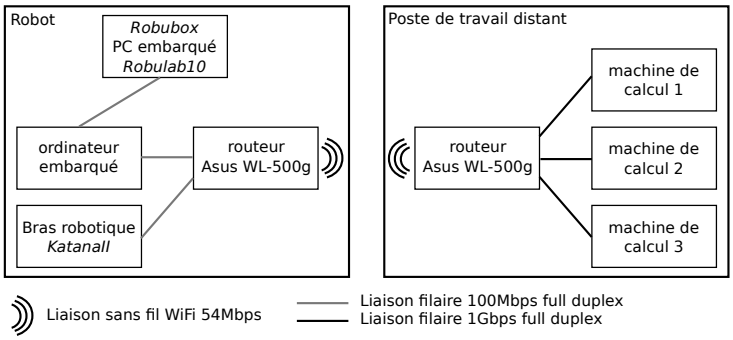


Fig. 9.3: Schémas détaillant le fonctionnement de la communication entre les différents ordinateurs et les matérielles robotiques. Les trois machines de calcul sont connectées avec une bande passante de 1 Gbps à un routeur sans fil via des câbles Ethernet. Les deux routeurs sans fil permettent de faire communiquer les machines de calculs avec le matériels embarqués sur le robot avec une bande passante de 54Mbps. Le robot embarque deux ordinateurs (La *Robubox* contrôlant la plateforme mobile à bas niveau et l'ordinateur ajouté dans le coffre). Ces deux ordinateurs sont connectés directement l'un à l'autre via un câble Ethernet avec une bande passante de 100 Mbps. L'ordinateur ajouté dans le coffre, ainsi que le bras robotique sont tous les deux connectés au routeur sans fil via des câbles Ethernet avec une bande passante de 100 Mbps.

contrainte liée aux problématiques classiques de l'informatique embarquée est l'alimentation. En effet, l'objectif est de permettre à un robot d'évoluer de manière autonome durant plusieurs heures. Le choix des batteries est alors important, car elles doivent être peu encombrantes (tenir dans le coffre du robot avec le reste du matériel) et permettre d'alimenter l'ordinateur et le routeur sans fil pendant plusieurs heures. Le choix s'est finalement porté sur des batteries lithium-ion ayant une puissance de 133 Watts et délivrant 16 volts. Ce type de batterie permet d'alimenter l'ordinateur embarqué ainsi que le routeur sans-fil pendant une durée d'environ trois à quatre heures (cette durée diminue avec le vieillissement des batteries).

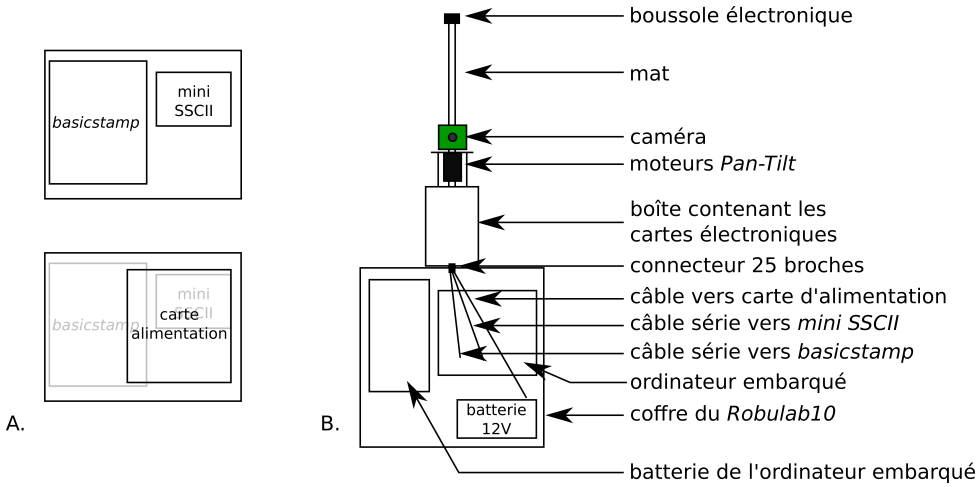


Fig. 9.4: A) Schémas de la boite contenant les cartes électroniques. Dans le fond de la boîte (en haut) sont fixés le basicstamp et la carte mini SSCII. Sur un étage au dessus (en bas) la carte d'alimentation. B) Schémas détaillant le montage de la caméra, des deux moteurs en configuration *Pan-Tilt*, de la boussole électronique, de la boite contenant les cartes électroniques ainsi que du *coffre du robulab10*

Pour permettre le bon fonctionnement des architectures de contrôle pour la réalisation de tâches,

il a fallu ajouter sur le robot une caméra montée sur deux moteurs en configuration *Pan-Tilt*, une boussole électronique et les cartes électroniques permettant la communication entre l'ordinateur et les matériels(figure 9.4). Ce matériel se trouve essentiellement sur le coffre de la plateforme mobile. Les cartes électroniques (un *basicstamp*[9] pour la lecture des valeurs de la boussole, une carte mini *SSCII* pour l'envoi des consignes motrices aux moteurs et une carte d'alimentation permettant d'alimenter ces différents matériels) sont regroupées dans une boite faisant office de cou pour le robot. L'alimentation des cartes électronique et du matériel y étant connecté (moteurs *Pan-Tilt*, boussole électronique) est fournie par une batterie au plomb de 12 volts située dans le coffre. Un connecteur vingt cinq broches permet de relier la batterie à la carte d'alimentation ainsi que l'ordinateur dans le coffre aux cartes électroniques. Ce montage permet de rendre plus facilement amovible l'ensemble du matériel (caméra, boussole, moteurs, cartes électroniques). Les moteurs ainsi que la caméra sont fixés sur la boite contenant les cartes électroniques. La boussole électronique est fixée en haut d'un mat (permettant d'éviter les parasites générés par les moteurs du montage *Pan-Tilt*) lui même fixé entre le coffre et la boite.

[9]Carte programmable Parallax

9.2 Annexe B : Un cou artificiel

Au cours des expériences réalisées dans une tâche de navigation avec le robot mobile, le professeur corrige la trajectoire du robot avec un joystick. J'ai présenté l'utilisation de ce joystick comme si le professeur tenait en laisse le robot et le tirait dans la direction désirée. Mais est-ce réellement le cas ? C'est en partant de cette image entre le joystick et la laisse que j'ai initié la construction d'un cou artificiel pour un robot mobile.

9.2.1 Test préliminaire

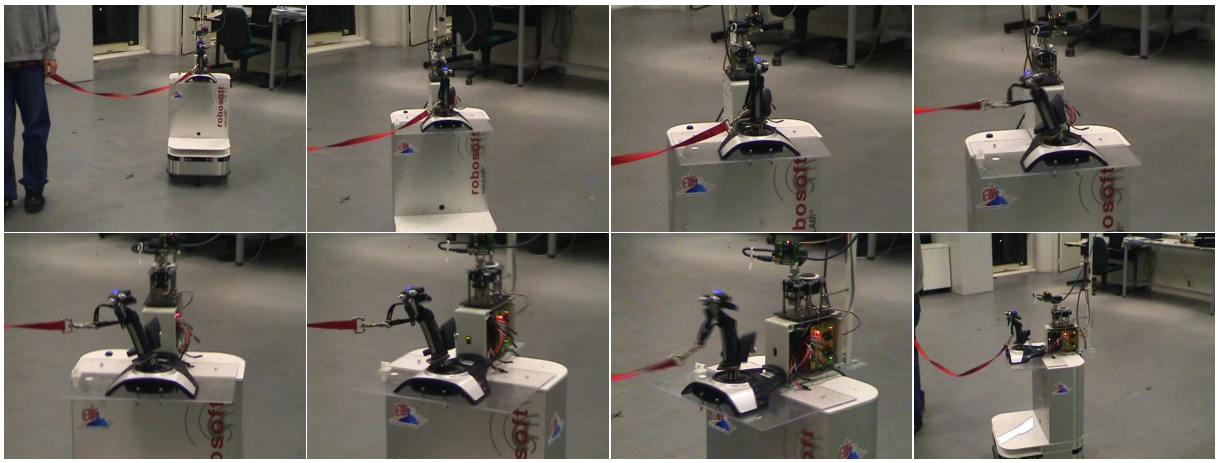


Fig. 9.5: Photo du joystick monté sur le robot mobile. Ce montage permet de se rendre compte du comportement du robot lorsqu'il est tiré avec une laisse par le professeur.

Dans un premier temps, avant de se lancer dans la construction du cou artificiel, un simple joystick a été fixé sur le robot. Une laisse a été attachée autour du manche du joystick. De cette manière, lorsque le professeur tire sur la laisse, le manche du joystick est tiré dans la direction du professeur. En testant ce dispositif, on se rend alors compte que le joystick n'est pas tiré soit à droite soit à gauche comme cela est fait lorsque le joystick est en main, mais il également tiré vers l'avant (le professeur se trouve devant le robot). Cet effet n'a pas de grandes conséquences si ce n'est que le débattement utilisé est finalement plus restreint que le débattement total du joystick. Ceci a impliqué une légère modification logicielle pour tenir compte de ce changement de débattement. Globalement, ce dispositif fonctionne correctement et permet de corriger la trajectoire du robot (Lorsque le professeur tire la laisse dans la direction désirée, le robot va suivre cette direction en tournant). Finalement, lorsque le robot se trouve dans la bonne direction, alors le joystick revient en position centrale.

9.2.2 Cou artificiel

La première version du cou artificiel (figure 9.6) est composée d'un mini joystick, d'un anneau autour duquel la laisse est attachée et d'un ressort permettant de faciliter le retour en position initiale. De manière à ne pas modifier le comportement du robot et des algorithmes utilisés, le déplacement du cou ne doit pas avoir d'impact sur la caméra. En effet, si à chaque correction de la part du professeur la caméra bouge, alors un certain nombre d'images capturées deviendraient

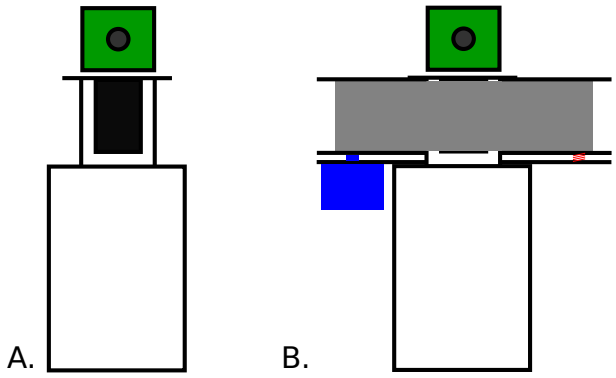


Fig. 9.6: Illustration du montage du cou artificiel sur le robot. A) Partie haute du robot avant l'installation du cou. Elle est composée d'une boite contenant différentes cartes électronique (en blanc), d'un moteur (en noir) qui permet de tourner la caméra (en vert) sur un panorama. B) Partie haute du robot après l'installation du cou artificiel. Le cou se compose d'un anneau (en gris) autour duquel la laisse est attachée, d'un mini joystick (en bleu) et d'un ressort (en rouge à droite) permettant le retour en position initiale.

floues et les angles des points d'intérêt seraient modifiés. Par conséquent, la caméra reste fixée à son support (la boite électronique) et le cou vient s'ajouter autour (figure 9.7).

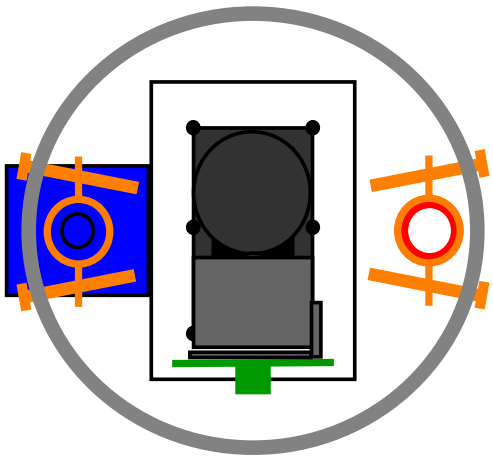


Fig. 9.7: Vue du dessus du montage du cou artificiel. De manière à rendre plus claire le schéma, les supports sur lesquels sont fixés le joystick et le ressort ainsi que ceux sur lesquels reposent l'anneau du cou n'apparaissent pas ici. Du côté gauche de l'anneau, le manche du joystick (en bleu) est entouré par une bague (en orange), celle-ci étant fixé au cou grâce à deux vis. Du côté droit de l'anneau, le ressort (en rouge) est fixé à une bague (en orange) également fixé à l'anneau du cou grâce à deux vis. Au centre de l'anneau, le montage *Pant-Tilt* (en gris foncé) supportant la caméra (en vert). Ce montage est indépendant du cou artificiel, il est directement fixé à la boîte contenant les cartes électronique (grand rectangle blanc)

Cette première version du cou artificiel a été montée sur le robot mobile comme indiqué précédemment (figure 9.8). La laisse était attachée autour du cou du robot de manière à permettre au professeur de tirer le robot dans la direction désirée. Les premiers tests ont essentiellement porté sur les aspects mécaniques du cou. Ils ont mis en évidence que le montage souffrait d'un problème bloquant. En effet, la trop grande proximité du joystick et du ressort avec l'anneau du cou empêche certains mouvements du cou : lorsque l'on tire sur le cou, le joystick ou le ressort font rapidement contact avec l'anneau.

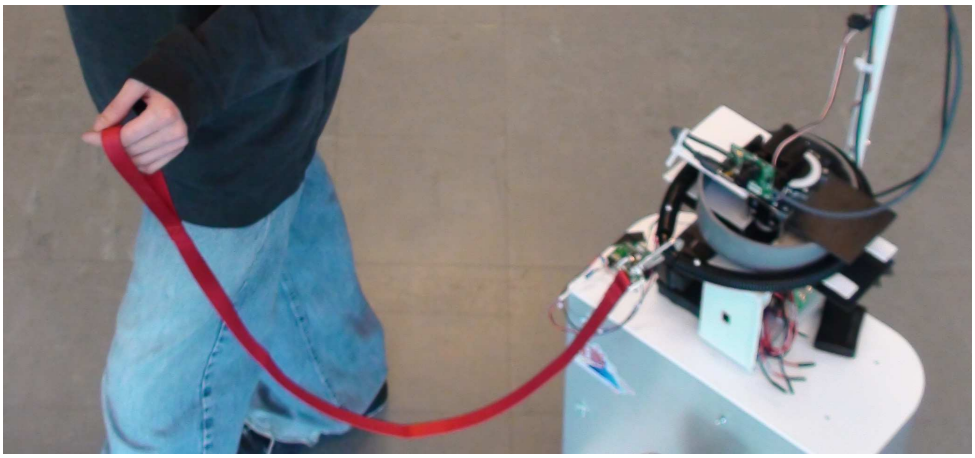


Fig. 9.8: Photo de la première version du cou artificiel monté sur un robot mobile tenu en laisse.

La conception du cou a alors été repensée de manière à ne plus avoir ce problème. Le nouveau montage a particulièrement été centré sur la position du joystick de manière à ce qu'il se trouve au centre de l'anneau du cou. Cette nouvelle contrainte a impliqué de revoir comment le montage *Pan-Tilt* ainsi que la caméra pouvait être fixé sans gêner le comportement des algorithmes qui les exploitaient. De plus, il a également fallu revoir le dispositif permettant le retour du cou en position initiale en retirant le ressort précédemment sur un des côtés de l'anneau du cou. Ces nouvelles contraintes ont été intégrées dans le nouveau cou artificiel (figure 9.9).

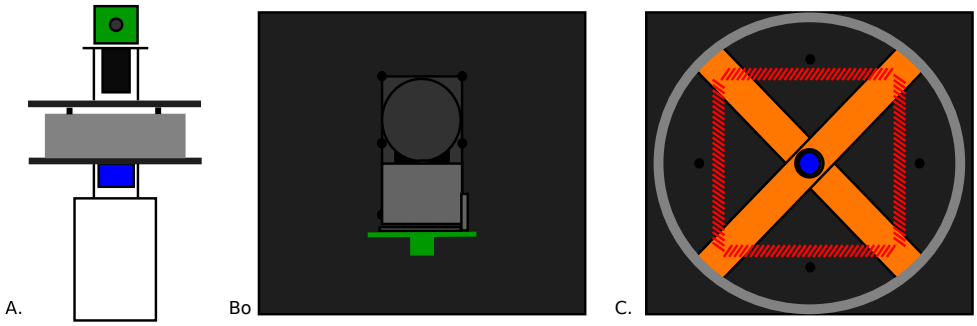


Fig. 9.9: A) Illustration du montage du cou artificiel sur le robot. Le support inférieur du cou est fixé à la boîte contenant les cartes électroniques de la même manière que le montage *Pan-Tilt* l'est avec le support supérieur. De cette manière, le cou est un seul et unique bloc amovible. B) Une fois le cou monté sur le robot, en vue de dessus, seule la caméra ainsi que son support sont visibles. C) Sous le support de la caméra, le montage du cou artificiel. Le manche joystick (en bleu) est guidé par deux lamelles (en orange) qui sont fixées à l'anneau du cou (en gris). De cette manière, lorsque le professeur tire sur le cou, les lamelles entraîne le manche du joystick. Pour revenir en position initiale, quatre ressorts ont été monté "couchés" qui lorsque le cou est tiré, bloquent sur des entretoises métalliques (quatre points noirs).

Avec ce nouveau dispositif, le montage *Pan-Tilt* avec la caméra est fixé à un nouveau support (support supérieur du cou). Ce support n'est pas mobile, il est directement fixé au support inférieur lui même fixé à la boite contenant les cartes électroniques. Entre ces deux supports se trouve le montage du cou artificiel. Le manche du joystick se trouve maintenant au centre du cou permettant ainsi d'éviter tout contact avec l'anneau. Pour entraîner le manche, deux lamelles

ont été perpendiculairement fixées à l'anneau du cou. Chacune des lamelles a un trou en son centre laissant passer le manche du joystick. Pour permettre le retour en position initiale, quatre ressorts en position couchée sont fixés aux extrémités des lamelles. En faisant contact avec les quatre entretoises (fixant les supports supérieur et inférieur), ce mécanisme permet le retour en position initiale. Un des avantages de cette nouvelle version du cou est qu'il est fabriqué en un bloc. Ceci permet de le rendre beaucoup plus facilement amovible, d'autant plus que les attaches avec la boite contenant les cartes électroniques sont identiques à celles de la caméra.
En plus du dispositif avec le joystick, cette version du cou artificiel accueille une série de huit capteurs de pression. Ces capteurs sont disposés autour de l'anneau couvrant ainsi la quasi totalité de la surface extérieure de l'anneau. De manière à permettre une certaine "élasticité", ces capteurs sont placés entre deux mousses de 4mm chacune. Ce nouvel ensemble n'est pas fixé directement à l'anneau du cou, mais il est suffisamment proche de son périmètre pour être serré suffisamment pour ne pas tourner autour de l'anneau. L'ajout de ces capteurs résulte d'une réflexion faite sur les éléments constitutifs d'un cou. En effet, si un joystick permet de savoir dans quelle direction le robot est tiré, il est difficile d'affirmer qu'un mécanisme équivalent est plausible. De manière à permettre de tester comment le robot peut être dirigé, les capteurs ont donc été ajoutés. Le traitement des valeurs retournées par l'ensemble des capteurs est ensuite laissé libre au développeur de l'architecture de contrôle. La carte électronique permettant de recueillir les valeurs des capteurs n'étant pas terminée, le dispositif tactile n'a pu être testé. En ce qui concerne la partie mécanique du cou, un nouveau problème est apparu lors des premiers tests. Lorsque le professeur tire sur la laisse, tout se déroule correctement : le cou est tiré dans la bonne direction et entraîne le joystick correctement. Mais lorsque le professeur tire la laisse en tournant, alors l'anneau du cou tourne également. Cette rotation de l'anneau n'a aucune influence supplémentaire sur le joystick n'est donc pas répercutée sur la direction prise par le joystick. Ce problème n'était pas apparu avec la version précédente du cou, car les positions du joystick et du ressort fixés à l'anneau et diamétralement opposés empêchaient cette rotation. Ce nouveau problème ne permet donc pas d'exploiter correctement le cou artificiel tel quel.

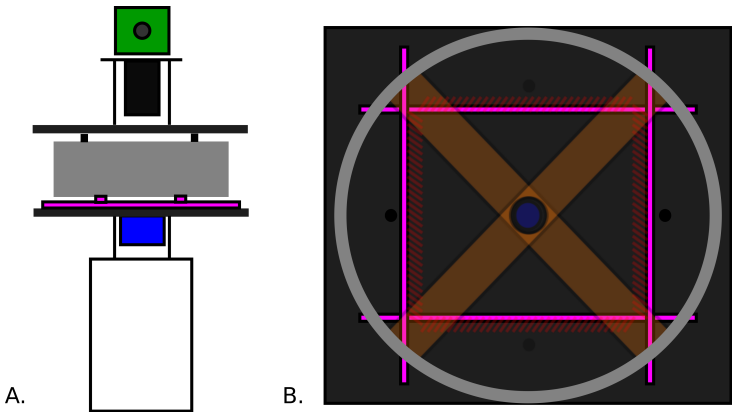


Fig. 9.10: A) Illustration du montage du cou artificiel sur le robot. Le support inférieur accueil un jeu de glissières (en violet) permettant les translations du cou, mais retirant la rotation indésirable. B) Sous le support de la caméra, le montage du cou artificiel. Pour plus de clarté, les éléments constituant le précédent montage ont été rendu moins opaques. On observe alors mieux le montage de l'anneau du cou sur les deux jeux de glissières (en violet).

Il est apparu que ce nouveau problème n'était pas simple à régler. En effet, la majorité des solutions pouvant résoudre ce problème implique de bloquer ou fortement limiter la mobilité de

l'anneau. Ce qui n'est pas acceptable, car tout l'intérêt du cou artificiel est perdu. Néanmoins, une solution semble viable. Elle consiste à fixer l'anneau sur un jeu de glissières (figure 9.10). L'anneau serait alors fixé sur un premier jeu de deux glissières. Ces glissières seraient alors elles même fixées sur deux autres glissières positionnées à la perpendiculaire des premières. Ce nouveau dispositif permettrait alors de permettre tous les mouvements en translation du cou et de supprimer toutes rotations de celui-ci. Ce montage reste aujourd'hui à réaliser, mais il ne remettrait nullement en cause le dispositif actuel. Les glissières constitueraient un "étage" supplémentaire du dispositif global sur le support inférieur du cou.