THÈSE de DOCTORAT de L'UNIVERSITÉ PARIS 6

Spécialité : **Informatique**

Présentée par : **Stéphane Doncieux**

pour obtenir le grade de DOCTEUR de L'UNIVERSITÉ PARIS 6

Évolution de contrôleurs neuronaux pour animats volants : méthodologie et applications

Soutenue le 23 juin 2003

devant le jury composé de :

Jean-Marc Alliot	(rapporteur)
Yves Brière	(examinateur)
Patrick Gallinari	(examinateur)
Patrick Hénaff	(examinateur)
Auke Jan Ijspeert	(rapporteur)
Jean-Yves Jaffray	(examinateur)
Jean-Arcady Meyer	(directeur de thèse)

Résumé

Dans cette thèse, nous nous intéressons à la conception automatique de contrôleurs de robots. Nous utilisons pour cela des réseaux de neurones, avec des Algorithmes Évolutionnistes pour méthode de conception et d'optimisation. ModNet, le codage que nous définissons dans cette thèse, est un codage indirect modulaire, dans lequel le réseau de neurones est un assemblage de modules de structures prédéfinies ou variables. L'objectif de ce codage est de générer des structures de complexité quelconque pour résoudre des problèmes de difficulté croissante.

Le pendule inversé nous sert de premier banc de test. Généralement, l'objectif à résoudre est de l'empêcher de tomber pendant une durée déterminée. Nous allons plus loin et nous nous intéressons à la stabilisation complète : nos contrôleurs doivent centrer le chariot et ramener le pendule à la verticale. Ce problème se résolvant facilement si on dispose des dérivées de la position et de l'angle que fait le pendule, nous ne donnons pas ces informations pour vérifier si notre codage est capable de générer des réseaux intégrant ces capacités.

Nous testons ensuite notre algorithme sur des systèmes instables ayant quatre à cinq degrés de liberté. Nous avons choisi pour cela des robots volants, qui nécessitent un contrôle à la fois robuste et efficace.

Nous commençons avec un dirigeable lenticulaire, plateforme disposant de sept moteurs et de cinq capteurs. Nous faisons évoluer des contrôleurs capables de maintenir l'assiette et l'altitude de cet appareil, puis nous faisons évoluer d'autres contrôleurs pour le maintien complet de l'aéronef, l'objectif étant alors de maintenir le dirigeable à la verticale d'un objectif donné.

Nous nous intéressons ensuite à un hélicoptère, dont nous cherchons d'abord à contrôler l'assiette, puis l'assiette et la position, avant de chercher à maintenir tous ses degrés de liberté.

Jamais des problèmes de ce niveau de complexité n'avaient été traités par évolution de réseaux de neurones. De plus, les ingénieurs ont coutume de traiter ce problème de façon incrémentale à partir d'une décomposition comportementale. Nous proposons des solutions alternatives à cette approche qui nous permettent de résoudre le problème complet en une fois, sans avoir à le décomposer.

Mots-clés: évolution, réseaux de neurones, modules, dirigeable, hélicoptère

Abstract

In this thesis, we address the problem of the automatic conception of robot controllers. To this end, we generate neural networks with evolutionary algorithms as design and optimisation techniques. ModNet, the encoding we present here, is an indirect modular encoding, that generates neural networks made of modules of predefined or changing structure. The goal of this encoding is to produce any kind of structure to solve problems of increasing complexity.

The cartpole is our first benchmark. Generally, the aim is to avoid its fall during a given period of time. We go further and try to stabilize it : our controllers have to center the cart while keeping the pole vertical. This problem being easily solved if the derivatives of the position and of the angle are known, we do not explicitely provide this information to check if our encoding is able to generate networks that discover its utility.

We then test our algorithm on unstable systems that have four to five degrees of freedom. To this end, we chose flying robots, which require both an efficient and robust control.

We start with a lenticular blimp, that has seven motors and five sensors. We evolve controllers able to keep both trim and altitude ; then we evolve other controllers able to keep all of its degrees of freedom, the goal being to keep the blimp above a given target at a given altitude.

We go on with an helicopter, controlling first its trim, then its trim and position, and finally all its degrees of freedom.

Such problems have never been tackled using neural networks and evolution. Usually, they are delt with by engineers through a decomposition into smaller and simpler problems, in an incremental way. We propose an alternative solution to this approach that allows us to solve the whole problem at once.

Keywords: evolution, neural networks, modules, blimp, helicopter

Remerciements

Je tiens tout d'abord à remercier Jean-Marc Alliot et Auke Jan Ijspeert, mes rapporteurs, d'avoir accepté de travailler sur ma thèse. Merci à Patrick Gallinari et à Jean-Yves Jaffray d'avoir accepté de faire partie de mon jury. Merci également à Yves Brière et à Patrick Hénaff d'avoir accepté mon invitation.

Je remercie sincèrement Jean-Arcady du soutien et des conseils qu'il a su m'apporter tout au long de ma thèse. Je le remercie de m'avoir accordé sa confiance sans défaillir pendant ces quatre années.

L'AnimatLab est une équipe très chaleureuse et très soudée à laquelle c'est un vrai plaisir d'appartenir, tant au plan scientifique que personnel. Tout le monde est toujours prêt à rendre service et si les conversations scientifiques sont fréquentes, les moments de détente le sont tout autant, ce qui permet de travailler dans de bonnes conditions. Merci donc à Agnès, JAM, Olivier, Vincent, Gildas, Pierre, Benoit, GRobert, Fabien, Gouri. Merci aussi à David, qui n'est plus physiquement au labo mais que j'ai la chance de cotoyer encore très régulièrement.

La conception et la réalisation de SFERES m'ont appris énormément et m'ont fait prendre conscience qu'avant je ne connaissais vraiment rien en informatique, contrairement à ce que je pensais. Je ne peux que remercier Samuel Landau d'avoir éclairé ma lanterne sur ce point. Les longs mois passés à réfléchir sur ce logiciel avec lui furent agréables et fructueux, de même que les discussions que nous eurent régulièrement par la suite. J'espère que nous pourrons mettre en place un projet à plus long terme autour des idées que nous avons évoquées lors de ces discussions.

Merci à mes parents, Maurice et Colette, de m'avoir toujours aidé et soutenu en toutes circonstances, tout au long de mon existence. Ils ont toujours été là lorsque j'avais besoin d'eux et m'ont toujours aidé.

Merci enfin à Christine, ma compagne, qui a su me soutenir et me faire rire jusque dans les moments les plus difficiles.

Table des matières

Introduction

Pendant longtemps, la robotique a consisté en l'étude d'automatismes en environnements clos et totalement connus. Les meilleurs exemples en sont les robots industriels qui sont aujourd'hui capables d'accomplir la plupart des tâches d'une chaîne de production avec une précision impressionnante sans la moindre intervention humaine. Ces robots savent découper ou plier des tôles d'acier, placer et souder des composants électroniques, peindre, etc. Toute tâche à caractère répétitif est à leur portée. Cependant, ce ne sont pas les robots qui sont adaptés à leur environnement, mais l'environnement qui est adapté aux robots. Les ateliers sont conçus de façon à ce que l'enchaînement des actions soit toujours le même et la disposition des objets à manipuler reste très précise. Ces préparatifs répondent à des besoins d'efficacité mais sont aussi nécessaires au bon fonctionnement de ces appareils. En effet, si une pièce arrive dans une mauvaise position ou au mauvais moment, le blocage de la machine est très probable et l'intervention humaine indispensable pour relancer le processus de fabrication. La tâche à résoudre pour les ingénieurs est alors de comprendre comment cet incident a pu se produire et de faire en sorte qu'il ne se reproduise plus en adaptant l'environnement de façon adéquate.

Les raisons de telles contraintes sont d'ordre pratique. Il est plus simple de considérer le robot dans un environnement favorable, adapté à son fonctionnement : un environnement idéal dans lequel tout est conçu pour rendre la tâche la plus simple possible. Cela permet de simplifier la mécanique nécessaire et donc de réduire la complexité du système de contrôle, les besoins de réglage, les risques de panne et, du coup, les coûts de production.

Cependant, si ces méthodes conviennent parfaitement aux ateliers de production des entreprises, de nombreuses autres applications potentielles de la robotique, telle que la robotique mobile, ne sauraient se satisfaire de contraintes aussi drastiques sur les conditions de fonctionnement.

Les robots mobiles, par opposition aux robots manufacturiers généralement statiques de l'industrie, sont susceptibles de se déplacer, ce qui accroît leur champ d'action et rend plus difficile le contrôle de leur environnement. Que leur champ d'action soit une centrale nucléaire après une catastrophe, un fond marin ou une planète lointaine, il devient évident qu'il n'est plus possible de contrôler l'environnement du robot et, dans certains cas, cet environnement est même inaccessible et complètement inconnu. Dans ces conditions, les efforts ne doivent plus se concentrer sur la préparation de l'environnement immédiat du robot, mais plutôt sur la robustesse du comportement du robot, voire sur l'ajout de capacités permettant au robot de s'adapter par lui-même à son environnement. Certains robots sont ainsi capables à présent de construire automatiquement et progressivement une carte de leur environnement. Ils peuvent ensuite se localiser de manière robuste à l'intérieur de cette carte et rejoindre un objectif donné en empruntant la trajectoire la plus courte (Filliat et Meyer, 2003; Meyer et Filliat, 2003). Les techniques adaptatives correspondantes reposent sur un principe radicalement différent des techniques des ingénieurs. Ces dernières reposent en effet sur une stratégie en trois étapes : analyse, modélisation, conception. En cas d'erreur ou de mauvais fonctionnement, on recommence pour affiner ou compléter le modèle avant de générer un nouveau contrôleur. Les techniques adaptatives, au contraire, s'appuyent en priorité sur une exploration directe par le robot de ses propres capacités, et non sur une analyse extérieure exhaustive.

L'approche systématique des ingénieurs donne ainsi des résultats intéressants, mais pas toujours des plus simples ni des plus efficaces. La raison en est que les outils et les méthodes dont ils disposent ne permettent pas de prendre en compte tous les aspects du problème. Une illustration simple de cette difficulté réside dans l'exploitation de la morphologie et, en particulier, de la conformation des capteurs. Ainsi, Maris et te Boekhorst ont étudié le problème du regroupement d'objets dans une arène (Maris et te Boekhorst, 1996). Plusieurs robots à roues se déplacent dans une arène où sont disposés aléatoirement des cubes qu'il faut regrouper en tas. Le robot ne dispose pour cela que de deux capteurs de distance lui indiquant la présence d'un obstacle devant lui. Une manière intuitive pour un humain de résoudre le problème serait de le décomposer en tâches élémentaires : explorer l'arène à la recherche d'un cube, pousser le cube jusqu'à atteindre un tas, puis laisser le cube et recommencer l'exploration. Cette tâche nécessiterait un contrôleur doté d'une mémoire, pour se rappeler dans quelle étape il se situe, et disposant d'au moins trois comportements élémentaires. Or Maris et te Boekhorst ont montré que cette tâche peut être résolue en utilisant des commandes moteurs directement proportionnelles à ce que renvoient les capteurs de proximité. Cette architecture, dont l'intérêt a été mis en évidence par Braitenberg (Braitenberg, 1984), est la plus simple qu'il soit possible d'imaginer. Pour générer le comportement désiré, il suffit de positionner les capteurs de manière adaptée. Si on les dispose à l'avant du robot et en les écartant suffisamment pour qu'un cube puisse passer entre les deux, le robot ne détectera pas la présence des cubes qu'il va rencontrer pendant son exploration. Cela signifie qu'il va avancer en poussant un cube rencontré sans le détecter. Dès qu'il va rencontrer un autre cube sur le côté ou un mur, il va tourner, cessant ainsi de pousser le cube qui était devant lui. Progressivement, ce comportement très simple va permettre de regrouper les cubes en tas de plus en plus gros. La tâche sera ainsi résolue avec une architecture des plus simples, dépourvue de mémoire et dotée d'un seul comportement en interne. Pourtant, d'un point de vue extérieur, les trois comportements cités précédemment seront présents. Cependant, ils ne seront pas implémentés directement dans le contrôleur du robot, mais émergeront de l'interaction entre le robot et son environnement.

Une autre limitation importante des outils des ingénieurs repose sur les hypothèses simplificatrices qui sont réalisées lors de la résolution d'un problème. Des phénomènes physiques qui pourraient être exploités, sont ainsi négligés et considérés comme une source de perturbation qui va réduire l'efficacité du système conçu, plutôt que l'améliorer. Des méthodes de conception adaptatives, basées sur l'exploration des interactions du système avec son environnement n'auront pas ces limitations. Elles pourront ainsi exploiter toutes les particularités du système qui peuvent leur être utiles. Thompson a utilisé des algorithmes évolutionnistes pour obtenir des configurations de FPGA[1] permettant de réaliser des fonctions prédéfinies (Thompson *et al.*, 1996). Il a trouvé des solutions dans lesquelles, à côté du circuit réalisant le calcul, se trouvait un autre circuit, non relié aux sorties du processeur et donc a priori inutile. Or, en supprimant ce circuit, le FPGA ne réalisait plus la fonction. Cela signifie que ce circuit, même s'il n'était pas relié aux sorties, avait une influence importante sur le fonctionnement du processeur et, de fait, ce sytème exploitait les interférences électro-magnétiques entre les pistes du circuit intégré. Jamais un concepteur humain n'aurait utilisé de telles propriétés, qui sont considérées comme du bruit et que l'on cherche par tous les moyens à supprimer. L'algorithme d'adaptation automatique a constaté que ce phénomène pouvait être utile plutôt que nuisible et l'a utilisé pour générer une solution efficace.

Ces exemples montrent l'intérêt des approches "proximales" par rapport aux approches "distales" utilisées en ingénierie. Au lieu de considérer le problème de notre point de vue, qui est distant et biaisé par les connaissances que l'on a sur le système, ainsi que par les méthodes de résolution dont on dispose, il est intéressant de le traiter d'un point de vue local, du point de vue du robot. En effet, le robot va explorer par lui-même les différentes possibilités et sera susceptible d'exploiter des caractéristiques particulières de l'environnement ou de sa dynamique auxquelles un ingénieur n'aurait pas pensé. La biologie est une importante source d'inspiration pour de telles techniques, car les êtres vivants ont résolu beaucoup des problèmes que nous nous posons actuellement (Meyer et Guillot, 2002; Guillot et Meyer, 2003), qu'il s'agisse de sélection de l'action (Girard *et al.*, 2002), de navigation (Filliat et Meyer, 2003; Gourichon *et al.*, 2003; Meyer et Filliat, 2003) ou encore de contrôle moteur (Meyer, 1997b; Meyer, 1997a; Meyer *et al.*, 2002).

Nous nous sommes intéressé à cette dernière catégorie de problèmes, dans le but de trouver comment agir sur les effecteurs dont on dispose, à partir des informations issues de capteurs, pour réaliser un comportement donné. Nous avons centré notre travail sur des systèmes instables nécessitant un contrôle constant, robuste et efficace. Dans la mesure où ces systèmes ne sont pas dotés de variables internes, les valeurs à envoyer aux moteurs dépendent de leurs états au pas de temps présent ainsi que, parfois, des états aux pas de temps précédents. Notre problématique consiste donc à trouver la fonction $f(.)$ permettant de réaliser au mieux le comportement souhaité :

$$o(t) = f(x(t), x(t-1), ...)$$

où o est le vecteur des commandes, $x(t)$ est l'état du système au pas de temps t.

La technique que nous avons choisie pour représenter $f(.)$ est celle des réseaux de neurones.

[1]Field Programmable Gate Array

Ces réseaux ont l'avantage de pouvoir générer aussi bien des comportements réactifs que des comportements à dynamique plus riche. De plus, contrairement à la plupart des autres types de contrôleurs, leur formalisme permet de gérer directement des systèmes ayant plusieurs entrées et plusieurs sorties.

L'algorithme d'adaptation que nous avons choisi est un algorithme évolutionniste, ce type d'algorithme permettant de manipuler des réseaux de neurones de structure quelconque pour résoudre des problèmes très variés (Meyer *et al.*, 2002). Ces algorithmes étant non supervisés, ils ne nécessitent pas de connaître une solution idéale au problème.

Pour étayer cette problématique, nous avons choisi trois plateformes de complexité croissante : un pendule inversé, un dirigeable de forme lenticulaire et un hélicoptère. Le pendule inversé est un banc de test très utilisé pour évaluer des algorithmes de contrôle (Wieland, 1991). Les autres plateformes sont des robots volants. Nous avons fait ce choix car il correspond à des systèmes réels et non plus à des cas d'école. Comme le pendule inversé, ce sont des plateformes qui nécessitent un contrôle permanent. De plus, ils disposent de beaucoup plus de degrés de liberté, ce qui permettra de tester l'aptitude de nos algorithmes à passer à l'échelle.

Plan du mémoire

Dans le premier chapitre, nous présenterons les problèmes liés aux robots volants ainsi que les différents travaux réalisés sur ce type de plateforme.

Dans le second, nous évoquerons les différents types de contrôleurs utilisés dans des approches adaptatives ainsi que les méthodes de conception automatiques les plus couramment employées.

Dans le troisième, nous reviendrons en détail sur les algorithmes évolutionnistes qui seront utilisés dans la suite de ce mémoire.

Le quatrième chapitre sera consacré à l'utilisation des algorithmes évolutionnistes pour la conception de réseaux de neurones.

ModNet, le codage que nous avons développé, sera présenté dans le cinquième chapitre et le sixième présentera SFERES[2], la plateforme d'évolution et de simulation que nous avons développée avec Samuel Landau.

Les résultats obtenus sur le pendule inversé seront détaillés au septième chapitre. Les résultats des expériences portant sur le dirigeable lenticulaire seront présentés au huitième et ceux sur l'hélicoptère le seront au neuvième.

Enfin, nous discuterons de la portée et des limitations de nos résultats avant de conclure ce document.

[2]SFERES is a Framework for Encouraging Research on Evolution and Simulation

Chapitre 1

Robots volants autonomes

Les robots volants (UAV, pour Unmanned Aerial Vehicle, en anglais) jouent un rôle croissant dans le domaine économique et militaire. Leurs utilisations dans les conflits en Irak et au Kosovo ont montré leur utilité pour la surveillance et l'acquisition d'informations sur le terrain. Cependant, leur intérêt est loin d'être confiné aux applications militaires. La science, la télévision, le cinéma, l'agriculture et les organismes de secours peuvent bénéficier de telles plateformes.

La manœuvrabilité et la taille, potentiellement très réduite, des UAV permettent de faire des prises de vues sur des zones difficiles à survoler avec un hélicoptère conventionnel. Les robots volants sont ainsi susceptibles de recueillir des informations sur des dégagements gazeux au dessus d'un volcan (Guccione *et al.*, 2000), de récupérer des données sur un cyclone sans mettre en danger la vie d'un pilote (Holland, 2002) ou peuvent plus généralement faire de la photographie aérienne. L'autonomie de certaines plateformes autorise la conception de systèmes de surveillance de l'atmosphère à haute altitude (Keidel, 2000; Romeo *et al.*, 2000; Stephens *et al.*, 2000). Leur précision et leur prix de revient comparés à des plateformes pilotées leur permettent d'être utilisés en agriculture pour épandre des engrais ou des insecticides de façon économique et efficace (de tels systèmes sont déjà commercialisés au Japon par la société Yamaha) ou suivre l'évolution de plantations (Herwitz, 2000). Les robots volants peuvent être utilisés par des secouristes pour détecter plus rapidement les victimes d'une catastrophe naturelle ou industrielle et guider les recherches. Ils peuvent repérer rapidement les lieux et informer sur les zones potentiellement dangereuses à éviter (projet RESSAC de l'ONERA et Concours International de Robotique Aérienne section 1.4). Leur utilisation en groupe permet d'assurer la couverture d'une zone géographique, de remédier aux pannes et d'assurer ainsi une grande fiabilité (Wu *et al.*, 1999).

De nombreuses autres applications sont envisageables. Les robots volants ne doivent cependant pas être considérés comme de simples substituts des aéronefs pilotés. Ils sont destinés à des opérations dangereuses ou nécessitant des véhicules très manoeuvrables et de petite taille ; ils peuvent aussi servir à des missions de plus longue durée nécessitant un vol de plusieurs semaines ou qui sont économiquement non envisageables avec des appareils pilotés.

L'éventail très large des applications possibles, ainsi que l'intérêt que peuvent avoir ces plateformes, ont provoqué un développement très rapide des recherches dans ce domaine. Stimulées par la création de plusieurs concours universitaires, les recherches portant sur ces appareils se sont multipliées en quelques années et de nombreuses universités disposent aujourd'hui d'équipes dédiées à l'étude de robots aériens. Nous allons présenter dans ce chapitre les problématiques liées à ce type de plateformes ainsi qu'une revue rapide des travaux réalisés dans ce domaine.

1.1 Problèmatiques associées

Le contrôle de robots aériens impose des besoins nouveaux et des contraintes originales par rapport au contrôle de robots terrestres. Cela tient principalement à l'espace dans lequel évoluent ces appareils.

Les robots aériens sont en effet susceptibles de se mouvoir dans un espace à trois dimensions contrairement aux robots mobiles terrestres, limités au plan plus ou moins horizontal du sol. Leurs possibilités de mouvement sont donc fortement accrues, ce qui nécessite un plus *grand nombre d'effecteurs*. Un robot terrestre de laboratoire ne dispose que de trois degrés de liberté - deux coordonnées de position spatiale auxquelles s'ajoute le cap. Un robot aérien dispose de six degrés de liberté - trois coordonnées spatiales et trois coordonnées angulaires - ce qui impose un plus *grand nombre de capteurs* pour pouvoir se situer dans l'environnement.

Les robots aériens sont en *équilibre instable* de façon permanente au sein de la masse d'air environnante, alors que les robots terrestres reposent sur leurs roues ou leurs pattes. Un robot terrestre peut ainsi s'arrêter ou même tomber en panne sans conséquences graves a priori. Si un robot aérien tombe en panne, il est à craindre qu'il soit entièrement détruit dans les secondes qui suivent, suite à un atterrissage brutal. La conception de robots aériens entraîne donc directement, dès les premières étapes, un souci de *fiabilité*.

La masse d'air dans laquelle évoluent les robots aériens suit, de plus, une *dynamique complexe, non déterministe*. Elle peut soumettre les robots à des forces importantes et nécessite donc que les contrôleurs mis en place soient très *robustes*.

De plus, l'aérodynamique impose des contraintes sur l'espace des variables d'état en dehors desquelles le robot ne suit plus un fonctionnement normal (décrochage[3] par exemple). Ces contraintes définissent le *domaine de vol* de l'aéronef, c'est-à-dire l'espace des variables d'état en dehors desquelles l'aéronef n'est plus contrôlable.

Ces systèmes consituent donc une étape importante vers des problèmes plus complexes. Le pendule inversé, qui nous sert de banc de test pour nos algorithmes de contrôle, est un cas intéressant. Cependant, s'il teste les capacités à contrôler un système instable, il ne permet pas d'évaluer les capacités de passage à l'échelle. Les robots volants constituent donc une étape supplémentaire dans cette voie. Leur instabilité, le nombre d'entrées et de sorties, ainsi que d'autres propriétés, liées à la dynamique de ces appareils (inertie, couplages entre degrés de liberté), en font des problèmes de test intéressants pour vérifier les capacités d'algorithmes de contrôle sur des problèmes plus complexes que ceux que traite habituellement la robotique autonome.

[3]Le décrochage intervient lorsque la portance devient trop faible à cause d'une vitesse trop petite ou de l'effet de turbulences.

1.2 Types et caractéristiques des différentes plateformes aériennes

1.2.1 Hélicoptère

Les hélicoptères sont des appareils à décollage et atterrissage vertical (VTOL[4]). Ils offrent l'avantage d'être très maniables, nécessitent peu d'espace pour évoluer et peuvent faire du vol stationnaire, ce qui les rend très intéressants pour des applications de surveillance ou de prise de vue aérienne.

Ils disposent cependant d'une autonomie généralement faible et d'un rayon d'action limité. De plus, ce sont des aéronefs instables nécessitant un contrôle constant. Cette caractéristique les rend peu appropriés pour inspecter des sites à hauts risques, que ce soient des usines chimiques ou des zones très peuplées. Ils sont actuellement très utilisés pour des applications nécessitant une courte portée, comme la prise de vue aérienne ou l'agriculture.

Des hélicoptères pendulaires, plus stables, sont en cours de développement (plateforme Hetel des sociétés Isnav et ECT). Le principe de ce type de plateforme est de superposer deux rotors tournant en sens inverse. Cela supprime la réaction due à la rotation du rotor principal dans les hélicoptères conventionnels et supprime ainsi la nécessité d'un rotor de queue, dont l'objectif premier est de compenser cet effet.

1.2.2 Dirigeable

Les recherches portant sur le contrôle de dirigeables se sont multipliées au cours des dernières années (Elfes *et al.*, 1990; Gomes et Ramos, 1998). Les dirigeables sont des aéronefs stables, qui sont en équilibre statique dans la masse d'air environnante, contrairement aux hélicoptères, car leur enveloppe, remplie d'un gaz plus léger que l'air, les maintient à une altitude constante ou presque[5]. Le gaz utilisé est généralement de l'hélium, gaz inerte et donc sans danger comparé à l'hydrogène qui a provoqué le spectaculaire incendie de l'Hindenburg. $1m^3$ d'hélium permet de soulever $1kg$ au niveau de la mer ($1m^3$ d'hydrogène soulève $1,2kg$) grâce à la poussée d'Archimède, ce qui explique le volume important de ce type d'aéronef.

Leurs avantages sur les autres plateformes volantes sont nombreux. Comme les hélicoptères, ils peuvent voler en stationnaire. Étant donné que le maintien en l'air ne nécessite pas de dépense d'énergie, l'autonomie des dirigeables est considérablement plus élevée que celle d'un hélicoptère ou d'un avion, si le vent n'est pas trop élevé. Leur stabilité en fait des appareils utilisables dans des contextes nécessitant une sécurité renforcée comme le survol de sites à risques. En principe, ils sont simples à piloter (au moins en ce qui concerne les maquettes de taille réduite) et bien moins dangereux à utiliser que des hélicoptères.

Leurs inconvénients résultent principalement de leur volume. Ce dernier constitue un véritable handicap en cas de perturbations météorologiques, en conditions de vent fort en particulier.

[4] Vertical Take-Off and Landing.

[5] Les dirigeables réels sont généralement légèrement plus lourds que l'air de façon à pouvoir regagner le sol facilement. Ils compensent le poids restant par la portance créée par leur vitesse relative à la masse d'air.

De plus le rapport $1m^3/kg$ entraine rapidement des tailles importantes dès que l'on souhaite embarquer des capteurs ou de la puissance de calcul. Plus le volume est gros, plus il faut des moteurs puissants pour pouvoir contrer le vent et plus il faut un volume grand pour porter la surcharge occasionnée par les moteurs et leur source d'énergie...

Leurs principales applications à l'heure actuelle concernent la prise de vue aérienne et les vols stratosphériques (relais de communication ou plateforme de surveillance). Ils sont souvent pilotés à partir d'un système de vision (Coelho *et al.*, 1998; van der Zwaan *et al.*, 2000).

1.2.3 Avions

Les avions se maintiennent en l'air grâce à la portance créée sur leurs ailes par la vitesse relative à la masse d'air[6]. Ils sont moins manœuvrables que les hélicoptères, mais sont aussi plus stables. Ils disposent d'une plus grande autonomie et d'un rayon d'action plus important que celui des hélicoptères et dirigeables. Ils sont également utilisables dans des conditions atmosphériques plus variées (vent fort, altitudes élevées, températures élévées).

Dans l'impossibilité physique de rester en vol stationnaire, ils sont plutôt destinés à des missions à moyenne ou longue portée, contrairement aux hélicoptères et aux dirigeables. Ils peuvent nécessiter, de plus, une piste de décollage et d'atterrissage.

1.2.4 Ailes battantes

Les aéronefs à ailes battantes cumulent potentiellement les caractéristiques des hélicoptères et des avions. En observant les oiseaux, on remarque rapidement qu'ils disposent de capacités encore hors d'atteinte pour des aéronefs artificiels : ils sont capables de vol stationnaire ou quasi stationnaire, ainsi que de vol en translation à vitesse importante, tout en consommant peu d'énergie. Leur maniabilité les situe au niveau des hélicoptères pour le vol stationnaire et des avions (modèles réduits) pour le vol en translation.

Peu de plateformes existent à l'heure actuelle, mais de nombreux projets sont en développement (Michelson et Reece, 1998; Yan *et al.*, 2001). Des aéromodélistes amateurs ont développé des planeurs à ailes battantes (présenté aux Journées MicroDrones, Toulouse 18-20 septembre 2002). Plusieurs plateformes académiques sont en cours de développement (Augustsson *et al.*, 2002), mais aucune encore n'est autonome énergétiquement. Le projet REMANTA destiné à préparer les outils nécessaires au développement de ce type de plateforme est en cours d'étude à l'ONERA (Boyer, 2002).

1.3 Revue des applications existantes

Dans la plupart des utilisations actuelles, le drone n'est pas autonome : il est piloté depuis le sol par un expert humain. Nous allons nous intéresser aux approches cherchant à lui conférer

[6] C'est le même principe que pour un hélicoptère sauf que, dans le cas de l'hélicoptère, l'aile, ou plutôt la pale, bouge et l'air est immobile alors que, dans le cas de l'avion, c'est l'air qui bouge relativement à l'aile.

plus d'autonomie. Le tableau 1.1 recense les recherches universitaires sur cette question.

Dans la suite de cette section, nous allons présenter les principales problématiques abordées en terme de contrôle ainsi que les solutions trouvées.

1.3.1 Problématiques abordées

Les deux problématiques présentées dans cette section interviennent à des niveaux différents dans le contrôle d'un robot volant et sont complémentaires. Elles sont le reflet des aspects essentiels du contrôle d'un robot : comment percevoir son environnement et comment interagir avec lui ? Chacune d'elles se pose d'une façon particulière sur des robot volants, par rapport à ce qui est connu et utilisé sur des robots terrestres.

Capteurs

La perception de l'environnement pose le problème de la référence au sol. Un robot terrestre se déplace par rapport au sol et il lui est possible d'observer, grâce à ses capteurs odométriques, ses mouvements relatifs et de les intégrer pour en déduire sa nouvelle position[7]. Un robot volant est en mouvement dans une masse d'air, qui est elle même en mouvement par rapport au sol. Il n'a aucun moyen simple et direct de mesurer ses mouvements relativement au sol. Des capteurs lui permettent de mesurer sa vitesse relativement à l'air environnant, mais les seuls moyens dont il dispose pour se repérer sont le GPS[8] et les informations visuelles issues d'une caméra. Le GPS est un capteur qui nécessite de connaître le terrain, la position absolue des objectifs à suivre ou à observer et celle des éventuels obstacles ; il ne peut pas être utilisé directement pour suivre une cible mobile par exemple. En revanche, la vision donne des informations relatives à l'environnement du robot qui sont directement exploitables, sans disposer d'une carte connue à l'avance.

Les capteurs visuels sont susceptibles de fournir deux types d'informations : des informations statiques et des informations dynamiques.

Les informations statiques sont fournies par des algorithmes de traitement d'images et de reconnaissance des formes. Ce sont des informations de positionnement relatif d'amers pertinents pour le comportement en cours : amers de repérage servant au module de navigation par exemple. Les applications associées sont le positionnement précis au-dessus d'un objectif au sol ainsi que, plus généralement, la navigation par amers qui consiste à suivre une trajectoire en allant d'un point de repère visuel à un autre (Gourichon *et al.*, 2002).

Les informations dynamiques permettent d'extraire des données relatives au mouvement du robot. En effet, l'étude du déplacement des différents points d'une image (flux optique) permet de déduire les mouvements du robot relativement à son environnement, information précieuse et

[7]Ce procédé est cependant instable à long terme et nécessite des méthodes de recalage (Filliat et Meyer, 2003; Meyer et Filliat, 2003).

[8]Global Positioning System : système de repérage par satellite donnant la position absolue exacte, à une marge d'erreur près, de l'appareil.

Nom	**Université**	**Aéronef**	**Contrôleur**	**Référence**
AURORA	LRV/ITI Brésil	Dirigeable	théorie du contrôle	(Carvalho *et al.*, 2001)
AVATAR	USC	Hélicoptère	subsomption (Brooks, 1986)	(Garcia-Pardo *et al.*, 2001 ; Vaughan *et al.*, 2001)
BEAR	Berkeley	Hélicoptère	contrôle flou	(Hoffmann, 1998 ; Hoffmann *et al.*, 1998 ; Shim *et al.*, 1998)
GTAR	Georgia Tech	Hélicoptère	théorie du contrôle et réseaux de neurones	(Kahn et Kannan, 2000)
HELI	CMU	Hélicoptère	réseaux de neurones	(Bagnell et Schneider, 2001)
HUMMINGBIRD	Stanford	Hélicoptère	théorie du contrôle	(Conway, 1995 ; Jones, 1998)
Melissa	Zurich	Dirigeable	réseaux de neurones (EMD)	(Iida, 2001)
Sarus & Jabiru	RMIT, Melbourne	Avion	contrôle flou	(Belton *et al.*, 1997)
Solus, ...	UMCP	Avion	CIRCA(Musliner, 1993)	(Atkins, 1999)
UAV	Sidney	Avion	théorie du contrôle	(Wong, 2001)
X36	NASA, USA	Avion	théorie du contrôle et réseaux de neurones	(Brinker et Wise, 2000)
Hélicoptère	CVS, Australia	Hélicoptère	réseaux de neurones, biovision	(Srinivasan *et al.*, 1998)
Hélicoptère	MIT/Draper Lab	Hélicoptère	théorie du contrôle	(Johnson *et al.*, 1996)
Hélicoptère	ETH, Zürich	Hélicoptère	théorie du contrôle	(Chapuis *et al.*, 2001)
Dirigeable	EPFL	Dirigeable	réseaux de neurones	(Zufferey *et al.*, 2002)
Karma	LAAS-CNRS	Dirigeable	théorie du contrôle	(Lacroix *et al.*, 2002)
DIVA et ROBEA	UTC Compiègne	VTOL	théorie du contrôle	(Dzul *et al.*, 2002)
LOTTE	Université de Stuttgart	Dirigeable	théorie du contrôle	(Kungl *et al.*, 2001)
Blimp	University of Queensland, Australia	Dirigeable	théorie du contrôle	(Wyeth et Barron, 1998)

TAB. 1.1: *Principaux projets universitaires concernant des UAV. Nous nous sommes restreints aux projets ayant donné lieu à publications. Abréviations : EMD : détecteur de mouvement élémentaire, IM : Inversion de Modèle, CIRCA : Cooperative Intelligent Real Time Control Architecture.*

difficile à obtenir par d'autres moyens sur des robots volants. La vision peut ainsi être utilisée directement pour contrôler un robot volant (Zhang et Ostrowski, 1999), même dans des phases critiques comme l'atterrissage (Bosse, 1997). La gestion des données visuelles pose toujours de nombreux problèmes car les algorithmes existants nécessitent d'importantes capacités de calcul, qui ne sont pas toujours compatibles avec la puissance embarquée. Les animaux ont pourtant résolu le problème et parfois de façon très simple ; aussi la biologie est-elle une importante source d'inspiration. Un odomètre visuel permettant de mesurer la distance parcourue, peut être ainsi défini en utilisant des principes inspirés de la physiologie et du comportement des abeilles (Iida et Lambrinos, 2000). De même, l'exploitation du flux optique, tel que le fait la mouche, permet de doter les robots de nombreuses capacités (Franceschini, 2002), comme le suivi de cible (Viollet et Franceschini, 2001) ou l'évitement d'obstacles en vol (Netter et Franceschini, 1999).

Contrôle

Un robot volant dispose de six degrés de liberté, là où les robots terrestres n'en disposent que de trois. Il est donc indispensable, pour pouvoir contrôler l'appareil, d'être en mesure de connaître la position de l'aéronef relativement à toutes ses dimensions, mais aussi d'avoir une influence sur ses degrés de liberté. Les robots volants ont, de plus, un domaine de fonctionnement à l'intérieur duquel ils doivent se maintenir pour rester contrôlables. Ce domaine de viabilité (Guillot et Meyer, 2001; Guillot et Meyer, 2003), appelé “domaine de vol” en aéronautique, impose des contraintes sur les différentes variable d'état que le contrôleur doit absolument respecter.

La problématique du contrôle est très générale : comment appliquer une consigne de la manière la plus efficace ? Quels ordres envoyer ? Une théorie du contrôle a été développée pour répondre à de tels besoins et des techniques éprouvées sont actuellement utilisées par les ingénieurs (pour une introduction, voir (Crosnier *et al.*, 2001)). Ces techniques nécessitent néanmoins une connaissance précise du système contrôlé ainsi que de son environnement. Elles utilisent un modèle précis de l'environnement, et déterminent généralement le contrôleur à utiliser par inversion des équations de ce modèle[9].

Lorsque le modèle de l'environnement et de ces interactions avec le robot ne sont pas exploitables ou inconnues, il est possible d'utiliser d'autres types de contrôleurs, comme des contrôleurs flous ou des réseaux de neurones, plus adaptés à des techniques de conception automatiques.

La place des réseaux de neurones est très variable parmi les travaux cités ici. Le réseau de neurones peut tout d'abord constituer le contrôleur complet. Dans (Bagnell et Schneider, 2001), il est de type perceptron et sa structure est fixe et donnée initialement. Ses poids sont optimisés avec un algorithme d'apprentissage par renforcement. Zufferey et al. (Zufferey *et al.*, 2002) utilisent un réseau de neurones à décharge conçu par un algorithme évolutionniste. Le nombre de neurones est fixe et la connectivité est évoluée pour faire naviguer un dirigeable à partir d'informations visuelles dans une arène. Les expériences ont été réalisées sur un dirigeable

[9] Si les équations sont non linéaires, on considére que l'on reste dans une zone linéaire proche du point de fonctionnement.

réel et ont pour objectif de faire avancer le dirigeable le plus possible pendant les 40 secondes que durent une évaluation.

Cependant, dans certains cas, le réseau de neurones n'est qu'une partie du contrôleur global. Ainsi, lorsque le système est non-linéaire, des réseaux de neurones peuvent être ajoutés à un contrôleur plus classique pour corriger les défauts de ceux-ci. Ils sont ainsi couramment utilisés pour atténuer les erreurs d'inversion de modèle (Brinker et Wise, 2000; Kahn et Kannan, 2000).

Dans les travaux que nous allons présenter ici, nous allons utiliser un réseau de neurones comme seul et unique contrôleur. Les techniques que nous allons utiliser ne reposent pas sur une connaissance approfondie du système, contrairement aux approches basées sur un modèle du système, notre objectif étant de permettre au système de découvrir par lui-même, autant que possible, la manière de réaliser efficacement le comportement désiré. Nous utiliserons des contrôleurs Proportionnel-Intégral-Dérivé (PID), issus de la théorie du contrôle, en tant que contrôleurs témoins pour évaluer les performances de nos algorithmes.

Les principes des contrôleurs réseaux de neurones, des contrôleurs flous ainsi que les contrôleurs de base de la théorie du contrôle, à savoir les contrôleurs PID, seront présentés au chapitre suivant.

1.4 Concours de robots volants

Plusieurs compétitions universitaires ont été créées pour attirer l'attention sur ce domaine et stimuler les recherches.

1.4.1 International Aerial Robotics Competition

Une compétition internationale a été organisée aux États-Unis pour promouvoir les recherches autour des UAV[10]. L'objectif est de concevoir des systèmes volants autonomes capables d'aider des secouristes lors de sauvetages après un tremblement de terre, une catastrophe nucléaire ou un incident biologique.

La compétition existe depuis une dizaine d'années et la complexité des comportements exhibés par les robots a fortement augmenté au cours des épreuves succcessives. Pendant les premières éditions, il suffisait de pouvoir explorer une zone prédéfinie et de reconnaître un symbole radioactif apposé sur un conteneur, puis d'en envoyer les coordonnées précises à une base au sol. Dans les éditions suivantes, il fallait pouvoir repérer un objet et le déplacer avec précision. Dans la dernière édition, la tâche à accomplir consistait à repérer différents types d'objets potentiellement dangereux pour les équipes de secouristes, à trouver les rescapés et les victimes d'une catastrophe de type tremblement de terre, enfin à envoyer les coordonnées de ces différents points d'intérêt à des équipes humaines au sol supposées ensuite intervenir sur le site.

[10] `http://avdil.gtri.gatech.edu/AUVS/IARCLaunchPoint.html`

1.4.2 Concours micro-drones de la DGA

Un concours sur le thème des micro-drones a été lancé en 2002 par l'ONERA sous le parainnage de la DGA[11]. L'objectif de la compétition est de concevoir et réaliser un micro-drone, d'une envergure inférieure à 70 cm, capable d'explorer son environnement et de repérer des zones dangereuses : véhicules, barrages, tireurs embusqués. Le drone doit alors indiquer les positions de ces emplacements à une équipe d'intervention afin de définir une trajectoire sûre permettant d'atteindre un objectif donné.

1.5 Notre problématique

Nous allons utiliser les robots aériens en tant que plateformes de test pour nos algorithmes de contrôle. Les aspects qui nous ont le plus intéressé sont le grand nombre de degrés de liberté, ainsi que la robustesse nécessaire des contrôleurs mis en œuvre. En effet, ces systèmes disposent d'une inertie importante et peuvent être soumis à d'importantes sources de perturbations extérieures. De plus, les variables d'états de ces appareils doivent être maintenues à l'intérieur de limites dépendant de leurs caractéristiques. Ces limites définissent le domaine de vol que les contrôleurs doivent impérativement respecter, ce qui implique qu'ils soient d'autant plus robustes.

Contrairement à la majorité des approches citées ici, la nôtre ne repose pas sur une connaissance approfondie du modèle de fonctionnement du robot. La seule information disponible sur le système est celle qui est acquise lors des expérimentations directes des solutions proposées. Les équations du modèle, à partir desquelles nous avons défini les différents simulateurs que nous avons utilisés, sont totalement inconnues de l'algorithme de conception, si ce n'est par le biais de l'évaluation de la qualité et de la robustesse d'une solution au travers de la simulation.

[11] `http://concours-drones.onera.fr/`

Chapitre 2

Contrôle de robots autonomes

Ce chapitre présente une revue des méthodes existantes pour le contrôle de robots autonomes. Nous nous sommes concentré sur les techniques utilisables en conjonction avec des algorithmes d'apprentissage ou d'optimisation automatique. Dans la suite de ce mémoire, nous supposons que les capteurs et les effecteurs sont adaptés aux problèmes que nous souhaitons résoudre et nous nous concentrons sur le système de contrôle qui les relie.

Nous nous intéressons donc ici aux différents types de contrôleurs existants, avant de développer les techniques d'apprentissage pouvant servir à les concevoir et à les optimiser.

2.1 Types de contrôleurs

Les entrées et les sorties du système connues, il reste à trouver comment relier les unes aux autres pour implémenter le comportement désiré. Plusieurs stratégies existent que nous allons détailler dans la suite de cette section.

2.1.1 Programmes

Mis à part les systèmes fondus directement dans le silicium ou implémentés sous forme électronique, voire sous forme mécanique, la plupart des contrôleurs se présentent sous forme de logiciels informatiques. Ceux que nous allons décrire dans cette section sont exprimés directement dans le formalisme d'un programme informatique et sont manipulés en tant que tel par l'algorithme de conception automatique. Autrement dit, on ne passe pas par un formalisme intermédiaire (comme celui des réseaux de neurones), et c'est directement le programme qu'on manipule.

Automates à états finis

Les automates à états finis sont des programmes représentables sous la forme de graphes dont les nœuds sont des états et les arêtes des transitions conditionnelles à chacune desquelles est associée une action.

Le passage d'un état à un autre est régi par des lois de transition possèdant une partie "condition" - qui doit être remplie pour que la transition soit possible - et une partie "action" - qui est exécutée lors de l'exécution de la transition (voir figure 2.1 pour un exemple). À chaque pas de temps, une seule transition est effectuée.

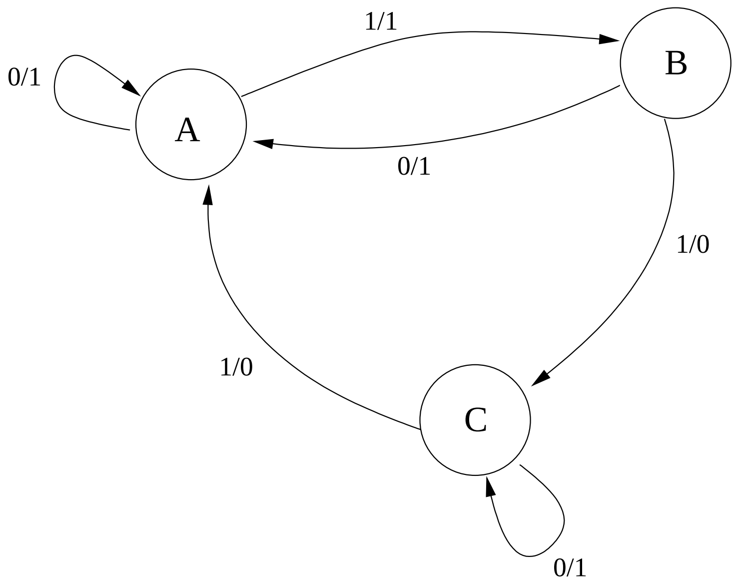


FIG. *2.1: Exemple d'automate à états finis avec 3 états : A, B et C. Le système dispose d'une entrée et une sortie. Les lois de transition sont de type <entrée>/<sortie>. Si le programme est dans l'état B, il passera à l'état A si l'entrée vaut 0, tout en émettant la valeur 1 en sortie ; sinon il passera à l'état C tout en émettant la valeur 0 en sortie.*

Programmation fonctionnelle

La programmation fonctionnelle est un formalisme dans lequel les programmes sont des fonctions. Ces programmes peuvent être représentés sous forme d'arbres dont les nœuds sont des instructions. La racine est la fonction principale et les feuilles en sont les paramètres. Ce type de formalisme a été utilisé dans la *programmation génétique* (Koza, 1992; Koza, 1994; Koza *et al.*, 1999; Koza *et al.*, 2003), que nous présenterons à la section 3.3.3.

De tels programmes peuvent être utilisés directement pour contrôler le fonctionnement d'un robot, mais ils peuvent aussi définir l'étape intermédiaire de construction de son système de contrôle. Gruau les a ainsi utilisés pour contrôler le développement d'un réseau de neurones (Gruau, 1994) et Brave les a utilisés pour contrôler le développement d'un automate à états finis (Brave, 1996).

2.1.2 Systèmes à base de règles

Une autre possibilité pour concevoir un contrôleur de robot consiste à décomposer le comportement souhaité en comportements élémentaires, qui ne seront appliqués que si certains critères sont remplis. Cela conduit à concevoir des systèmes composés de *règles <conditions>-<actions>*. Ce principe est à l'origine de plusieurs types de systèmes de contrôle en fonction de la nature des conditions et du type d'action possible.

Règles *SI ... ALORS ...*

Les règles de production.

Grefenstette et Schultz ont ainsi utilisé dans leur système baptisé SAMUEL (Grefenstette et Schultz, 1994) des règles de type :

SI capteur3 est dans [35,45] ET $sonar2 < 20$ *ALORS tourner* $= -24$ *(force 0.8)*

Les conditions portent sur la valeur des capteurs et les actions portent sur les valeurs à appliquer aux effecteurs. Le contrôleur est composé de l'ensemble de ces règles. À chaque pas de temps, les règles dont la partie "condition" est remplie sont utilisées pour déterminer les consignes à envoyer aux effecteurs. Un mécanisme de pondération permet de lever les ambiguïtés lorsque plusieurs règles sont valides en même temps. La force, paramètre associé à chaque règle, permet ainsi de résoudre les conflits potentiels entre règles concurrentes. La valeur à appliquer à l'effecteur est la moyenne des valeurs proposées par les règles valides, moyenne pondérée par leurs forces. Ces forces sont optimisées par l'algorithme de conception (un algorithme évolutionniste dans le cas de SAMUEL).

Classeurs

Les *classeurs* sont inspirées de travaux de Holland (1986). Les règles utilisées ont une partie "condition" et une partie "action" binaires (Holland et Reitman, 1978). Elles sont de la forme :

1011##10#1#00 → 1001011

La partie gauche est la condition de la règle. Le symbole # correspond à un joker, qui indique de ne pas prendre en compte l'entrée correspondante. La partie droite est la partie "action".

Dans cet exemple, cette règle sera active si l'entrée n°1 vaut 1, l'entrée n°2 vaut 0, si la troisième entrée vaut 1 ... Les entrées 5, 6, 9 et 11 ne sont pas prises en compte. Si la règle est active, la sortie n°1 vaudra 1, la deuxième 0, la troisième 0...

D'autres règles plus élaborées contiennent également une anticipation sur les prochaines perceptions dans la partie condition, ce qui permet de réaliser de l'apprentissage latent, qui n'est pas guidé par un objectif particulier (Gérard, 2002).

Règles *floues*

Les systèmes flous ou systèmes à base de *logique floue* (Zadeh, 1965; Bouchon-Meunier, 1995), sont composés de règles dont la partie "condition" fait intervenir des ensembles flous. Les ensembles flous sont des ensembles dont la fonction d'appartenance[12] n'est pas à valeurs binaires, mais à valeurs réelles et variables entre 0 et 1. La logique floue n'est donc pas uniquement

[12]La fonction d'appartenance d'un ensemble est une fonction booléenne associant à chaque élément la valeur "vrai", si l'élément fait partie de l'ensemble ou "faux" sinon.

composée de prédicats vrais ou faux, mais aussi de prédicats de valeurs intermédiaires. Elle est utilisée, par exemple, pour formaliser des concepts abstraits comme “grand”, “petit” ou “moyen”. Parlant d'êtres humains, par exemple, 1m70 commencera à être “grand”, mais moins que 1m80 ou 1m90 (figure 2.2).

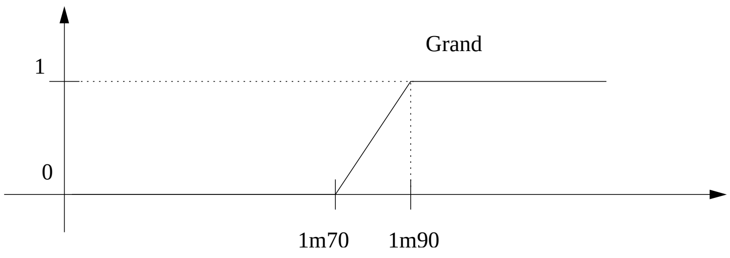


FIG. *2.2: Exemple de concept flou d'un “grand” humain. En abscisse, taille, en ordonnée, appartenance à l'ensemble flou.*

L'appartenance à l'ensemble flou, aussi appelée *crédibilité*, peut être utilisée pour déterminer un poids à attribuer à la règle. L'action à effectuer sera choisie parmi les actions possibles en prenant cette pondération en compte. Par exemple, il est, possible d'appliquer l'action qui résulte de la moyenne des actions possibles, pondérée par la crédibilité de chacunes d'elles. Le passage d'une règle à une autre se fait alors de façon très souple.

Les règles de Mamdani (Mamdani, 1974) sont des règles à plusieurs entrées et à sortie unique de la forme :

SI x_1 est A_1 ET ... ET x_n est A_n ALORS y est B

où A_i et B sont des ensembles flous. En général, cependant, les systèmes robotiques imposent une sortie continue, aussi ce type de contrôleur nécessite-t-il une étape délicate de *dé-flouification*[13], prenant en compte la sortie de chaque règle ainsi que sa crédibilité. Cette étape est relativement complexe et constitue la principale faiblesse de ce type de système. Cette méthode a, cependant, été utilisée pour contrôler une voiture réelle (Garnier et Fraichard, 1997).

Règles TSK. Takagi, Sugeno et Kang ont proposé des règles différentes, appelées *règles TSK* (Takagi et Sugeno, 1985), inspirées des règles de Mamdani, mais avec une conclusion non floue qui permet d'éviter l'étape complexe de dé-flouification.

SI x_1 est A_1 ET ... ET x_n est A_n ALORS $y = \sum_i k_i \times x_i$

où les A_i sont des ensembles flous. La conclusion de la règle est la somme des entrées pondérées par des coefficients k_i. À chaque pas de temps, le système calcule la conclusion de chaque règle et la sortie globale est alors la somme des sorties des règles, pondérées par leur crédibilité.

[13]“Defuzzification” en anglais. Étape permettant de retrouver une valeur réelle à partir d'ensembles flous

2.1.3 Réseaux de neurones

Le neurone biologique

Le *neurone* biologique est une cellule ramifiée, recevant des informations provenant d'autres neurones grâce à ses *dendrites* et envoyant des signaux grâce à son *axone*. La connexion entre plusieurs neurones est réalisée par des *synapses* qui assurent la transmission d'un signal d'une cellule à l'autre par des moyens chimiques ou électriques. Les synapses disposent d'une efficacité de transmission de l'information variable : un neurone peut ainsi avoir plus ou moins d'influence sur les neurones auxquels il est connecté. Cette influence peut varier au cours du temps en fonction de mécanismes d'apprentissage.

Le signal transmis le long des axones, synapses puis dendrites est un *potentiel d'action*, une décharge électrique véhiculée en propageant une différence de potentiel entre l'intérieur et l'extérieur de l'axone ou de la dendrite. Le passage du potentiel d'action du neurone pré-synaptique au neurone post-synaptique s'effectue grace à des *neurotransmetteurs* émis par la terminaison synaptique du neurone pré-synaptique et captés par la membrane du neurone post-synaptique.

Le neurone formel de McCulloch et Pitts

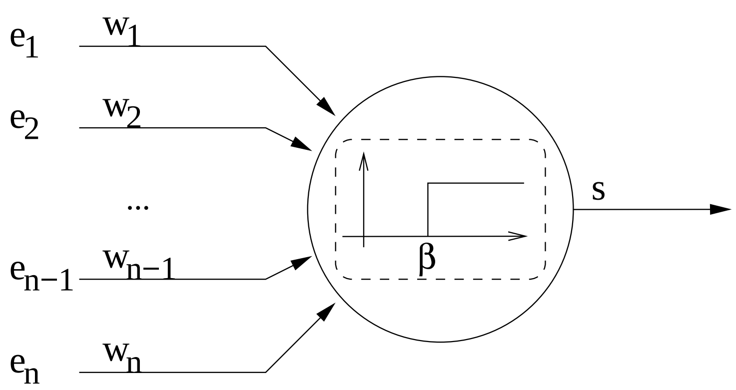


FIG. *2.3: Le neurone formel de Mc Culloch et Pitts.*

Le *neurone formel* de McCulloch et Pitts (McCulloch et Pitts, 1943) est une version simplifiée du modèle biologique (figure 2.3). C'est un automate booléen : toutes les entrées et la sortie du neurone sont des booléens. Chacune des entrées e_i est pondérée par une efficacité synaptique w_i. L'activation du neurone est la somme des entrées pondérées par leurs efficacités synaptiques. La sortie s du neurone ne sera active que si son activation dépasse un certain seuil β :

$$s = f\left(\sum_{i=0}^{n} w_i.e_i\right)$$

f étant la fonction seuil de paramètre β :

$$f(x) = \begin{cases} 0 & \text{si } x < \beta \\ 1 & \text{si } x \geq \beta \end{cases}$$

Le neurone formel généralisé

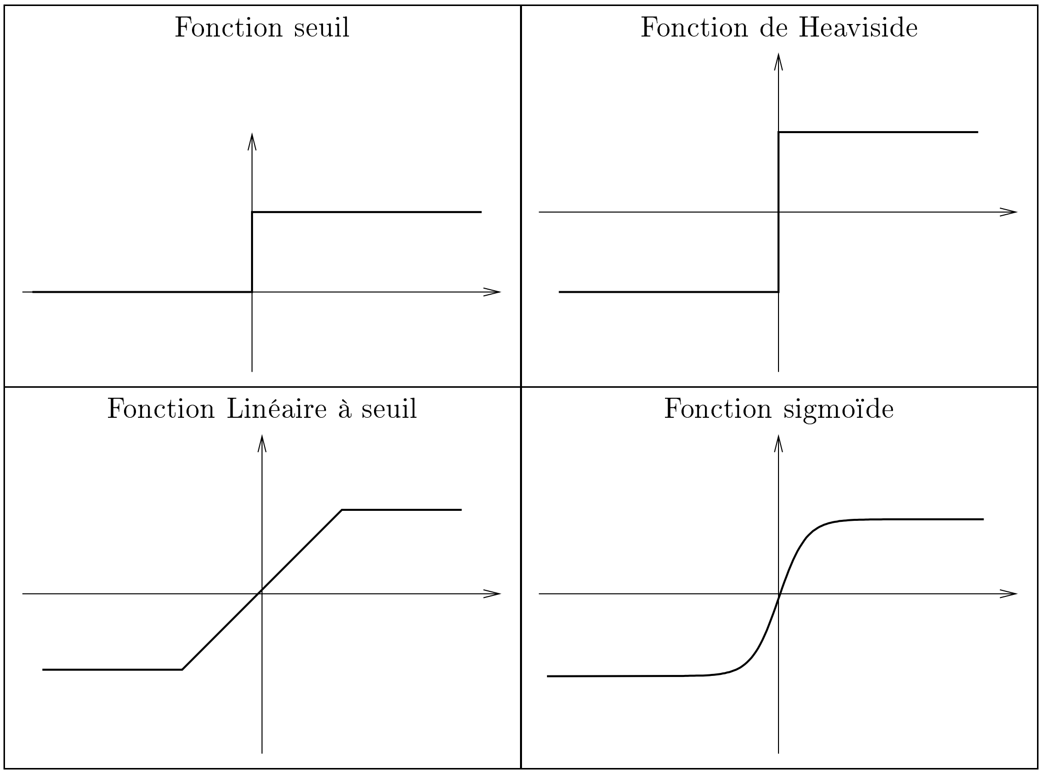


FIG. *2.4: Différentes fonctions de transfert utilisables.*

De manière plus générale, un neurone formel peut être vu comme une unité de calcul élémentaire recevant des entrées et générant une sortie (Rumelhart et McClelland, 1986). Plusieurs paramètres du modèles peuvent le différentier du modèle initial de McCulloch et Pitts :

- La nature des entrées
- La fonction d'activation
- La fonction de transfert
- La nature de la sortie

Les entrées et les sorties peuvent être binaires $((-1, +1)$ ou $(0, 1))$ ou réelles. Elles peuvent être de même type ou de type différent[14].

L'*activation a* du neurone est un paramètre interne décrivant l'état du neurone. Il est calculé à partir des entrées par la fonction d'activation du neurone. Sa forme la plus simple est la somme pondérée des entrées :

[14]Il est ainsi possible d'avoir, par exemple, un neurone avec des entrées réelles et une sortie binaire.

$$a = f_a(e_1, \ldots, e_n) = \sum_{i=0}^{n} w_i.e_i$$

Cependant, pour des neurones à dynamique plus complexe, comme les neurones intégrateurs à fuite, cette fonction peut prendre d'autres formes :

$$\tau \frac{da}{dt} = -a + \sum_{i=0}^{n} w_i.e_i$$

La valeur de sortie est obtenue en appliquant une *fonction de transfert* à l'activation. Les fonctions de transfert les plus couramment utilisées sont les suivantes (figure 2.4) :

- fonction binaire à seuil : variantes de la fonction signe ou de la fonction de heaviside
- fonction linéaire à seuil
- fonction sigmoïde : $f(x) = tanh(x)$ ou encore $f(x) = \frac{1}{1+\exp^{-x}}$
- fonction stochastique :

$$f(x) = \begin{cases} 1 & \text{avec la probabilité } \frac{1}{1+e^{\frac{-x}{T}}} \\ 0 & \text{sinon} \end{cases}$$

  T étant un paramètre appelé température. Lorsqu'il tend vers 0, cette fonction tend vers la fonction seuil
- toute autre fonction généralement croissante et bornée.

Ces différentes fonctions peuvent être paramétrées. Un biais peut être ajouté de façon à décaler l'origine sur l'axe des abscisses. De plus, la pente à l'origine peut aussi être changée pour la fonction linéaire à seuil ou la fonction sigmoïde de façon à prendre des allures intermédiaires entre une fonction linéaire sur le domaine de fonctionnement (pente faible tendant vers 0) et une fonction de Heaviside (pente infinie).

Réseaux de neurones

Un *réseau de neurones* est un graphe orienté dont chacun des nœuds est un neurone formel. À chaque lien est associé un *poids synaptique*, reflet de l'efficacité synaptique du neurone biologique.

La structure de réseau la plus étudiée et la plus utilisée jusqu'à présent est le *perceptron* (Rosenblatt, 1965), ou réseau à couches (figure 2.5). Dans ce modèle, les neurones sont disposés par couches. Les neurones d'une couche reçoivent leurs entrées uniquement des neurones de la couche précédente. Le nombre de neurones sur la ou les couches cachées[15] dépend de la complexité du problème. Certains algorithmes permettent d'ajuster ce paramètre (Fahlman et Lebiere, 1991; Rivals et Personnaz, 2000), cependant, il est généralement déterminé empiriquement.

Un perceptron à une couche cachée[16] peut approximer n'importe quelle fonction continue s'il y a suffisamment de neurones cachés, en utilisant une fonction de transfert continue et différentiable (Cybenko, 1989; Hornik, 1993). Il peut en particulier approximer le fonctionnement

[15] Couche de neurones qui ne sont ni des entrées, ni des sorties du système.
[16] C'est-à-dire ayant une couche de neurones entre les entrées et les sorties.

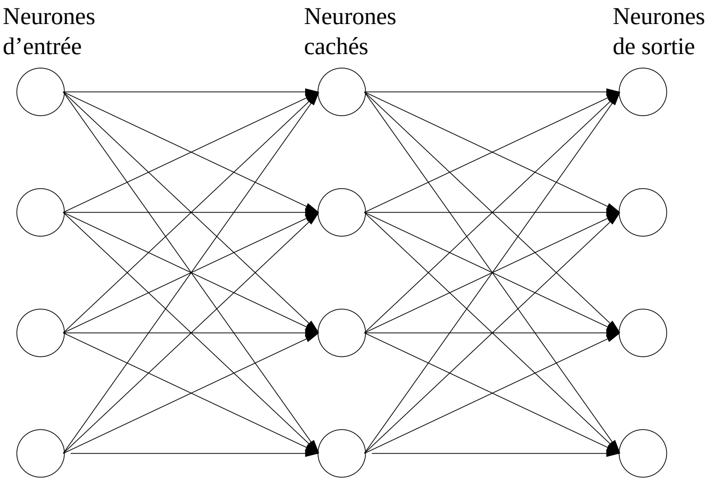


FIG. *2.5: Perceptron à une couche cachée. Tous les neurones d'une couche n reçoivent en entrée les sorties des neurones de la couche $n-1$ et projettent leur sortie vers les neurones de la couche $n+1$.*

d'un contrôleur PID, système de base du contrôle classique dont nous parlerons ci-après. Avec deux couches cachées, un perceptron multi-couches peut approximer n'importe quelle fonction calculable (Lapedes et Farber, 1987; Cybenko, 1988).

Cependant, les réseaux de neurones ne sont pas limités à ces structures. Il est possible de connecter entre elles les unités d'une même couche cachée ou d'ajouter en entrée du réseau les sorties des neurones de la couche cachée de façon à prendre en compte le dernier pas de temps. Il est également possible de connecter tous les neurones entre eux ou même d'utiliser des structures quelconques (figure 2.6).

Un réseau de neurone peut gérer un nombre quelconque d'entrées et de sorties et prendre en compte le passé ou générer des dynamiques complexes sans avoir à faire de modifications ou d'ajout au formalisme initial. Ce type de contrôleur est celui que nous avons retenu pour nos expérimentations car il dispose d'un important pouvoir expressif qui lui permet d'être utilisé dans de nombreuses applications (Fogelman-Soulie et Gallinari, 1998). Des réseaux de neurones ont ainsi été utilisés pour gérer l'altitude d'avions afin d'éviter les collisions (LeFablec et Alliot, 1999; Durand *et al.*, 2000) ou pour la reconnaissance de caractères écrits (Knerr *et al.*, 1992; LeCun *et al.*, 1998a; Bellili *et al.*, 2002).

Associés à l'évolution, les réseaux de neurones permettent ainsi de résoudre des problèmes de contrôle très variés allant du contrôle de robots à roues au contrôle de robots marcheurs, nageurs ou volants (Doncieux, 1999; Meyer *et al.*, 2002).

2.1.4 Contrôleurs P, PI, PD et PID

Lorsque le contrôle se ramène au maintien d'une variable à une valeur cible et que le domaine de fonctionnement est linéaire, il est possible d'appliquer des correcteurs de type Proportionnel (P), Proportionnel Intégral (PI), Proportionnel Dérivé (PD) ou encore Proportionnel Intégral

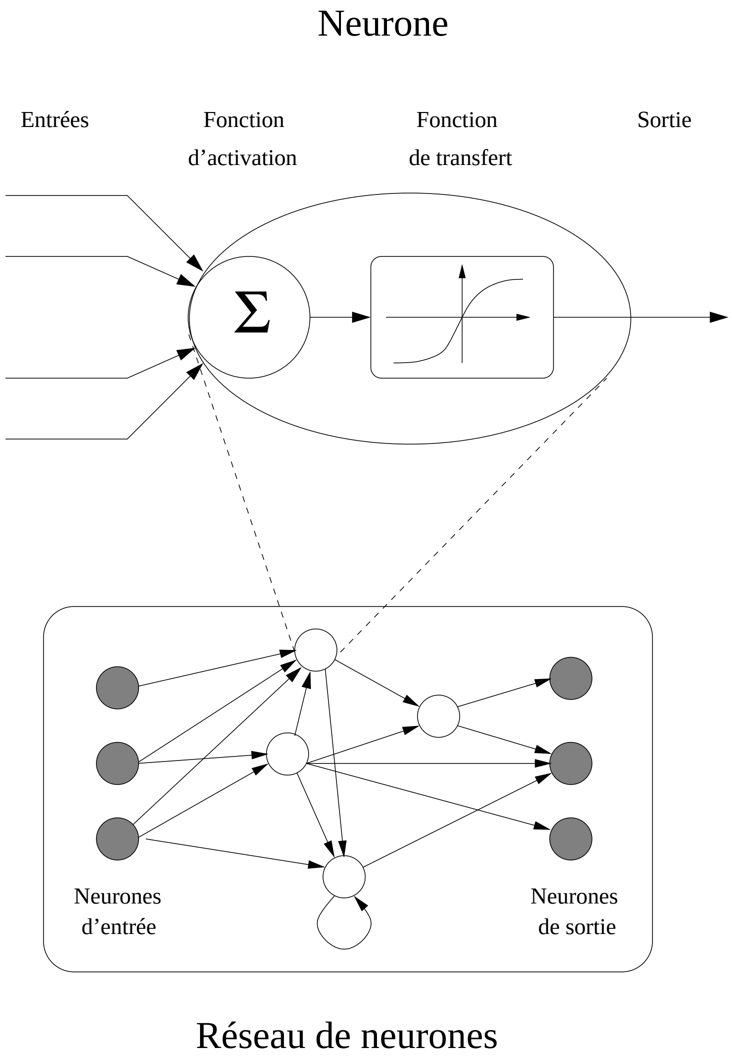


FIG. *2.6: Réseau de neurones de structure quelconque.*

Dérivé (PID). Le principe consiste alors à fournir en entrée du système l'erreur sur la variable à maintenir - autrement dit l'écart entre la valeur souhaitée et la valeur réelle - ainsi que, selon les cas, sa dérivée et/ou son intégrale. La sortie du système vaut alors :

$$\begin{array}{rrcl} \text{P :} & s(t) & = & k_p\epsilon(t) \\ \text{PD :} & s(t) & = & k_p\epsilon(t) + k_d\frac{d\epsilon(t)}{dt} \\ \text{PI :} & s(t) & = & k_p\epsilon(t) + k_i\int\epsilon(t)dt \\ \text{PID :} & s(t) & = & k_p\epsilon(t) + k_i\int\epsilon(t)dt + k_d\frac{d\epsilon(t)}{dt} \end{array}$$

Les coefficients k_p, k_d et k_i étant respectivement les coefficients proportionnel, integral et dérivée. $\epsilon(t)$ est l'entrée du système, c'est l'erreur par rapport à la consigne x_d sur la variable que x l'on cherche à maintenir :

$$\epsilon(t) = x - x_d$$

Pour illustrer les capacités des PID, nous allons étudier les performances de ce type de contrôleur pour stabiliser un pendule monté sur un chariot. L'objectif du contrôleur est de faire bouger le chariot de façon à maintenir le pendule à la verticale. Cet exemple correspond à une version simplifiée du pendule inversé sur lequel nous reviendrons plus en détail au chapitre 7[17].

Ce problème se résout à l'aide d'un correcteur PD recevant en entrée l'erreur de position angulaire[18] du pendule et générant en sortie la commande à appliquer au moteur faisant bouger le chariot. Les coefficients optimaux sont $k_p = 1$ et $k_d = 0.13$ (optimisés par une Stratégie Évolutionniste, voir section 3.1.5). Le terme intégral n'est pas nécessaire sur ce problème.

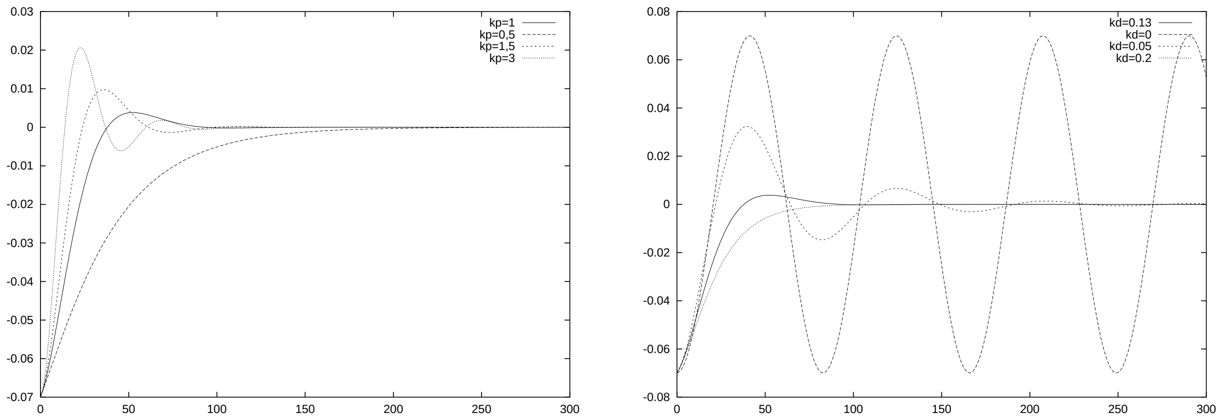


FIG. *2.7: Comportement du pendule contrôlé par un correcteur PD lorsque l'on fait varier les coefficients k_p et k_d. L'ordonnée est l'angle du pendule par rapport à la verticale (en radians). Les abscisses sont en pas de temps (un pas de temps représente 0.01s). Le pendule part avec un angle initial de $-0,07rad$ et doit être ramené à $0rad$. À gauche, comportement du correcteur PD lorsque l'on fait varier le coefficient k_p (avec $k_d = 0,13$); à droite, lorsque l'on fait varier le coefficient k_d (avec $k_p = 1$).*

La partie proportionnelle est la partie essentielle du contrôleur, c'est elle qui donne la puissance au moteur. Le réglage du coefficient k_p influe sur les performances dynamiques. Il va modifier le temps de réponse du système, ainsi que le dépassement (figure 2.7). Un coefficient k_p trop faible correspondra à une correction lente (pente initiale faible de la courbe $k_p = 0,5$), un coefficient trop élevé donnera lieu à une correction rapide (pente initiale élevée), mais avec un dépassement[19] important . La partie dérivée ne joue pas sur la rapidité du contrôle (pentes initiales identiques sur la courbe de gauche sur la figure 2.7), mais augmente la stabilité du contrôleur. Elle permet d'amortir les oscillations. Dans le cas du pendule, le contrôleur est incapable de stabiliser le système sans ce terme (cas $k_d = 0$).

La partie intégrale améliore la précision statique du contrôle. Elle diminue l'écart entre la consigne et la valeur réelle observée en régime permanent. Elle n'est pas nécessaire pour le contrôle du pendule, puisqu'un correcteur PD parvient déjà à annuler cette erreur.

[17]Dans la version canonique de l'expérience du pendule inversé, il faut également contrôler la position du chariot.

[18]L'angle du pendule par rapport à la verticale.

[19]Différence maximale entre le régime permanent et le régime transitoire. Il est par exemple de $0,02rad$ dans le cas $k_p = 3$ de la figure 2.7.

Ce type de contrôleur est particulièrement bien adapté au contrôle bas-niveau de manière générale et au contrôle de robots volants en particulier lorsque l'objectif est le maintien de variables d'état à l'intérieur d'un domaine de viabilité. Cette méthode est la plus simple utilisée pour résoudre ce type de problèmes. Cependant, ce type de contrôleur n'accepte qu'une entrée et une sortie et ne fonctionne bien que dans un contexte linéaire, autour d'un point de fonctionnement. Il correspond à un contrôleur élémentaire exploité au sein de structures plus complexes.

2.2 Méthodes de conception

Riedmiller (Riedmiller, 1994) a classé les différentes méthodes de conception existantes en fonction de la connaissance disponible. La figure 2.8 reprend et complète cette classification.

Les différentes stratégies sont regroupées en trois catégories en fonction du type de connaissance utilisé : connaissance au niveau contrôle, connaissance au niveau système[20] et pas de connaissance a priori.

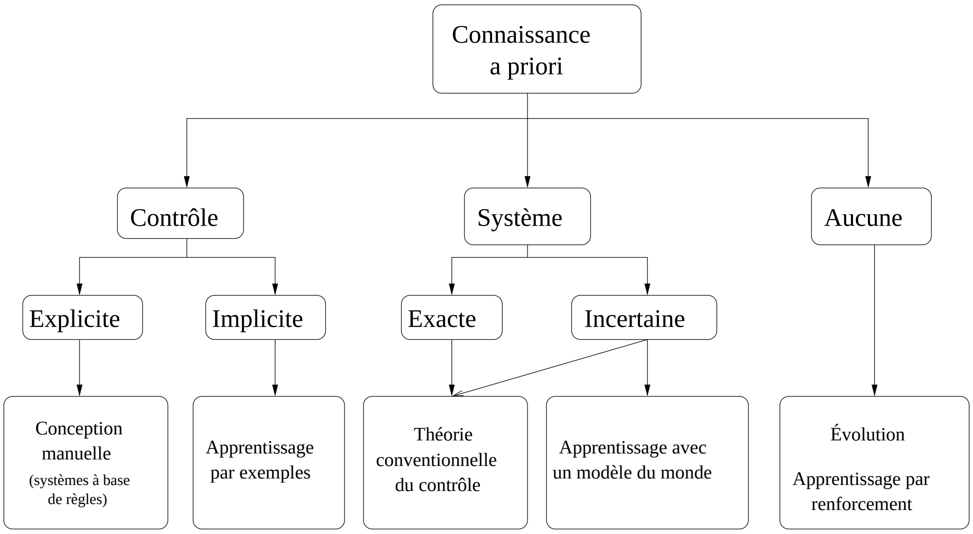


FIG. *2.8: Classification des différents contrôleurs et des méthodes de conception.*

Les systèmes de la première catégorie (contrôle) exploitent des connaissances quant à la façon de résoudre le problème. Ce sont des connaissances procédurales qui peuvent être obtenues à partir d'experts du domaine et être exploitées dans les systèmes à base de règles, ou bien des exemples permettant d'entraîner progressivement le contrôleur.

Le deuxième type de méthodes de conception (système) requiert des connaissances quant à l'environnement dans lequel doit évoluer le robot. Les équations physiques de cet environnement et ses interactions avec le robot doivent être connues. Le contrôleur est obtenu en inversant ces équations autour d'un point de fonctionnement prédéterminé.

La troisième et dernière catégorie est composée de méthodes de conception qui ne nécessitent

[20] Connaissance de l'environnement dans lequel le robot évolue.

aucune connaissance particulière, mais qui peuvent cependant exploiter d'éventuelles connaissances disponibles.

Cette section va présenter les principaux types de méthode de conception automatique.

2.2.1 Apprentissage par exemples

L'apprentissage par exemples est un apprentissage supervisé. Une base de données contenant des exemples de perceptions et des commandes associées est utilisée pour faire apprendre le système. Dans le cas du contrôle de robot, cette base de données peut-être obtenue par enregistrement du comportement d'un expert humain. Cette technique, appliquée aux réseaux de neurones, se nomme la rétropropagation de l'erreur : on force en entrée du réseau les paramètres d'entrée d'un exemple, on compare les sorties fournies par le réseau avec celles de l'exemple et on rétropropage l'erreur dans le réseau (Werbos, 1974; LeCun, 1987; LeCun *et al.*, 1998b). Les réseaux de neurones fonctionnant avec ce type d'apprentissage doivent être de type perceptron, mais des variantes fonctionnent sur réseaux récurrents (ramenés à une forme canonique[21]) (Nerrand *et al.*, 1993; Pearlmutter, 1995).

Si cette technique est particulièrement bien adaptée à des problèmes non dynamiques, comme la reconnaissance d'écriture (Knerr *et al.*, 1992; LeCun *et al.*, 1998a; Bellili *et al.*, 2002), elle pose des problèmes de généralisation importants lorsqu'elle est appliquée au contrôle de robots. En effet, un ensemble d'exemples perception-action ne prend pas en compte l'aspect dynamique du système. C'est pourquoi les aspects les mieux appris seront ceux des conditions normales de fonctionnement. Mais si l'on s'éloigne un peu de ces conditions, si l'on entre dans une dynamique qui n'avait pas été rencontrée lors de la séance de récupération des données, on ne dispose d'aucune garantie de stabilité concernant le comportement exhibé. Pour que le réseau généralise au mieux face aux situations non encore rencontrées pendant son apprentissage, il faut que la base de données couvre au maximum l'espace des phases du système ; autrement dit, il faut que chaque zone de fonctionnement soit couverte par un ou plusieurs exemples, ce qui n'est pas toujours réalisable. (Ronco et Gawthrop, 1997) contient une revue des approches basées sur la rétropropagation de l'erreur à des fins de contrôle.

L'apprentissage par rétropropagation d'erreur peut, en revanche, être utilisé pour optimiser les paramètres d'un système dont le but est d'anticiper les perceptions futures en fonction des perceptions courantes (Nolfi *et al.*, 1994). En effet, dans ce cas, le système fournit une réponse qui peut immédiatement être comparée à la solution idéale dès le pas de temps suivant. Dans certaines approches, la connaissance du modèle de fonctionnement du système à contrôler permet de se passer de la connaissance d'une solution idéale (Hénaff *et al.*, 1994; Hénaff et S., 1996).

2.2.2 Apprentissage par renforcement

L'apprentissage par renforcement (Sutton et Barto, 1998) nécessite un retour d'information beaucoup plus faible que les algorithmes de type supervisé comme la rétropropagation de l'erreur.

[21] Tous les réseaux, récurrents ou non, peuvent se ramener à cette forme.

Aucune information n'est disponible quant aux erreurs commises, seul un signal de récompense ou de punition indique si la dernière séquence comportementale est bonne ou non. Le retour n'est pas toujours instantané, mais peut être retardé et n'intervenir qu'après une longue séquence comportementale.

Le principe de ces algorithmes consiste à partir d'une politique d'actions aléatoire π, associant à chaque couple (état, action) une probabilité particulière. L'algorithme va ensuite évaluer les performances de cette politique et l'améliorer progressivement. Pour cela, il évalue par estimations successives la récompense cumulée attendue à partir de chaque état :

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_t + 1)|s_t = s\} \end{aligned}$$

$\gamma < 1$ étant un coefficient d'amortissement.

Trois types d'apprentissage par renforcement peuvent être distingués en fonction de la méthode utilisée pour optimiser la politique d'actions : la Programmation Dynamique, les méthodes de Monte Carlo et l'apprentissage TD (*Temporal Difference*).

La Programmation Dynamique (Bellman, 1957) va exploiter un modèle complet du monde, contenant tous les états possibles ainsi que leur lois de transition, pour mettre à jour des estimations de plus en plus précises des récompenses cumulées associées à chaque état. L'amélioration de politique se fera ensuite par modifications simples et évaluation de ces modifications jusqu'à atteindre la politique optimale.

Les méthodes de Monte Carlo (Metropolis et Ulam, 1949) n'utilisent pas un modèle complet du monde mais basent leurs estimations sur des séquences expérimentales : partant d'un état s, on réalise un ensemble d'expériences qui nous permet de déduire une récompense moyenne attendue, en partant de cet état avec la politique d'action choisie. L'apprentissage se fera ensuite, comme en Programmation Dynamique, en modifiant progressivement la politique, tout en évaluant les répercussions de chaque modification.

L'apprentissage TD (Sutton, 1988) cumule les avantages des deux méthodes. Il ne nécessite pas de modèle complet du monde et fonctionne avec des données expérimentales, comme les méthodes de Monte Carlo. Il exploite les estimations de récompense en cours d'évaluation d'une politique, comme la Programmation Dynamique.

2.2.3 Évolution

Les méthodes évolutionnistes sont inspirées de la sélection naturelle des espèces de Darwin. Contrairement aux approches précédentes qui ne travaillent que sur un seul élément que l'on cherche à améliorer, l'évolution travaille en parallèle sur un ensemble, une population d'éléments qui sont soumis à une pression sélective favorisant la “survie” des plus performants.

Dans la section suivante, nous reviendrons plus en détail sur le fonctionnement des algorithmes évolutionnistes qui sont au cœur des travaux décrits dans ce mémoire.

Chapitre 3

Algorithmes évolutionnistes

3.1 Principes généraux

3.1.1 Objectif

Les algorithmes évolutionnistes (AE) sont des méthodes d'optimisation automatique qui explorent un espace de recherche potentiellement très vaste. Contrairement à d'autres algorithmes partant d'une solution singulière et cherchant à remonter un gradient de performances, comme les algorithmes de recuit simulé (Kirkpatrick *et al.*, 1983), par exemple, les AE utilisent un ensemble de solutions dont seule la performance ponctuelle est utilisée. Aucune autre propriété mathématique n'est nécessaire. La continuité ou la dérivabilité ne sont pas utilisées. La répartition, initialement aléatoire, de ces solutions sur l'espace de recherche limite les risques de convergence prématurée vers des extrema locaux. Le champ des applications potentielles est très vaste compte tenu de la pauvreté des informations nécessaires et de la généricité des principes exploités. Des problèmes d'optimisation peuvent être traités par de tels algorithmes (Gen et Cheng, 1997), mais aussi des problèmes d'apprentissage (Whitley et Hanson, 1989; Whitley, 1989; Durand *et al.*, 2000), ou même de conception de contrôleurs de robot (Meyer, 1998; Meyer *et al.*, 1998).

Le fait de travailler sur un ensemble de solutions améliorées progressivement, plutôt que de n'en considérer qu'une, est particulièrement adapté aux problèmes multi-critères dans lesquels des compromis doivent être trouvés. Là où un algorithme d'apprentissage autre ne donnera qu'une seule solution à la fois sur un compromis imposé à l'avance, un AE va pouvoir matérialiser le front de Pareto (Ben-Tal, 1979), c'est-à-dire l'ensemble des solutions présentant les meilleurs compromis possibles. L'expérimentateur acquiert ainsi à la fois la connaissance de ce front ainsi que les solutions correspondantes (Fonseca et Fleming, 1995; Van Veldhuizen et Lamont, 1998; Deb *et al.*, 2000), ce qui lui permet de réaliser ses choix sans avoir à relancer son algorithme d'optimisation à de multiples reprises.

Les AE sont également très adaptés à des environnements dynamiques, car ils disposent d'un réservoir d'individus différents, dont certains répondent peut-être mieux aux nouvelles exigences

(Schultz et Grefenstette, 2000). Un individu adapté à un certain contexte peut se révéler inefficace en cas de changement des conditions. Cependant, un autre individu de la population, auparavant moins performant, peut se révéler meilleur dans le nouvel environnement, ou être le point de départ d'une lignée qui conduira à des individus plus efficaces. La diversité maintenue dans la population permet ainsi de répondre rapidement à des changements de conditions. Ce type de fonctionnement nécessite une utilisation continue de l'AE. Plusieurs modalités sont possibles. Soit un seul robot physique existe et il faut alors partager le temps alloué à chaque individu pour le contrôle du robot. Une stratégie utilisable dans ce cas consiste à exploiter le meilleur individu en permanence et à ne relancer l'AE que lorsque des changements surgissent et que cet individu n'est plus aussi performant (Nordin et Banzhaf, 1997). Soit plusieurs robots peuvent fonctionner en même temps et, dans ce cas, les individus peuvent former un écosystème, dans lequel ils coexistent et se reproduisent (Nehmzow, 2002). Cela correspond mieux à la réalité naturelle, mais nécessite la coexistence d'un grand nombre de robots, ce qui n'est pas toujours applicable.

3.1.2 Inspiration

Les AE s'inspirent de manière très libre de la théorie darwinienne de l'évolution des espèces. Ces algorithmes travaillent sur une population d'individus qui contiennent l'information décrivant la solution qu'ils représentent. Par référence à la biologie, cette information est désignée par le terme de *génotype* alors que la solution qu'ils représentent - autrement dit le système cherché - est nommée *phénotype*. Le génotype est l'information manipulée par l'AE, alors que le phénotype en est la traduction après développement et constitue le système dont on cherche à optimiser ou concevoir le comportement. Dans cet objectif, les individus subissent un processus de sélection dont les critères sont définis par le concepteur. Génération après génération, seuls les plus performants survivent et se reproduisent dans le but de générer des descendants plus performants que leurs parents.

L'objectif de la version artificielle de l'évolution est de maximiser une fonction d'adaptation (appelée également fonction de *fitness*). Cette fonction d'adaptation quantifie l'efficacité des solutions générées en vue de résoudre un problème donné et constitue le principal biais introduit par le concepteur pour guider l'évolution vers les solutions recherchées. Sans chercher à reproduire, ni même à comprendre le fonctionnement de la sélection naturelle, le but est de disposer d'une méthode efficace de conception et d'optimisation de structures pour répondre à des problèmes particuliers, même si des interactions sont possibles et souhaitables avec des biologistes(Handley, 1995; Notredame *et al.*, 1997; Dandekar *et al.*, 2001).

3.1.3 Détails d'un algorithme évolutionniste

Dans cette section, nous allons détailler chacune des parties d'un algorithme évolutionniste, dont la figure 3.1 illustre les principes.

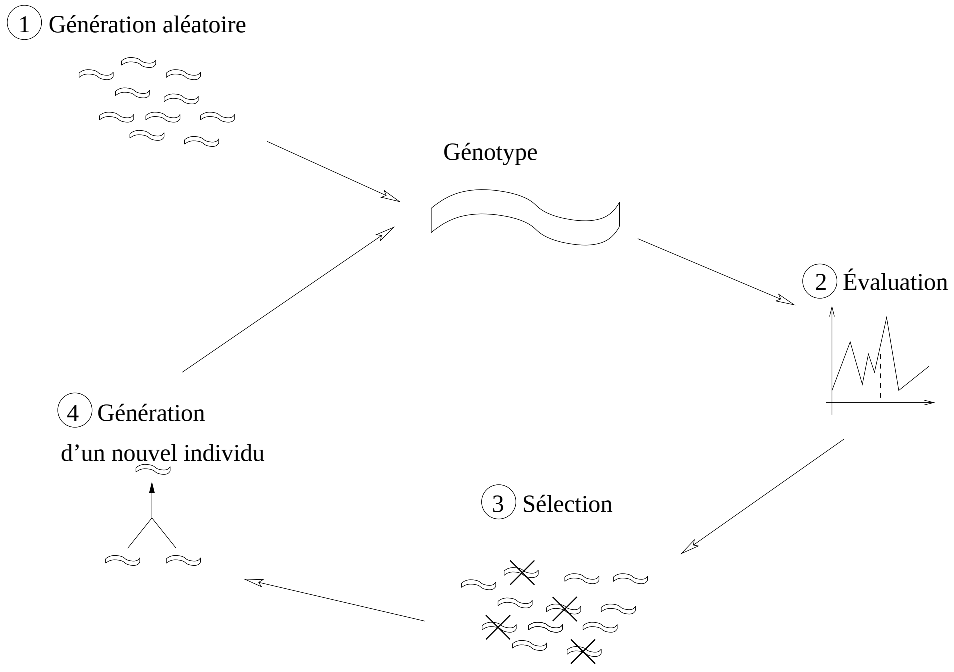


FIG. *3.1: Principe des algorithmes évolutionnistes. 1. Génération aléatoire de la population de génotypes. 2. Évaluation d'un génotype. 3. Sélection des meilleurs. 4. Génération d'un nouvel individu : copie d'un individu performant et application des opérateurs génétiques de mutation et de croisement.*

Les AE travaillent sur une population d'individus, qui représentent les solutions potentielles au problème posé. Ils procèdent par générations successives selon la loi :

$$P(g+1) = S.E.V(P(g))$$

$P(g)$ étant la population à la génération g. $V(.)$ est l'opérateur de variation, qui va générer de nouveaux individus à partir des individus de la population courante. $E(.)$ est l'opérateur d'évaluation qui va affecter une note à chaque individu. Cette note est utilisée par l'opérateur de sélection $S(.)$, qui va retenir les individus les plus performants au détriment des autres.

Évaluation

L'évaluation consiste à associer à un individu une note fournie par la fonction d'évaluation.

L'évaluation d'un génotype nécessite une étape de traduction appelée *développement*, pendant laquelle le génotype est traduit en un phénotype décrivant un agent, soit entièrement, soit en partie. Le comportement de l'agent est évalué et une note lui est attribuée selon des critères préétablis dépendants du problème à résoudre. La notation peut être très directive et porter sur chacun des comportements élémentaires ou, au contraire, laisser plus de latitude à l'évolution en n'évaluant que le comportement global.

La notation est un élément capital des AE. Elle est également le principal biais introduit par le concepteur. En effet, elle est la base du processus de sélection qui assure la survie des plus performants et leur utilisation pour la génération de nouveaux individus. Une notation très directive va imposer des étapes intermédiaires très précises, alors qu'une notation plus implicite ne s'intéressera qu'au résultat final.

Une notation précise laisse moins de latitude à l'évolution, mais permet d'avoir un contrôle plus important sur les solutions qui seront générées. Elle permet de guider l'évolution en lui indiquant quelle stratégie elle doit utiliser pour résoudre le problème. Une telle notation peut être utilisée lorsque la solution souhaitée doit exhiber un comportement connu précisément et pas un autre. Pour faire marcher un robot hexapode, une notation explicite prendra en compte, par exemple, les mouvements des pattes : leur rythme, leur amplitude, leur synchronisation.

Une notation imprécise laisse plus de liberté à l'évolution. Sur le problème de la marche d'un robot hexapode, cela reviendra à noter uniquement la distance parcourue pendant un temps donné et ce sera à l'évolution de trouver la technique la plus efficace. Il s'agit donc d'une description de plus haut niveau pour laquelle on ne s'intéresse qu'au résultat sans donner de consignes particulières sur la façon de l'atteindre. Ce type de notation peut permettre de trouver des solutions originales au problème posé, car aucune contrainte n'est imposée sur la méthode à employer. Kodjabachian et Meyer (1997) ont ainsi trouvé un rythme de marche très efficace pour un robot hexapode - le rythme tripode, utilisé par les insectes - sans avoir spécifié à l'évolution les caractéristiques d'un tel rythme.

Cependant, sur une tâche trop complexe, se pose le problème de l'amorçage (bootstrapping). Si aucun individu de la population initiale ne parvient à résoudre, ne serait-ce que partiellement, le problème, l'algorithme peut rester bloqué dans un minimum local. Si on demande directement à l'évolution de générer un contrôleur capable de faire survivre un oiseau artificiel dans son environnement, il est évident que l'algorithme ne convergera pas car résoudre cette tâche nécessite de résoudre de nombreux problèmes à la fois : maîtrise du vol, recherche de nourriture, évitement de prédateurs... La solution, pour éviter un tel blocage, consiste à guider l'évolution en lui fournissant des étapes intermédiaires plus simples, dans une approche dite *incrémentale* (Urzelai *et al.*, 1998) : on décompose le problème et on utilise l'évolution pour résoudre tout d'abord un premier problème simple que l'on complète petit à petit jusqu'à atteindre l'objectif final.

Sélection

L'algorithme de sélection opère sur l'ensemble des individus générés, ainsi que sur la population de la génération courante, pour retenir un échantillon des individus les plus performants pour créer la nouvelle population.

Il doit assurer la convergence vers une solution efficace, tout en maintenant la diversité de la population, aspect caractéristique des AE par rapport aux autres algorithmes d'apprentissage ou d'optimisation. La diversité dans la population diminue les risques de convergence prématurée vers un extremum local et assure un taux de réponse plus rapide en cas de modification de l'environnement pendant la durée de l'évolution.

Il y a plusieurs façons d'exercer une pression sélective sur la population. La plus radicale consiste à ne conserver que les meilleurs individus, mais les risques de perte de la diversité étant importants avec ce type de stratégie, des stratégies plus souples ont été développées. L'algorithme de *la roue de la fortune*, par exemple, affecte à chaque individu une probabilité d'être conservé qui est proportionnelle à ses performances. Le meilleur a ainsi de grandes chances d'être conservé, mais des individus de performance plus faible peuvent également se maintenir d'une génération à l'autre, ce qui peut aider à conserver la diversité de la population. Pour plus de détails sur les différents algorithmes de sélection, voir par exemple (Goldberg, 1989).

Variations : Génération d'un nouvel individu

$$\begin{aligned} p(g) &= V(p_1(g), p_2(g), p_3(g), ..., p_n(g)), (p_1(g), p_2(g), p_3(g), ..., p_n(g)) \in P(g)^n \\ &= M.C(p_1(g), p_2(g), p_3(g), ..., p_n(g)) \end{aligned}$$

$C(p_1(g), p_2(g), p_3(g), ..., p_n(g))$ est l'opérateur de croisement qui génère un nouvel individu à partir d'un ensemble d'individus existants[22], sélectionnés selon leurs performances. $M(p(g))$ est l'opérateur de mutation qui modifie aléatoirement un individu.

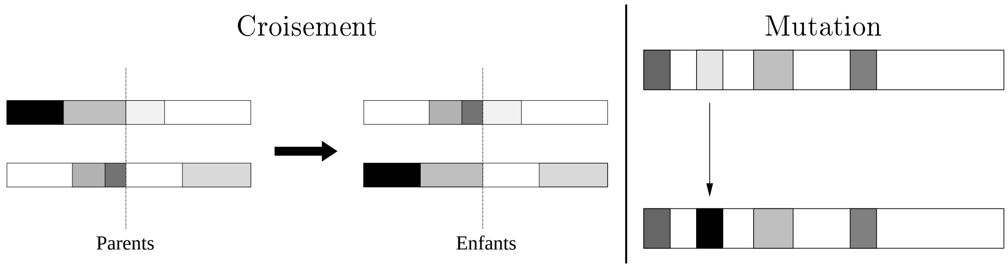


FIG. *3.2: Opérateurs génétiques*

Le *croisement* échange du matériel génétique entre plusieurs individus. Cet opérateur permet d'explorer la zone de l'espace de recherche qui est intermédiaire entre les différents parents. Il doit échanger des blocs fonctionnels entre individus.

La *mutation* modifie légèrement l'individu de façon aléatoire, explorant ainsi l'espace de recherche dans la zone voisine de l'individu de départ, tout en conservant la possibilité - statistiquement peu fréquente - de modifications importantes, qui peuvent aider à sortir d'un extremum local. Ces deux opérateurs sont appliqués avec des probabilités variables selon les algorithmes. Ces probabilités ne sont, d'ailleurs, pas toujours fixées à l'avance. Ainsi, dans les Stratégies Évolutionnistes (Bäck et Schwefel, 1995; Schwefel et Bäck, 1995), l'évolution optimise également le taux de mutation.

Les deux opérateurs ne sont pas toujours utilisés simultanément, certains algorithmes n'en utilisant qu'un seul. Les Stratégies Évolutionnistes, par exemple, n'utilisent, dans leurs premières versions, que l'opérateur de mutation (Schwefel, 1977).

[22]Généralement, cet ensemble est de taille égale à 2.

La génération d'un nouvel individu est la seule phase exploratoire de l'algorithme, c'est le seul moment pendant lequel une solution est modifiée. Les individus n'apprennent pas pendant leur durée de vie : leur comportement est entièrement déterminé par leur génotype et il n'y a pas de phase d'apprentissage intermédiaire[23].

Génération initiale d'une population

À l'initialisation de l'algorithme, il est nécessaire de générer la population complète qui sera utilisée ensuite.

La possibilité la plus couramment utilisée consiste à générer aléatoirement des individus. Cependant, il est possible de partir d'une population dont les individus ont été, au moins partiellement, préparés manuellement en incluant des connaissances a priori, de façon à restreindre l'espace de recherche et à accélérer la convergence.

3.1.4 Le génotype : le problème du codage de l'information

Le génotype représente l'information manipulée par l'AE au travers des opérateurs génétiques : c'est la représentation d'une solution. C'est un point crucial des AE : comment représenter une solution potentielle ? De ce choix découle directement l'espace de recherche qui, selon les circonstances, peut être petit ou de très grande taille, ce qui va jouer directement sur le type de solutions trouvées, ainsi que sur le temps de convergence.

Le génotype peut être très simple : une chaîne d'éléments binaires dans les Algorithmes Génétiques de Goldberg et Samtani (1986), ou bien beaucoup plus complexe et structuré comme les arbres de développement de réseaux de neurones de Gruau (Gruau, 1994). Il sera choisi en fonction de la tâche à résoudre et de ses propriétés dépendront, pour beaucoup, les performances de l'algorithme évolutionniste.

Pour optimiser des paramètres, une représentation génétique directe de ceux-ci suffit. Cependant, lorsque le génotype représente une structure complexe que l'évolution doit manipuler, le choix du codage devient un problème fondamental : comment représenter une structure complexe d'une façon aussi compacte et robuste que possible, tout en facilitant et en encourageant l'apparition de nouveaux individus viables ? Un génotype ne pourra pas être constamment un plan complet et détaillé d'un système cognitif : cela nécessiterait une quantité d'informations croissant de manière exponentielle avec la complexité du contrôleur. Cependant, un codage compact peut compliquer l'exploration de l'espace de recherche : si la moindre modification du génotype entraine un changement complet dans la structure du phénotype, la découverte d'une solution en deviendra d'autant plus difficile et reposera pour beaucoup sur le hasard, ce qui rendra la convergence quasiment impossible. Les changements de génotype doivent aussi créer de préférence des systèmes viables, des solutions acceptables, ce qui n'est pas évident dès que l'on s'intéresse à

[23]De nombreux travaux s'orientent cependant dans cette voie (Gruau et Whitley, 1993; Floreano et Urzelai, 2001; Husken et Igel, 2002).

des systèmes dynamiques complexes. Autrement dit, le codage et la relation génotype-phénotype doivent suivre une certaine continuité de façon à exploiter au mieux les propriétés des AE.

Les systèmes dynamiques générés à l'heure actuelle avec des AE, dont la structure est la plus riche mais aussi la plus complexe, sont des réseaux de neurones artificiels. Des codages très variés ont été employés jusqu'à présent pour faire évoluer la structure et les paramètres de réseaux de neurones quelconques. L'approche la plus simple consiste à utiliser une représentation directe du réseau, comme par exemple dans (Whitley *et al.*, 1990; Peter Angeline *et al.*, 1994), mais un tel codage ne permet pas facilement de passer à des réseaux plus complexes. En effet, le génotype et le phénotype sont reliés par une bijection, aussi la complexité du génotype croit linéairement avec la complexité du phénotype. Pour répondre à ce problème, des codages indirects ont été développés (Kodjabachian et Meyer, 1995). Ils utilisent des règles de réécriture (Kitano, 1990a), des programmes de développement (Gruau, 1994; Kodjabachian, 1998) ou encore des processus de neurogénèse (Harvey, 1993; Michel, 1999; Astor, 1998). Nous reviendrons plus en détails sur les différentes possibilités de faire évoluer des réseaux de neurones au chapitre 4, qui est entièrement consacré à cette question.

3.1.5 Description des principaux AE

Plusieurs éléments varient d'un AE à l'autre : l'algorithme de sélection, le codage des solutions et enfin les opérateurs génétiques utilisés. Nous allons présenter les deux principaux algorithmes évolutionnistes : les Algorithmes Génétiques ainsi que les Stratégies Évolutionnistes.

Algorithmes Génétiques

Les Algorithmes Génétiques (AG) de Holland (1975) utilisent un codage composé d'éléments binaires. Le génotype est un ensemble ordonné de bits qui peuvent être convertis en nombres réels ou en un système cognitif plus complexe, comme un réseau de neurones par exemple, selon l'étape de développement. Plusieurs possibilités de croisement existent. La plus simple consiste à choisir un emplacement dans l'ensemble ordonné des élements binaires et à échanger les sous-chaînes situées avant et après ce point (cas représenté sur la figure 3.2) : c'est le croisement à un point. Il existe également des croisements à plusieurs points. Dans ce cas, on commence par recopier un des parents et, chaque fois que l'on atteint un point, on continue la copie en prenant en compte le génotype de l'autre parent :

$$\begin{array}{ccc} 100|0100111|01010 & & 100|1101010|01010 \\ & \rightarrow & \\ 010|1101010|00111 & & 010|0100111|00111 \end{array}$$

Les mutations consistent à inverser un ou plusieurs bits de la chaine binaire. Holland a défini plusieurs algorithmes de sélection. La note d'un individu ou son rang peuvent être utilisés directement pour déterminer sa probabilité de générer de nouveaux individus. Une autre méthode, plus radicale, consiste à ne garder d'une génération à l'autre qu'un certain pourcentage de la population, qui survivra et servira de géniteurs pour complèter la nouvelle population, c'est l'algorithme *"steady state"*. (Goldberg, 1989) contient une description complète des AG ainsi qu'un guide et des exemples permettant de les implémenter facilement.

Stratégies évolutionnistes

Les Stratégies Évolutionnistes (Bäck et Schwefel, 1995; Schwefel et Bäck, 1995) utilisent directement un vecteur de nombres réels $\boldsymbol{x}$. Le croisement n'est pas utilisé dans les premières versions (version à "deux membres"), mais les variantes plus élaborées ("multi-membres") définissent un opérateur de ce type (Bäck *et al.*, 1991). Les mutations sont des modifications aléatoires gaussiennes des éléments du vecteur. Chaque élément x_i est muté en lui ajoutant un nombre aléatoire Gaussien d'écart-type σ_i. Le vecteur des σ_i est également soumis à évolution alors que, pour les Algorithmes Génétiques, les paramètres de mutation sont fixes. Les corrélations possibles entre les éléments d'un même vecteur sont également prises en compte de façon automatique par l'algorithme, grâce à l'utilisation d'une matrice de covariance également soumise à évolution. Le croisement consiste à sélectionner plusieurs parents et à construire progressivement les vecteurs $\boldsymbol{x}$ et $\boldsymbol{\sigma}$ en choisissant aléatoirement, pour chaque élément, le parent à partir duquel on va le recopier :

$$
\begin{aligned}
x'_i &= \begin{cases} x_{a,i}, & \chi \leq 1/2 \\ x_{b,i}, & \chi > 1/2 \end{cases} \qquad \forall i \epsilon \{1, \ldots, n\} \\
\sigma'_i &= \begin{cases} \sigma_{a,i}, & \chi \leq 1/2 \\ \sigma_{b,i}, & \chi > 1/2 \end{cases} \qquad \forall i \epsilon \{1, \ldots, n\}
\end{aligned}
$$

La création d'une nouvelle population consiste à générer λ individus à partir de μ parents et à ne conserver que les μ meilleurs, soit parmi les descendants uniquement (il est alors nécessaire que λ soit supérieur à μ, c'est la stratégie (λ, μ)), soit parmi les descendants ainsi que leurs parents (stratégie $(\lambda + \mu)$).

3.2 L'évolution comme méthode de conception

3.2.1 Avantages

Les méthodes évolutionnistes disposent de plusieurs avantages sur les autres méthodes d'optimisation. Tout d'abord, le retour d'information requis est très faible. En effet, le seul retour nécessaire est la note finale attribuée aux performances d'une solution potentielle. Cette note peut-être très qualitative et ne contenir aucune information concernant les moyens disponibles pour résoudre la tâche imposée, comme nous l'avons vu précédemment.

Comme beaucoup d'autres algorithmes, l'évolution dispose de paramètres qui doivent être positionnés convenablement pour que l'algorithme fonctionne avec toute son efficacité. Cependant, l'évolution, contrairement aux autres méthodes, peut optimiser ces paramètres en cours de fonctionnement (Hinterding *et al.*, 1997).

La fonction que l'évolution cherche à optimiser n'a pas besoin de propriétés mathématiques particulières. Il suffit de pouvoir connaître sa valeur en un point, nul besoin de dérivée, par exemple, contrairement aux approches basées sur la remontée de gradient. Cela rend ces méthodes plus souples et applicables à des problèmes sur lesquels on ne dispose que de peu de connaissances. En effet, il n'est pas besoin de disposer d'une mise en équation complète du système, il suffit de pouvoir classer les solutions proposées par l'évolution.

Il n'est pas nécessaire non plus de connaître la solution idéale ou même simplement des exemples, contrairement aux algorithmes supervisés basés sur la rétropropagation de l'erreur.

Les méthodes évolutionnistes n'ont donc besoin que de peu d'informations, l'algorithme en soi ne reposant pas sur des propriétés particulières à un problème. Ce faible retour d'information nécessaire rend les méthodes évolutionnistes utilisables dans des applications très variées, qu'elles soient dans le domaine de l'ingénierie - conception d'armatures métalliques pour l'industrie (Michalewicz *et al.*, 1996) - ou dans le domaine des Systèmes Adaptatifs ou de la Vie Artificielle - conception de la morphologie d'une créature virtuelle et de son système de contrôle (Sims, 1994) - ou plus généralement robotique évolutionniste (Gomi et Griffith, 1996; Meyer, 1998; Meyer *et al.*, 1998; Nolfi et Floreano, 2001; Guillot et Meyer, 2001; et al., 2003).

3.2.2 Inconvénients

L'inconvénient principal découle directement des points cités ci-dessus : le retour d'information étant relativement faible et l'algorithme étant très général, les mécanismes à base d'évolution seront souvent plus lents à converger que les algorithmes optimisés pour un problème particulier qui vont exploiter au maximum les connaissances disponibles pour restreindre la recherche aux seules solutions pertinentes. Kitano a montré qu'un Algorithme Génétique était plus lent qu'un algorithme de rétropropagation pour apprendre les poids d'un réseau de type feed-forward à trois couches (Kitano, 1990b) sur un problème particulier. Cependant, dans plusieurs cas, ils se sont révélés plus efficaces que d'autres algorithmes classiques d'apprentissage (Montana et Davis, 1989; Radcliffe, 1990; Whitley *et al.*, 1990).

L'évolution est une méthode de recherche par essais et erreurs. Elle ne pourra pas rivaliser avec des méthodes classiques sur des problèmes dont on connait directement l'expression de la solution. A priori, l'évolution ne pourra donc pas rivaliser avec des méthodes conventionnelles de conception de contrôleurs dans leur zone de fonctionnement habituelle. En revanche, elle deviendra intéressante lorsque l'on s'éloignera du domaine de fonctionnement de ces méthodes ou lorsque le fonctionnement du système ne sera pas connu avec précision.

Un autre défaut des AE est leur opportunisme. En effet, ces algorithmes vont exploiter des caractéristiques auxquelles on n'aurait pas forcément pensé. Si c'est parfois un avantage, cela peut également être un inconvénient, en particulier lorsque l'évolution exploite des caractéristiques peu robustes ou encore des failles de la fonction de fitness. Dans ce contexte il est indispensable de

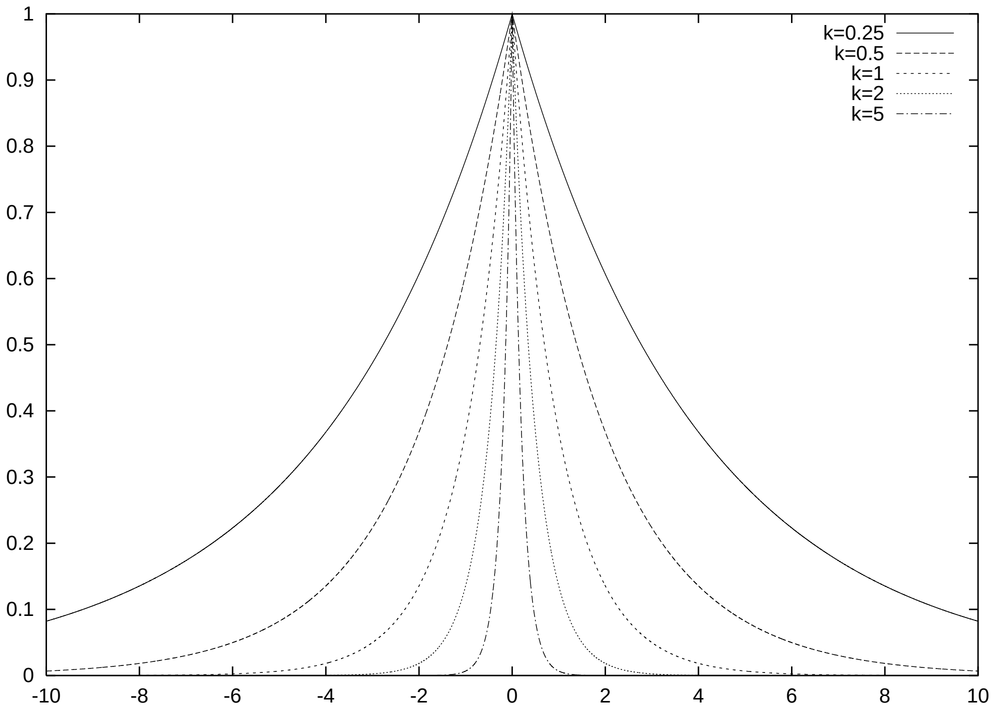


FIG. *3.3: Fonction f(x) à optimiser en dimension n=1 avec différentes valeurs de k.*

toujours vérifier le comportement généré et de ne pas se contenter de l'indicateur de performance donné par la fitness, qui se révèle parfois très éloigné de ce que l'on aurait effectivement attendu.

3.2.3 Influence de l'espace de recherche

Les méthodes d'apprentissage et d'optimisation dépendent fortement de deux paramètres : la taille de l'espace de recherche et ses caractéristiques. De nombreux algorithmes reposent sur des propriétés particulières et ne sont pas toujours applicables, contrairement à l'évolution dont les principes sont très génériques. Ses performances n'en restent pas moins très dépendantes de ces paramètres. Pour le mettre en évidence, nous allons étudier les performances d'un algorithme évolutionniste pour optimiser une fonction de type :

$$f(x) = \Pi_{i=0}^{n} \exp^{-k|x_i|}$$

La figure 3.3 représente cette fonction pour différentes valeurs de k. Nous allons étudier les résultats obtenus en fonction de la valeur de k et de la dimension n. Nous avons utilisé une Stratégie Evolutionniste (Schwefel, 1981; Schwefel et Rudolph, 1995), mais le comportement serait similaire avec un Algorithme Génétique.

L'espace de recherche est constitué d'un seul et unique pic centré sur la solution idéale, qui est la valeur nulle. La largeur du pic dépend du paramètre k.

Les résultats de la table 3.4 montrent tout d'abord qu'avec un même paramètre k, lorsque la dimension augmente, le problème devient plus difficile à résoudre. En effet, la taille de l'espace

Dimension	k=0.25	k=0.5	k=1	k=2	k=5
n=1	0 (10%)	0.44 (10%)	1 (10%)	0.55 (10%)	2.44 (10%)
n=2	8.88 (10%)	16 (10%)	16.88 (10%)	27.66 (10%)	42.62 (20%)
n=5	75.5 (60%)	125.16 (40%)	166.5 (40%)	302.33 (40%)	317 (40%)
n=10	211 (90%)	nc (100%)	nc (100%)	nc (100%)	nc (100%)

FIG. *3.4: Optimisation de la fonction f. Nombre de génération requis en fonction de la dimension n et du coefficient k pour trouver la valeur optimisant la fonction (qui est la valeur nulle). L'intervalle de valeurs possible est [-10 ;10]. Les expériences sont arretées à la* 1000^{eme} *génération. Les résultats présentés sont une moyenne sur dix expériences. Les pourcentages cités correspondent aux expériences qui n'ont pas convergé à la 1000ème génération. "nc" correspond à des conditions dans lesquelles aucune expérience n'a convergé (taux d'échec de 100%).*

de recherche croît exponentiellement avec n et, plus l'espace de recherche est vaste, plus il est difficile de trouver la solution.

Il apparaît également que, à dimension constante, plus k est élevé, plus l'évolution a du mal à converger. Les valeurs élevées de k correspondent à des espaces de recherche très accidentés. Dans ce cas, compte tenu des arrondis, la solution optimale ne peut être trouvée que par accident. Résoudre un tel problème revient à trouver une aiguille dans une botte de foin. Les algorithmes évolutionnistes, comme tous les autres algorithmes dans ce cas, ne se comportent pas mieux (et, de toute façon, ne peuvent pas se comporter mieux) qu'une recherche aléatoire.

La taille de l'espace de recherche et ses caractéristiques dépendent pour beaucoup de la façon de poser le problème et des moyens mis à la disposition de l'algorithme d'apprentissage ou d'évolution. Pour résoudre un problème de contrôle de robot avec un réseau de neurones, par exemple, il n'est pas du tout équivalent d'imposer une structure (dont on sait qu'elle est suffisante) en ne faisant évoluer que les poids des connexions et de rechercher à la fois les poids et la structure du réseau. Dans ce dernier cas, l'espace de recherche est plus vaste et ses caractéristiques vont dépendre pour beaucoup de la façon de représenter la structure du réseau. Nous reviendrons ultérieurement sur cette question.

Les algorithmes évolutionnistes disposent dans ce contexte d'une grande souplesse par rapport à des algorithmes plus classiques. En effet, il est toujours possible d'introduire un biais qui restreindra l'espace de recherche en rendant la recherche plus directive. Pour cela, le concepteur dispose principalement de deux possibilités : modifier la représentation d'une solution et la façon d'explorer l'espace de recherche, ou modifier la fonction d'évaluation.

La représentation d'une solution peut refléter des caractéristiques inhérentes au problème que l'on souhaite traiter. Les symétries éventuelles du problème peuvent être exploitées pour restreindre l'espace de recherche (Kodjabachian et Meyer, 1997; Ijspeert et Kodjabachian, 1999). De manière générale, les performances de l'algorithme évolutionniste dépendront de la qualité du codage choisi pour le génotype. Si la solution doit être symétrique ou modulaire, le codage choisi devra, bien entendu, le permettre. De plus, le codage doit rendre l'espace de recherche le plus lisse possible. En effet, plus les solutions recherchées seront ponctuelles dans l'espace de recherche, plus elles seront difficiles à trouver comme nous venons de l'illustrer.

La fonction d'évaluation peut également aider l'évolution. Elle peut fournir des contraintes plus précises qu'une description de haut niveau du comportement souhaité, soit en donnant des détails de plus bas niveau et en récompensant des comportements élémentaires intermédiaires, soit en traitant le problème de façon incrémentale.

Si, sur des problèmes simples, l'utilisation de ce type de technique n'est pas nécessaire, sur des problèmes où l'espace de recherche est plus vaste, il peut devenir indispensable de guider un peu l'évolution pour que la convergence se fasse dans des temps raisonnables.

Le principal problème lié aux méthodes évolutionnistes réside donc dans la définition et les propriétés de l'espace de recherche que l'évolution doit explorer. Ce problème est le même pour toutes les méthodes d'adaptation et d'apprentissage, mais il se pose de façon beaucoup moins importante dans les autres méthodes, car elles sont généralement associées à un espace de recherche particulier que l'utilisateur a peu de moyens de modifier ; généralement, seuls quelques paramètres sont adaptables. Dans le cas des algorithmes évolutionnistes, l'espace de recherche dépend entièrement des choix de l'utilisateur. Selon le codage et la fonction d'adaptation choisie, l'espace de recherche peut être réduit ou vaste, “lisse” ou “accidenté”.

3.3 Évolution de contrôleurs

Nous allons à présent présenter l'utilisation de l'évolution pour concevoir et optimiser les contrôleurs que nous avons présentés dans la section 2.1.

3.3.1 Optimisation de paramètres

Les Algorithmes Génétiques ou les Stratégies Évolutionnistes permettent d'optimiser une fonction. Si on considère une fonction évaluant les performances d'un robot sur un problème donné, ces algorithmes peuvent donc être utilisés pour optimiser les paramètres d'un système de contrôle quelconque, dont la structure a été au préalable fixée manuellement. L'information génétique est alors un vecteur de nombres, qui subira l'effet des opérateurs génétiques de mutation et de croisement. L'évaluation d'un génotype G consistera à positionner tout d'abord les paramètres du système de contrôle du robot en utilisant G et à observer et noter ses évolutions dans différentes conditions. Ces conditions doivent impérativement constituer un échantillon représentatif des conditions que le système devra ensuite affronter. Dans le cas contraire, l'évolution risque de trouver des solutions spécialisées à des conditions particulières qui ne fonctionneront pas convenablement en dehors de ce champ de fonctionnement.

L'évolution peut ainsi être utilisée pour paramétrer des contrôleurs de type PID, problème souvent résolu de façon empirique, ou pour optimiser des contrôleurs plus spécifiques. Le contrôleur de marche du robot AIBO de Sony dans sa version commerciale a ainsi été optimisé grâce à des méthodes évolutionnistes (Hornby *et al.*, 2000).

3.3.2 Systèmes à base de règles

Si les systèmes à base de règles permettent d'inclure la connaissance d'experts dans un contrôleur automatique, leur définition manuelle se révèle rapidement fastidieuse. Aussi des méthodes basées sur des algorithmes évolutionnistes ont-elles été utilisées pour automatiser leur conception.

Les systèmes de classeurs peuvent être reliés à l'évolution de deux façons. Un ensemble de règle constituant un contrôleur peut être contenu dans un seul individu (approche Pittsburgh (Smith, 1980)), ou bien chaque règle peut constituer un génotype à part entière (approche Michigan (Goldberg, 1989)), le contrôleur étant alors la population dans son ensemble.

Des systèmes à base de règles floues de type TSK ont été conçus par évolution pour contrôler un hélicoptère (Hoffmann *et al.*, 1998). Ces règles associent une condition portant sur des ensembles flous à une fonction linéaire d'une ou plusieurs variables. L'algorithme procède par étapes depuis un contrôleur composé d'une seule règle toujours valable avec un seul terme dans la partie conclusion, vers un système plus élaboré par divisions successives de l'espace des variables et/ou par ajouts de termes dans la partie conclusion. Les coefficients de la partie conclusion sont évolués en utilisant une Stratégie Évolutionniste. Les performances obtenues avec ce type de contrôleur ont été comparées à celles d'autres contrôleurs (Shim *et al.*, 1998). Il s'est avéré que le contrôleur à base de règles floues avait des performances similaires à ceux qui ont été conçus en utilisant des méthodes de contrôle robuste : ils avaient une grande robustesse à l'incertitude et au bruit, mais leur domaine de fonctionnement restait relativement restreint autour du vol stationnaire. Le contrôle non-linéaire donnait des contrôleurs utilisables sur un domaine de fonctionnement plus vaste, mais nécessitait une connaissance beaucoup plus précise du système et était plus sensible aux perturbations. Cependant, à chaque domaine de vol particulier peut correspondre un ensemble de règles de contrôle adaptées. Aussi, si la subdivision du domaine de vol est correcte et si les règles correspondantes sont bien configurées, un contrôleur à base de règles devrait théoriquement pouvoir se comporter de façon efficace sur un grand domaine de fonctionnement. Cependant, partitionner le domaine de vol est un problème complexe avec un espace de recherche très important, ce qui rend sa découverte délicate sans utiliser d'heuristiques ou de connaissances a priori. Cela explique sans doute les faibles performances de l'algorithme d'Hoffmann, Koo et Shakernia (Hoffmann *et al.*, 1998) sur ce point.

3.3.3 Programmes

Automates à états finis

Fogel, un des pionniers des approches évolutionnistes, a fait évoluer des automates à états finis (Fogel et Walsh, 1966) pour apprendre à prévoir le prochain symbole dans une séquence fournie en entrée du système.

Landau et Picault ont utilisé un algorithme génétique pour faire évoluer des automates à états finis (Landau *et al.*, 2001b). Un vecteur de valeurs binaires est soumis à évolution et subit ensuite une étape de développement pendant laquelle un automate va être généré, puis utilisé, pour contrôler un agent. Le développement consiste à traduire les éléments de la chaîne binaire en jetons qui sont des instructions de développement, empilées immédiatement après création. Après traduction, les instructions empilées sont exécutées pour obtenir l'automate. Elles peuvent manipuler la pile, ou créer une connexion ou un nœud. Les paramètres dont elles ont éventuellement besoin sont pris dans la pile et dépilés après usage.

Brave (Brave, 1996) a utilisé la programmation génétique, que nous présenterons à la section suivante, pour le développement d'automates à états finis.

Programmation fonctionnelle

Koza (1992) a fait évoluer des programmes en utilisant un formalisme fonctionnel basé sur des arbres[24]. Les programmes manipulés sont des fonctions dont l'enchaînement, ainsi que l'éventuel paramétrage, sont définis par évolution. L'expérimentateur doit fournir l'ensemble des *symboles terminaux* susceptibles d'être utilisés, ainsi que l'ensemble des *fonctions primitives* dont le programme sera composé. Les symboles terminaux sont les variables sur lesquelles porteront les calculs et les fonctions primitives sont les éléments de base permettant de construire les programmes. Pour résoudre des problèmes d'approximation de fonctions de type $z = f(x, y)$, par exemple, l'ensemble des symboles terminaux sera $\{x, y\}$ et l'ensemble des fonctions primitives pourra être $\{+, -, \times, /\}$. Le type de fonction obtenu sera de la forme (expression LISP) :

$$(/\ (\times\ (+\ x\ 4,1)\ (-\ y\ 5,2))\ (\times\ (-\ x\ 1,3)\ (+\ y\ 7,1)))$$

ce qui correspond à la fonction suivante :

$$\frac{(x + 4,1) \times (y - 5,2)}{(x - 1,3) \times (y + 7,1)}$$

La figure 3.5 représente cette fonction sous la forme d'un arbre.

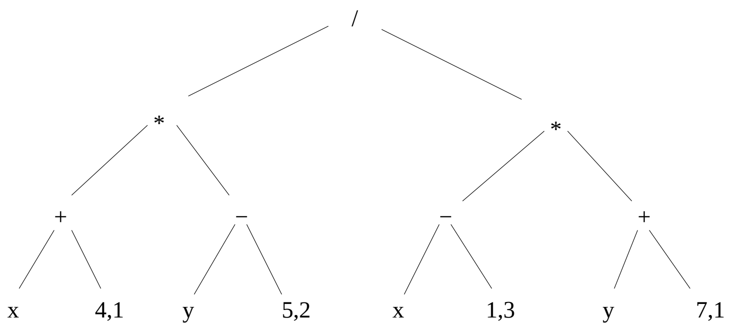


FIG. *3.5: Exemple de représentation de programme sous forme d'arbre.*

Les programmes manipulés dans cette version simple de la programmation génétique ne permettent pas de réutiliser des fonctions à plusieurs endroits dans le programme, une possibilité couramment utilisée en programmation classique. Pour remédier à ce manque, Koza (1994) a

[24] Il a utilisé pour cela le langage fonctionnel LISP.

ajouté la possibilité de définir automatiquement des fonctions susceptibles d'être appelées par le programme principal. Ces fonctions, appelées ADF pour *Automatically Defined Functions*, sont également soumises à évolution. Le programme est alors composé d'une ou plusieurs de ces fonctions auxquelles s'ajoute un programme principal, appelant ou non ces fonctions.

Les populations utilisées sont initialisées avec des milliers, ou même des millions, d'individus.

Le croisement permet d'échanger des sous-arbres entre programmes. Le croisement des programmes ci-dessous selon les marques indiquées :

$$(/\ (\times\ (+\ x\ 4,1)\ (-\ y\ 5,2))\ \downarrow (\times\ (-\ x\ 1,3)\ (+\ y\ 7,1))\ \downarrow)$$

$$(\times\ (/\ \downarrow (-\ x\ 3,2) \downarrow\ (\times\ y\ 9,5))$$

donne le résultat suivant :

$$(\times\ (\ /\ (\times\ (-\ x\ 1,3)\ (+\ y\ 7,1))\ (\times\ y\ 9,5))$$

La mutation remplace un sous-arbre par un autre sous-arbre généré aléatoirement.

L'utilisation des ADF nécessite l'usage de six nouveaux opérateurs qui permettent la duplication, l'ajout ou la suppression de fonctions ou de paramètres de fonctions.

3.3.4 Réseaux de neurones

Les travaux étudiant les liens possibles entre évolution et réseaux de neurones, forment un domaine très actif (Meyer *et al.*, 1998; Yao, 1999). L'évolution peut être utilisée à différents niveaux dans la conception d'un réseau de neurones. Le chapitre suivant est entièrement consacré à la présentation des recherches sur cette question ainsi qu'aux problématiques sous-jacentes.

Chapitre 4

Évolution de réseaux de neurones

Nous allons présenter dans ce chapitre les différentes approches possibles pour concevoir la structure de réseaux de neurones avec des algorithmes évolutionnistes. Nous pointerons ensuite ce qui constitue pour nous les principaux problèmes liés à la conception de réseaux de neurones en général, ainsi que les difficultés qui sont plus liées à l'utilisation d'algorithmes évolutionnistes.

4.1 Optimisation des poids uniquement

Une première possibilité pour optimiser des réseaux de neurones consiste à figer la structure d'un réseau et à faire évoluer les poids uniquement dans une approche semblable à celle décrite dans la section 3.3.1.

La structure du réseau est déterminée à partir d'heuristiques et l'évolution se charge ensuite de trouver les poids adaptés (Whitley et Hanson, 1989; Whitley, 1989; Durand *et al.*, 2000). D'autres algorithmes, tels la rétropropagation de gradient, peuvent être utilisés dans ce contexte. Cependant, ils nécessitent de connaître la solution idéale de façon à calculer l'erreur, information qui n'est pas toujours disponible.

Les biologistes construisent des modèles de systèmes nerveux issus de leurs observations. Cependant, si la structure est identifiable, le paramétrage du modèle ne l'est pas. L'évolution peut alors fournir une réponse efficace et permettre aux chercheurs d'optimiser un réseau dont ils connaissent globalement la structure, mais dont les paramètres sont hors de portée. Ijspeert, Hallam et Willshaw ont utilisé l'évolution pour étudier un modèle neuronal de locomotion de lamproie dont les paramètres avaient été initialement positionnés manuellement (Ijspeert *et al.*, 1999). Ils ont montré que l'évolution peut aider à la conception de modèles biologiques en profitant des connaissances disponibles et en laissant à l'évolution le soin de trouver les paramètres difficilement observables.

4.2 Conception de la structure et des paramètres du réseau

La conception de la structure d'un réseau de neurones demeure un problème ouvert. La structure la plus utilisée est le perceptron, le nombre de neurones sur la couche cachée étant généralement choisi approximativement en fonction de la quantité d'informations que le réseau doit mémoriser. Si l'apprentissage n'aboutit pas, on recommence en ajoutant des neurones sur la couche cachée... Les algorithmes évolutionnistes peuvent apporter une solution à ce problème en travaillant à la fois sur la structure et les paramètres du réseau, que ce soit sur un perceptron ou sur des structures récurrentes plus complexes.

Si pour optimiser des paramètres, la représentation est relativement directe - un vecteur de nombres réels ou des chaînes binaires directement traduisibles en réels - la conception d'un réseau de neurones nécessite un codage ou une étape de développement plus complexes.

Pour être efficace, un codage nécessite plusieurs propriétés énoncées par Gruau (1992) :

- **la complétude :** tout réseau doit pouvoir être codé par un génome,
- **la fermeture :** tout génome doit représenter un réseau,
- **la modularité :** un sous-réseau présent en plusieurs exemplaires dans le réseau complet doit pouvoir n'apparaître qu'une seule fois dans le génome.

Propriétés auxquelles on peut ajouter les suivantes :

- **la dynamique :** les dimensions de l'espace de recherche ne doivent pas être imposées, le codage doit permettre une recherche incrémentale,
- **la compatibilité avec les opérateurs génétiques** de mutation et de croisement,
- la possibilité de spécifier des **entrées et des sorties** au réseau (vers capteurs et effecteurs).

Un des principaux problèmes réside dans l'efficacité des opérateurs génétiques. La mutation doit effectuer une recherche préférentiellement locale, qui permette d'explorer l'espace de recherche au voisinage d'une solution. Cependant, d'importantes modifications doivent rester possibles - et n'intervenir qu'avec de faibles probabilités - pour pouvoir sortir d'extrema locaux. Une telle définition n'impose que peu de contraintes sur le codage et s'avère généralement facile à implémenter sur un codage particulier. L'opérateur qui pose le plus de problèmes est le croisement. Il doit, théoriquement, permettre l'échange de blocs fonctionnels significatifs. Le résultat du croisement de deux individus doit être susceptible de contenir des caractéristiques de chacun de ses parents. Cette propriété n'est pas facilement implémentable sur tous les types de codage. Elle va imposer d'importantes contraintes qui doivent être prises en compte lors de la conception du codage. Nous reviendrons plus loin sur ce point.

Deux types de codage sont possibles : un codage direct et un codage indirect. Un codage direct définit une représentation simple du réseau - une liste de neurones et une liste de connexions inter-neurones. La correspondance entre le génotype et le phénotype est alors bijective : chaque neurone et chaque connexion vont être décrits de façon unique et indépendante dans le génome. Un codage indirect est une représentation plus compacte du réseau nécessitant une étape de développement, qui va traduire l'information contenue dans le génotype en un phénotype. La correspondance entre génotype et phénotype ne sera alors plus bijective. Une partie de génome pourra décrire plusieurs neurones ou plusieurs connexions[25].

4.2.1 Codage direct

Les tableaux 4.1 et 4.2 résument les caractéristiques des méthodes utilisant un codage direct. Dans la suite de la section, nous allons revenir sur les réponses apportées par ces méthodes aux différents problèmes posés par l'évolution de structure. Tous ces problèmes tournent autour de la définition des opérateurs génétiques, point crucial des algorithmes évolutionnistes.

Une représentation génétique adaptée peut grandement faciliter la définition des opérateurs génétiques. Fullmer et Miikkulainen (1991) ont utilisé un codage constitué d'une suite d'entiers qui représentent les paramètres d'un nœud du réseau : un marqueur de début, une clé d'identification, la valeur de sortie initiale du réseau et une liste de connexions constituées de la clé d'identification du neurone-source et du poids de la connexion (figure 4.3). Le codage contient ensuite un marqueur de fin. Cette représentation facilite la définition des opérateurs génétiques, qui peuvent être identiques à ceux d'un algorithme génétique simple : les mutations changent un codon, le croisement échange des sous-chaînes de codons entre chromosomes.

Le croisement entre chromosomes doit permettre l'échange de blocs. L'idéal serait de pouvoir échanger des blocs fonctionnels de façon à ce que le résultat du croisement soit susceptible de profiter de caractéristiques différentes issues de chacun des parents, ce qui est loin d'être évident avec un codage direct. La simple définition d'un opérateur de croisement n'est pas non plus un problème simple, à tel point que, dans de nombreux cas, le croisement n'est même pas utilisé (Peter Angeline *et al.*, 1994; Pasemann et Dieckmann, 1997b; Yao et Liu, 1997). *PDGP* (Parallel Distributed Genetic Programming) (Pujol et Poli, 1997) utilise une représentation du réseau sous forme de grille, chaque neurone étant un nœud de la grille. Deux nœuds sont choisis et les liens, représentés sous forme relative (connexions au neurone de la ligne précédente (L-1), 2 colonnes après (C+2), par exemple), sont transposés d'un des nœuds à l'autre lors du croisement. Les auteurs n'ont fait évoluer que des réseaux de type perceptron, mais cette approche doit pouvoir se transposer à des structures quelconques. La principale limitation réside cependant dans l'obligation d'utiliser une grille de taille fixe, imposée initialement. Un autre problème est lié à l'utilisation du croisement, c'est le problème de la permutation (Radcliffe, 1993) : un même réseau peut être représenté de différentes façons. C'est le cas, par exemple, d'un perceptron à une couche cachée dont on peut représenter les neurones cachés dans différentes séquences. Ainsi, le croisement de deux réseaux identiques mais dont les neurones sont placés dans un ordre différent donnera, a priori, un résultat peu efficace, car certains neurones (avec leur connexions entrantes et sortantes) apparaîtront à plusieurs reprises, alors que d'autres ne figureront pas dans l'individu généré. *NEAT* (Stanley et Miikkulainen, 2002b) apporte une solution originale à ce problème. Chaque connexion dispose d'un numéro d'innovation, attribué de façon unique

[25] Ou bien elle pourra ne rien décrire du tout. Elle sera alors une partie non utilisée du génome, un *intron*.

Nom	**Type de réseau**	**Mutation**	**Croisement**	**Poids des connexions**	**Limitations structurelles**
INNERVATOR (Miller *et al.*, 1989)	perceptron	?	?	apprentissage	?
GENITOR (Whitley *et al.*, 1990)	perceptron	?	?	évolués	?
GNARL (Peter Angeline *et al.*, 1994)	quelconque	Taux variable en fonction des performances	non utilisé	évolués	aucune
EPNet (Yao et Liu, 1997)	perceptron	suppression de neurones et de connexions en utilisant un algorithme d'élagage, ajout aléatoire de connexions et ajout de neurones par duplication	non utilisé	rétropropagation	aucune
ENS^3 (Pasemann, 1997; Pasemann et Dieckmann, 1997a; Pasemann et Dieckmann, 1997b)	quelconque	manipulation de la matrice de connexion du réseau	non utilisé	évolués	aucune
PDGP (Pujol et Poli, 1997)	perceptron	addition/suppression de connexions, croisement avec un chromosome/nœud généré aleatoirement	utilise une grille	évolués	nombre de nœuds fixe

FIG. 4.1: *Principales caractéristiques des différentes approches basées sur un codage direct du réseau (1).*

Nom	**Type de réseau**	**Mutation**	**Croisement**	**Poids des connexions**	**Limitations structurelles**
NEAT (Stanley et Miikkulainen, 2001)	quelconque	ajout de connexions entre deux nœuds, remplacement d'une connexion par une connexion - un nœud - une connexion	utilise un marquage historique	évolués	aucune
Marqueurs (Fullmer et Miikkulainen, 1991)	quelconque	modification aléatoire d'éléments de la chaine de marqueurs	croisement à deux points	évolués	taille fixe
SANE (Moriarty et Miikkulainen, 1997)	perceptron	un chromosome code pour un neurone et ses connexions entrantes et sortantes		évolués	nombre de neurones fixe
ESP(Gomez et Miikkulainen, 1998; Gomez et Miikkulainen, 2002)	perceptron simple ou perceptron avec rebouclage des sorties en entrée du réseau	basé sur SANE, mais avec des sous-populations pour chaque position dans le réseau		évolués	taille fixe

Fig. 4.2: *Principales caractéristiques des différentes approches basées sur un codage direct du réseau (2).*

Définition d'un nœud :

$\boxed{<\textit{début}><\textit{clé}>< valeur_{initiale} >< k_1 >< w_1 >< k_2 >< w_2 > ... < k_n >< w_n >< fin >}$

Où :

- *début* marqueur de départ
- *clé* identification du nœud utilisée pour les connexions
- $valeur_{initiale}$ valeur initiale du nœud avant la première évaluation
- $< k_i >< w_i >$ Spécification des connexions : k_i spécifie l'identification du nœud source. Le nœud ayant la clé la plus proche de k_i est choisi comme source. w_i spécifie le poids de la connexion.
- fin marqueur de fin du nœud

FIG. *4.3: Structure d'un nœud dans le codage de Fullmer et Miikkulainen.*

lors de l'apparition de la connexion. Ce numéro d'innovation permet de définir un opérateur de similarité : sont similaires les chromosomes disposant d'un certain nombre de connexions ayant le même numéro d'innovation. Cette similarité permet de faire la correspondance entre des connexions et d'échanger l'information génétique de connexions historiquement issue d'une même mutation. Ces connexions sont en effet supposées exercer une fonction comparable dans les différents réseaux. La définition d'une similarité permet également de définir des niches écologiques permettant de protéger les nouvelles structures en utilisant le partage de fitness explicite (Goldberg et Richardson, 1987). Cette méthode permet de maintenir la diversité de la population en répartissant les individus selon des niches écologiques. Les individus appartiennent à une même niche s'ils sont suffisamment proches[26]. La note de chaque individu est divisée par la taille de la niche, ce qui donne une chance d'apparaître et de survivre à des génomes très différents du reste de la population.

Évolution de neurones. Miikkulainen et al. apportent une solution originale au problème de la modularité. Elle consiste à ne considérer qu'un neurone par chromosome au lieu d'un réseau complet. Les modules considérés sont constitués d'un seul neurone ainsi que de toutes ses connexions avec les entrées et sorties. L'évaluation se fait en choisissant un certain nombre de neurones et en les assemblant pour former un réseau complet. Cette approche, baptisée SANE (Symbiotic, Adaptive Neuro-Evolution) (Moriarty et Miikkulainen, 1997), déplace le problème de la modularité. Les modules sont à présent les chromosomes. L'évolution, en combinant les différents chromosomes, va avoir tendance à les spécialiser, chacun dans un rôle particulier. Le croisement n'a alors plus le même sens et ne se produit plus au même moment. À présent, l'échange de blocs fonctionnels se produit au moment de la sélection avant évaluation et va devenir un point critique de l'algorithme. Initialement, cette sélection est faite aléatoirement. ESP (Enforced SubPopulations)(Gomez et Miikkulainen, 1998), une méthode dérivée de SANE, est plus directive en ce qui concerne le choix des neurones à assembler : à chaque 'position' d'un

[26]Cette notion de distance dépend des codages utilisés.

neurone dans le réseau correspond une sous-population différente. La création d'un réseau revient alors à choisir un neurone dans chacune des sous-populations qui peuvent alors se spécialiser dans une fonction particulière. SANE et ses approches dérivées nécessitent de savoir comment assembler des neurones pour former un réseau complet, ce qui en est la principale limitation. Il ne s'applique donc qu'à des réseaux de structure fixe et de type perceptron à une couche cachée. ESP s'applique également à des perceptrons avec un rebouclage des sorties en entrée du réseau (Gomez et Miikkulainen, 2002).

4.2.2 Codage indirect

Dans les méthodes décrites précédemment, la correspondance entre un chromosome et le réseau de neurones est bijective, la traduction immédiate. Cependant, pour résoudre les problèmes cités ci-dessus ainsi que d'autres problèmes, comme le passage à l'échelle, il est intéressant de considérer des codages pour lesquels la correspondance génotype-phénotype n'est plus immédiate, mais nécessite une phase de développement. En effet, pour la résolution de problèmes nécessitant des réseaux de taille importante, une approche directe entrainera obligatoirement des génomes de taille importante. Autrement dit le nombre de paramètres à optimiser - en plus de la structure - augmentera exponentiellement, exactement au même rythme que le nombre d'unités et de paramètres du réseau. Dans un codage indirect, la correspondance entre génotype et phénotype n'est plus bijective. La taille du génome augmentera à un rythme plus faible que la taille du réseau, ce qui permettra de manipuler des réseaux de taille importante tout en restant dans un espace de recherche de taille raisonnable. Cela signifie implicitement que l'espace de recherche sera contraint et que toutes les structures ne seront pas possibles. Le problème consistera alors à choisir un codage représentant une famille de réseaux permettant de résoudre les problèmes étudiés. Ce n'est pas forcément gênant car tous les réseaux ne sont pas intéressants - ceux dont les entrées ne sont pas reliés aux sorties, par exemple, génèrent le même comportement qu'un réseau vide et n'ont donc pas besoin d'être inclus dans l'espace de recherche.

Règles de réécriture

Un graphe peut être obtenu à partir d'une cellule initiale par exécution de règles de production. Ces règles vont s'appliquer lorsque certaines conditions seront remplie - présence d'un schéma particulier, par exemple - et vont remplacer une partie du réseau par une autre.

L-Systems. Lindenmayer a développé une classe de fractales appelées L-Systems (Lindenmayer, 1968) pour modéliser le développement des plantes. Un L-Système est un système de réécriture de chaînes en parallèle. Il est composé d'un ensemble de règles de production qui définissent des transformations de chaînes élémentaires en d'autres chaînes élémentaires. Kitano (Kitano, 1990a) et Boers (Boers *et al.*, 1993) ont exploité ce principe pour faire évoluer des réseaux de neurones.

Programmation génétique

La programmation génétique (Koza, 1992) a inspiré de nombreuses méthodes de développement de réseaux de neurones dont la plupart sont inspirées du *Codage Cellulaire* de Gruau (Gruau, 1992; Gruau, 1994; Whitley *et al.*, 1995).

Le codage cellulaire part d'un réseau simple contenant des cellules de développement reliées aux entrées et sorties. Ces cellules exécutent leur programme de développement qui est un programme génétique sous forme d'arbre. Lorsque l'exécution atteint un symbole terminal, la cellule se transforme en neurone. Chaque cellule de développement dispose d'une copie du programme de développement ainsi que d'une tête de lecture qui pointe sur la prochaine instruction à exécuter. Les instructions de développement positionnent les différents paramètres du futur neurone et complexifient la structure par des instructions de division cellulaire. Ces instructions de division sont le point-clé de cette méthode. Il en existe différents types qui transmettent les connexions de différentes manières : elles peuvent les recopier dans la nouvelle cellule, transmettre les connexions sortantes et créer un lien entre la cellule mère et la cellule fille...

SGOCE (Kodjabachian et Meyer, 1995; Kodjabachian et Meyer, 1997; Kodjabachian, 1998; Meyer *et al.*, 2002) est une méthode inspirée du codage cellulaire dans laquelle les cellules de développement sont situées dans un substrat à deux, voire trois, dimensions. Cela permet de créer des connexions depuis, ou vers, des neurones par des opérations autres que des divisions : il est possible de viser une zone du substrat et d'attirer ou de créer une connexion depuis, ou vers, la cellule la plus proche de cette zone (figure 4.4).

L'*Edge Encoding* (Luke et Spector, 1996) est une autre variante du codage cellulaire dans laquelle le développement n'est plus basé sur des cellules de développement, mais sur des liens, des connexions entre cellules. Ce sont ces liens qui exécutent le programme de développement.

Neurogénèse

Si les méthodes évoquées précédemment restent relativement éloignées de la réalité biologique, certaines essaient d'y être beaucoup plus fidèles et mettent en place un développement dépendant d'émissions de médiateurs chimiques. L'évolution a alors pour tâche d'ajuster ces émissions et de déterminer le comportement des cellules développementales en réponse aux concentrations locales des différents agents chimiques (Harvey, 1993; Vaario *et al.*, 1997; Michel, 1999; Astor et Adami, 2000; Eggenberger, 2000; Bongard, 2002).

Autres

Unit Cluster. Salustowicz a développé un codage permettant de générer des réseaux feed-forward modulaires (Salustowicz, 1995). Le génome est une suite de symboles indiquant soit la création d'une unité, soit un changement de couche. Lors du changement de couche, toutes les unités créées depuis le dernier changement de couches sont considérées comme appartenant au même cluster et sont reliées à toutes les unités du cluster précédent. Elles seront reliées à toutes les

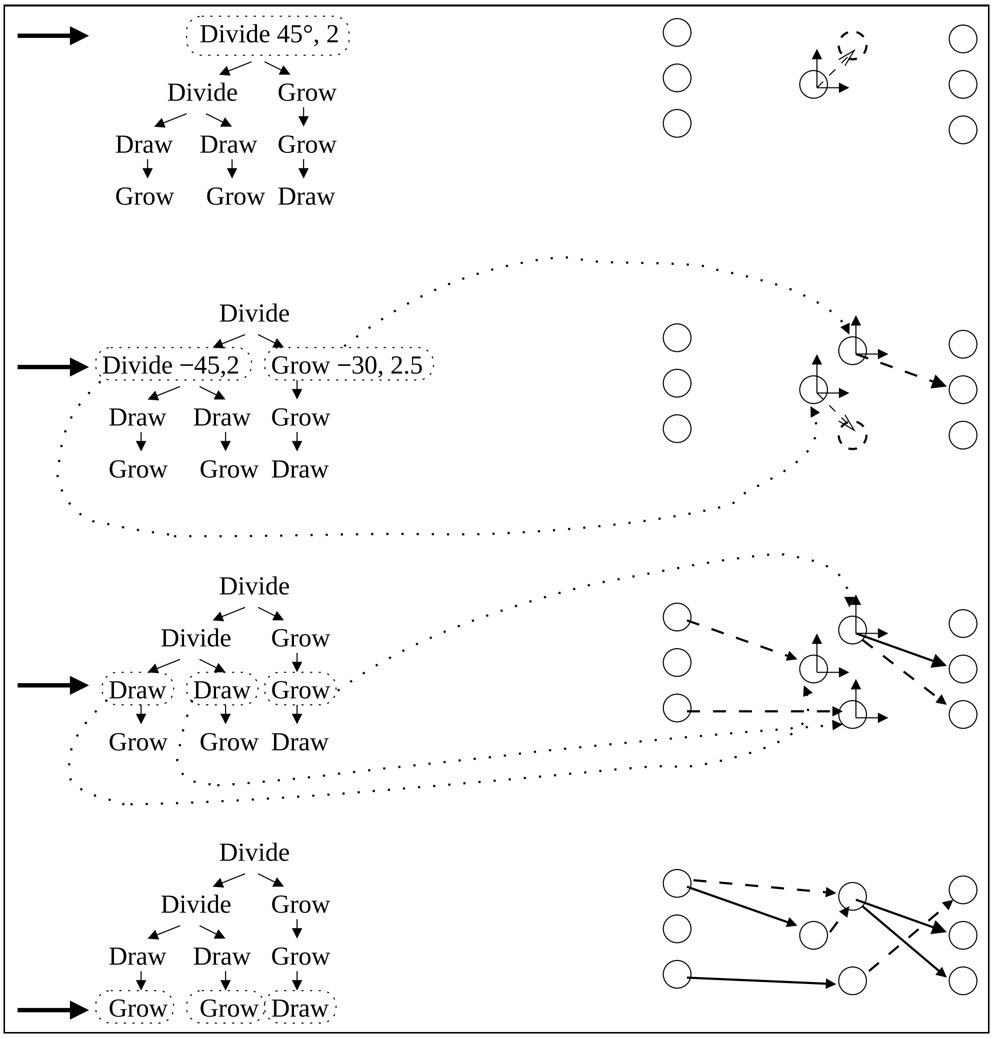


FIG. *4.4: Exemple de développement utilisant SGOCE. Partant d'un substrat initial contenant une cellule ainsi que les neurones d'entrée et de sortie, on exécute le programme de développement, instruction par instruction. Le programme de développement est donné dans la partie gauche de la figure et le réseau généré dans la partie droite.*

unités du cluster suivant. Le réseau est ensuite entrainé avec un algorithme de rétropropagation de l'erreur.

Sparse Trees. Zhang et al. utilisent des arbres “épars” pour représenter des réseaux de neurones Sigma-Pi (Zhang *et al.*, 1997). Dans ce codage, un arbre de développement est associé à chaque entrée et représente la partie du réseau qui relie l'entrée aux sorties. Chaque nœud de l'arbre représente un neurone ainsi que ses paramètres (par exemple le type d'unité : sigma ou pi). Ils ont séparé l'évolution de la structure de l'évolution des poids : une structure étant donnée, un algorithme de recherche local dédié basé sur le Breeder Genetic Algorithm (Zhang et Mühlenbein, 1993) va optimiser les poids.

Gruau a montré l'efficacité d'approches indirectes par rapport à des approches basées sur un codage direct (Gruau *et al.*, 1996). D'autres travaux (Stanley et Miikkulainen, 2001) montrent qu'un codage direct peut être plus efficace qu'un codage indirect sur le problème du pendule inversé. Ce problème nécessite cependant des structures simples, aussi serait-il intéressant de disposer de comparaisons sur des problèmes nécessitant des structures plus complexes.

4.3 Limitations

Plusieurs points limitent les possibilités offertes par l'utilisation conjointe des réseaux de neurones et de l'évolution. Certains des codages présentés jusqu'à présent apportent une solution ou un élément de solution à un ou plusieurs de ces points, cependant aucun ne résout encore entièrement tous ces problèmes. PDGP aborde le problème du croisement (Pujol et Poli, 1997), mais il échange des neurones uniquement et ne peut pas échanger de blocs connexes du réseau. NEAT résout le problème de la permutation (Stanley et Miikkulainen, 2002b), mais ne permet pas d'échanger des blocs fonctionnels. Les approches basées sur l'évolution de neurones (Moriarty et Miikkulainen, 1997; Gomez et Miikkulainen, 1998; Gomez et Miikkulainen, 2002) nécessitent d'avoir une structure fixe.

4.3.1 Poids des connexions

La résolution de problèmes simples de contrôle ne nécessite que peu de neurones et peu de connexions, quelques dizaines au plus. Cependant, si l'on souhaite résoudre des problèmes plus complexes, même si parfois des réseaux simples suffisent, il est fort probable que des réseaux de taille plus importante deviennent nécessaires.

Même lorsque la structure du réseau est connue, l'optimisation des poids des connexions risque de poser problème. En effet, comme nous l'avons vu à la section 3.2.3, l'augmentation de la dimension de l'espace de recherche ralentit fortement la convergence, voire l'empêche totalement. Ce problème de combinatoire n'est pas lié aux algorithmes évolutionnistes, il est présent dans tous les algorithmes d'apprentissage.

Une des solutions proposées jusqu'à présent pour résoudre ce problème consiste à compresser l'information génétique, c'est-à-dire à ne pas mettre les poids de toutes les connexions dans le génome, plusieurs poids étant représentés par une même partie de ce génome. C'est l'approche généralement associée aux codages indirects. Elle restreint l'espace de recherche, facilitant ainsi son exploration et la convergence de l'algorithme évolutionniste.

4.3.2 Structure

Le nombre de graphes possibles augmente exponentiellement avec le nombre de nœuds. C'est pourquoi, même si on dispose d'un moyen pour connaître approximativement le nombre de neurones requis, dès que ce nombre va dépasser un certain seuil, une recherche directe simple se révélera impossible.

Un des principaux problèmes réside dans la génération aléatoire de la structure du réseau. Ce problème est rarement évoqué. Pourtant c'est un des problèmes clés dans la recherche de structure. Des choix effectués lors de la génération aléatoire dépendent pour beaucoup les performances de l'algorithme évolutionniste : c'est le problème de l'amorçage évoué au paragraphe 3.1.3.

En effet, si la plupart des réseaux générés aléatoirement ne relient même pas les entrées aux sorties, ce qui peut être le cas si l'on n'y prend pas garde, peu de structures de la population initiale seront valides et l'évolution aura du mal à converger. Il faudra attendre qu'une mutation structurelle ou qu'un croisement heureux se produise pour que l'évolution puisse réellement commencer...

Certains auteurs choisissent délibérément de partir de réseaux simples reliant directement les entrées aux sorties. C'est le cas de Stanley et Miikkulainen avec NEAT (Stanley et Miikkulainen, 2001), par exemple, mais aussi de Gruau et du réseau initial du codage cellulaire (Gruau, 1994).

D'autres problèmes nécessitent l'apparition d'une structure particulière pour être résolus. Un oscillateur, par exemple, nécessite, selon le type de neurone utilisé, deux neurones ou plus, reliés par des connexions récurrentes dont les poids ont des valeurs particulières. Or, en procédant de façon incrémentale, en partant d'un réseau simple (comme dans NEAT, par exemple), il n'existe pas forcément d'étapes intermédiaires qui permettent d'aboutir à la structure souhaitée. Pour reprendre l'exemple de l'oscillateur, tant qu'une connexion récurrente particulière avec de bons paramètres n'est pas apparue, aucune oscillation ne peut se produire. On se retrouve donc face à un autre problème de l'amorçage de l'évolution.

4.4 Codage : propriétés souhaitées

Du choix du codage dépendent pour beaucoup les performances d'un algorithme évolutionniste. La représentation génétique d'un réseau de neurones a attiré l'attention d'un grand nombre de chercheurs qui ont conçu des codages, ayant chacuns leurs avantages et leurs inconvénients. Devant la multitude de solutions d'ores et déjà existantes, il n'est pas inutile de faire le point sur les propriétés que devrait avoir un codage idéal.

Deux points de vue sont à prendre en compte : l'efficacité du codage en vue de son utilisation avec un algorithme évolutionniste et la réutilisation d'informations préalablement acquises.

4.4.1 Adaptation à l'utilisation d'un algorithme évolutionniste

Un codage ne doit pas seulement représenter un réseau, sa structure doit être adaptée à l'utilisation des opérateurs génétiques de croisement et de mutation. En effet, l'efficacité de l'algorithme évolutionniste dépend pour beaucoup de la façon dont l'espace de recherche est exploré. Aucun théorème ne garantit la convergence de l'évolution pour la conception de réseaux de neurones. Cependant, il est possible de dégager un certains nombre d'heuristiques inspirées des Algorithmes Génétiques pour la faciliter.

Le croisement, pour être efficace et garder son sens, nécessite que soient échangés entre individus des blocs fonctionnels significatifs. L'information échangée doit donc représenter un sous-réseau compact et non des éléments disséminés dans tout le réseau. A priori, échanger des connexions non voisines contre d'autres connexions réparties aléatoirement, semble moins pertinent que d'échanger des morceaux connexes de réseau.

Les mutations doivent assurer une exploration principalement locale de l'espace des structures tout en gardant la possibilité d'effectuer des sauts plus importants.

4.4.2 Réutilisation d'informations

Un codage idéal devrait permettre de profiter des informations acquises précédemment dans la résolution d'un nouveau problème. Il nous semble ainsi intéressant de disposer d'un codage dont la puissance expressive est susceptible de progresser avec l'augmentation du nombre des expériences.

Cet aspect n'est pas souvent pris en compte, si ce n'est dans le cas de la résolution incrémentale de problèmes. Cependant, cela se ramène souvent à fixer une partie du réseau et à faire évoluer une nouvelle partie en partant de zéro, ou alors à partir des individus de la dernière génération sur le problème simple en modifiant uniquement la fitness pour prendre en compte d'autres aspects du problème.

Dans le chapitre suivant, nous allons présenter *ModNet*, le codage que nous avons développé pour répondre à ces besoins.

Chapitre 5

ModNet : évolution de réseaux modulaires

5.1 Introduction

L'évolution naturelle ne s'est pas attaquée directement à la création de formes de vie complexes. Les formes de vie évoluées, telles les mammifères, ne sont arrivées que très tard d'après l'horloge de l'évolution. Les premières formes vivantes étaient des êtres mono-cellulaires simples. La création d'une cellule eucaryote, protégée par une membrane et disposant d'un noyau, est intervenue ultérieurement et les êtres pluri-cellulaires ne sont apparus que bien plus tard. Ces êtres mono et pluri-cellulaires primitifs, qui existent toujours, ont créé de nouvelles possibilités et ont donné l'occasion à des êtres vivants d'occuper de nouvelles niches écologiques. Leurs capacités, accrues au cours du temps par des processus de sélection naturelle, leur ont permis de survivre dans des conditions parfois difficiles en exploitant des ressources inutilisables auparavant. L'apparition de cellules complexes, puis leur spécialisation, ont fourni des éléments de base qui ont pu être utilisés et recombinés pour la création d'organismes multi-cellulaires qui ont colonisé progressivement toute la planète. Si l'on s'intéresse à la complexité des formes de vies existantes, la progression depuis l'apparition de la vie n'est pas linéaire. Elle a été très lente initialement, puis s'est ensuite brutalement accélérée. Ce changement de rythme dans l'apparition de formes de vie complexes apparaît lié à la découverte d'éléments réutilisables, comme les cellules eucaryotes.

La plupart des solutions que les différentes espèces animales ou végétales ont ainsi trouvées pour résoudre les problèmes auxquels elles étaient confrontées n'étaient que des recyclages d'autres éléments qui avaient servi dans un tout autre contexte : les êtres multicellulaires sont un assemblage de cellules qui vivaient isolées auparavant, les pattes des premiers sauriens sont issues des nageoires modifiées de leurs ancêtres, les ailes des oiseaux sont des pattes affinées sur lesquelles une membrane puis des plumes se sont développées.

L'objectif que nous nous sommes fixé dans cette thèse n'est pas de chercher à imiter la sélection naturelle des espèces dans ses moindres détails, mais de disposer d'une méthode de conception automatique de contrôleurs de robots aussi efficace que possible. Cette méthode doit pouvoir exploiter des caractéristiques spécifiques de l'interaction du robot utilisé et de son environnement et elle doit pouvoir prendre en compte des connaissances précédemment acquises.

Dans cette perspective, nous avons choisi d'utiliser des réseaux de neurones qui disposent d'importantes capacités d'apprentissage et peuvent générer des comportements très variés qui peuvent être réactifs ou avoir une dynamique complexe. L'évolution de réseaux de neurones est cependant un problème délicat, principalement à cause de la taille de l'espace de recherche qui croît exponentiellement avec le nombre d'entrées, le nombre de sorties ainsi qu'avec la taille du réseau. Une façon de dépasser ce problème consiste à procéder de façon incrémentale : commencer par des problèmes simples, puis progressivement gagner en complexité. De nombreux chercheurs ont utilisé cette idée. Dans (Kodjabachian et Meyer, 1997), les auteurs résolvent tout d'abord le problème de la marche d'un robot hexapode, puis l'évitement d'obstacle en conservant le contrôleur de marche obtenu dans un premier temps, puis la remontée d'un gradient lumineux en conservant les solutions précédentes. Urzelai a proposé une méthodologie reposant sur ce principe (Urzelai *et al.*, 1998) : le comportement que le robot doit exhiber est analysé et décomposé en comportements élémentaires qui seront implémentés dans des modules avant d'être regroupés dans une architecture globale. Les modules utilisés génèrent, en plus de leur signal de sortie, un signal d'activation qui sera utilisé pour déterminer le signal à choisir. Lara et al. (Lara *et al.*, 2001) font évoluer séparément des modules résolvant différents problèmes (évitement d'obstacles et remontée de gradient lumineux), puis font évoluer l'interface qui les réunit, de façon à résoudre le problème complet.

Ce type d'approche nécessite une décomposition comportementale manuelle de la tâche à résoudre. Dans les cas évoqués précédemment, la décomposition est relativement évidente, mais c'est loin d'être le cas général. Nous avons déjà cité le cas de robots résolvant une tâche de regroupement d'objets avec un comportement d'évitement d'obstacles bien ajusté (Maris et te Boekhorst, 1996). Dans cette expérience, un contrôleur reliant directement les entrées aux sorties peut résoudre le problème si les capteurs sont positionnés convenablement. Une décomposition manuelle non réfléchie - recherche de blocs, ramassage, recherche d'un tas et dépôt sur le tas - rend le problème beaucoup plus complexe qu'il ne l'est réellement, comme nous l'avons vu dans l'introduction.

Ce type d'incrémentalité exploite une décompostion des tâches à résoudre, cependant, il en existe d'autres formes. L'incrémentalité peut aussi se situer dans les moyens mis à la disposition de l'évolution, qui doivent être de plus en plus puissants et adaptés à la résolution de problèmes de complexité croissante. La décomposition comportementale ne permet pas de traiter directement des tâches de plus grande complexité : elle transforme un problème complexe en une série de problèmes simples que l'on sait déjà résoudre. Une incrémentalité dans les moyens mis à la disposition de l'évolution permettrait à l'évolution de résoudre directement des problèmes de complexité croissante, sans avoir à les analyser en profondeur.

5.2 Présentation de ModNet

Le système que nous avons conçu dans le cadre de cette thèse, *ModNet*, a pour double objectif de définir un codage plus souple - prenant en compte des modules et non des neurones individuels - et d'exploiter potentiellement des connaissances acquises précédemment dans la résolution de nouveaux problèmes.

Le principe consiste tout d'abord à changer la granularité de la représentation du réseau de neurones, afin de ne plus manipuler des neurones individuellement, mais des modules représentant des sous réseaux de complexité variable qui peuvent être réutilisés d'une expérience à l'autre.

Les modules sont issus d'un réservoir de modules initiaux dont la structure reste globalement fixe, mais dont les paramètres vont être soumis à évolution. Cet ensemble de modules initiaux constitue la base de connaissance mise à la disposition de l'évolution. Ils peuvent être de structure fixe, imposée initialement, ou être générés aléatoirement.

5.2.1 Travaux similaires

L'idée d'utiliser des modules dans les réseaux de neurones pour faciliter le passage à l'échelle, notamment lors de la résolution de problèmes multi-critères, a déjà été exploitée à de multiples reprises. Buessler et Urban présentent une revue des applications de réseaux de neurones modulaires dont la structure est fixe, mais dont les poids sont soumis à un apprentissage (Buessler et Urban, 2002). Ils présentent un exemple d'application au positionnement visuel d'un bras robotique.

Dans des architectures de type perceptron, un apprentissage de type rétropropagation sur une structure entièrement connectée va avoir tendance à répartir l'information sur toutes les connexions du réseau. Pour des problèmes de complexité importante, des réseaux modulaires - concurrents ou chacuns spécialisés dans un domaine particulier - peuvent entraîner un important gain de performance (Sharkey, 1996; Sharkey, 1999). De plus, la modularité peut diminuer la complexité du réseau et donc simplifier son interprétation ou le rendre plus facile à étendre (Gallinari, 1995).

Ruekl et al. (1989) ont démontré que l'identification et la localisation d'un objet étaient plus efficacement réalisés si l'on utilisait deux modules séparés plutôt qu'un seul. Di Fernando, Calabretta et Parisi ont utilisé l'évolution pour étudier ce même problème et ont obtenu des réseaux à couches modulaires (DiFerdinando *et al.*, 2001). De même, Nolfi a démontré qu'un réseau permettant l'émergence de modules est plus performant que d'autres types de réseaux sur un problème se décomposant en tâches élémentaires que le réseau doit effectuer séquentiellement (Nolfi, 1997). Il a toutefois observé que les modules générés par évolution ne correspondent pas au découpage qu'un concepteur humain aurait réalisé. En effet, à un module du réseau généré ne correspond pas nécessairement un comportement intuitivement élémentaire pour ce problème.

Dans la génération de comportements complexes, une architecture modulaire peut ainsi permettre de décomposer le problème : chaque comportement élémentaire sera associé à un élément particulier de l'architecture. Duro, Santos et Becerra proposent une architecture hiérarchique modulaire dans laquelle chaque module représente un comportement particulier, qu'il soit de bas-niveau (évitement d'obstacles) ou de plus haut niveau ("suivre un mur", qui déclenche des comportements de bas-niveau comme "trouver un mur", "suivre un mur de près", etc) (Duro *et al.*, 2002). Ils ont également proposé l'utilisation de capteurs et d'effecteurs virtuels permettant d'abstraire les modules comportementaux et de les réutiliser d'un robot à l'autre.

Dans certains problèmes, on connait déjà une structure élémentaire efficace. Pour le passage à l'échelle, on peut alors combiner ces structures. Happel et Murre font ainsi évoluer des assemblages de modules CALM (*Categorizing and Learning Module*) pour résoudre plus efficacement des problèmes de catégorisation (Happel et Murre, 1994). Ces modules ont une structure particulière à base de winner-takes-all et de nœuds "véto" dédiée à la classification non supervisée.

SANE (Moriarty et Miikkulainen, 1997) et ses approches dérivées, comme ESP (Gomez et Miikkulainen, 2002), implémentent aussi une forme de modularité. Dans ces méthodes, l'évolution ne manipule plus des réseaux complets, mais des neurones (avec leurs connexions entrantes et sortantes). Chacun des neurones va progressivement se spécialiser dans la résolution d'une partie de la tâche et constitue ainsi un module fonctionnel.

Le Codage Cellulaire est un codage indirect basé sur la programmation génétique de Koza (voir section 4.2.2.0). Dans (Gruau, 1995), Gruau a étudié l'effet de l'utilisation d'ADF (Automatically Defined Functions) - rebaptisées "Automatic Definition of Sub Neural Networks" (ADSN) - sur l'évolution de contrôleurs pour un robot hexapode. Le principe consiste à ne plus utiliser un seul arbre de développement, mais plusieurs. Les arbres sont ordonnés et les nœuds terminaux d'un arbre sont des liens vers un des arbres suivants (sauf pour le dernier). Cette approche permet de réutiliser un arbre de développement à plusieurs reprises lors du développement d'un réseau de neurones. Le nombre d'arbres utilisés est, cependant, imposé initialement. Le Codage Cellulaire a inspiré d'autres travaux portant sur la modularité de réseaux de neurones. Friedrich et Moraga ont utilisé une version du Codage Cellulaire modifiée pour ne générer que des réseaux à couches (Friedrich et Moraga, 1997). Ils ont utilisé une base des graphes générés afin d'étudier les structures qui ont donné les meilleurs résultats. Ils en ont extrait des modules intéressants et ont relancé des expériences exploitant ces modules.

Kodjabachian propose de répliquer un module à plusieurs reprises pour contrôler un robot hexapode (Kodjabachian et Meyer, 1998). Chaque module est le résultat du développement d'un même programme génétique mais, commme l'exécution des différentes instructions est dépendante du contexte[27], les modules ne sont pas tous identiques. Ils peuvent en particulier se connecter les uns aux autres, ce qui permet de générer les synchromisations nécessaires au développement de la marche du robot hexapode. La modularité est cependant imposée par l'expérimentateur dans ces expériences.

[27]Les neurones peuvent se connecter à leurs voisins topologiques. Les différents modules étant dans le même substrat, ils peuvent intervenir dans le développement des autres modules.

5.2.2 Modules

ModNet utilise un codage de réseau modulaire. L'élément de base est un module, un sous-réseau, possédant une ou plusieurs entrées et sorties. Le réseau de neurones est alors un assemblage de ces modules dont l'agencement est décrit dans le chromosome. La structure des modules ne change pas, ou peu, pendant la boucle évolutive, alors que leurs paramètres (poids des connexions par exemple) et leur agencement dans le réseau sont soumis à évolution. Dans ce contexte, les réseaux utilisés dans ModNet ne sont pas contraints à correspondre à un comportement élémentaire contrairement à d'autres approches (Nolfi, 1997; DiFerdinando *et al.*, 2001; Duro *et al.*, 2002). L'aspect modulaire lié à ModNet apparaît dans la définition de la structure. Ainsi les modules peuvent correspondre à un calcul intermédiaire propre au fonctionnement du réseau sans coder pour autant un comportement particulier observable, mais plutôt une fonctionnalité, telle qu'un calcul de dérivée, une génération de sinusoïde ou encore un interrupteur. Ce sont de telles fonctionnalités qui, regroupées avec d'autres, peuvent générer les comportements exhibés par le robot.

La complexité des modules n'est pas imposée. Le réservoir initial contient des modules générés manuellement, de façon aléatoire, ou bien issus d'expériences précédentes : lorsqu'une expérience a convergé, les modules utilisés par la (ou les) meilleure(s) solution(s) peuvent être ajoutés au réservoir dans le but d'être réutilisés dans de futures expériences (figure 5.1).

5.3 Chromosome

Un chromosome est composé de deux listes de modules et d'une liste de connexions inter-modules (figure 5.2 et 5.3) . La première liste de modules contient les “modules-modèles”, copies d'éléments du réservoir de modules - dont les paramètres sont soumis à évolution. Leur nombre n'est pas limité. La deuxième liste contient les modules qui seront effectivement utilisés - modules qui sont des copies des modules-modèles. Ce mécanisme permet de réutiliser à plusieurs endroits dans le réseau un même module-modèle, propriété intéressante pour permettre la réutilisation du code génétique[28]. La liste de connexions précise comment connecter les modules entre eux, les neurones d'entrée et de sortie étant considérés comme des modules particuliers n'ayant respectivement qu'une sortie ou qu'une entrée. Une connexion ne concerne qu'un seul neurone d'entrée ou de sortie d'un module.

Le lien d'un module à un autre ne se matérialise pas par une connexion, mais par la fusion du neurone d'entrée du module destination et du neurone de sortie du module source avec un neurone intermédiaire (figure 5.4). Les paramètres du réseau sont ainsi entièrement contenus dans les modules. Les neurones intermédiaires sont tous identiques et d'un type prédéfini dans le chromosome par l'expérimentateur. Nous avons prévu la possibilité de différencier les neurones intermédiaires et de faire évoluer leurs éventuels paramètres, mais nous ne nous en sommes pas servi dans les applications décrites dans cette thèse. Aussi, dans toutes les expériences décrites

[28]Ce qui permet notamment de faire apparaitre des symétries.

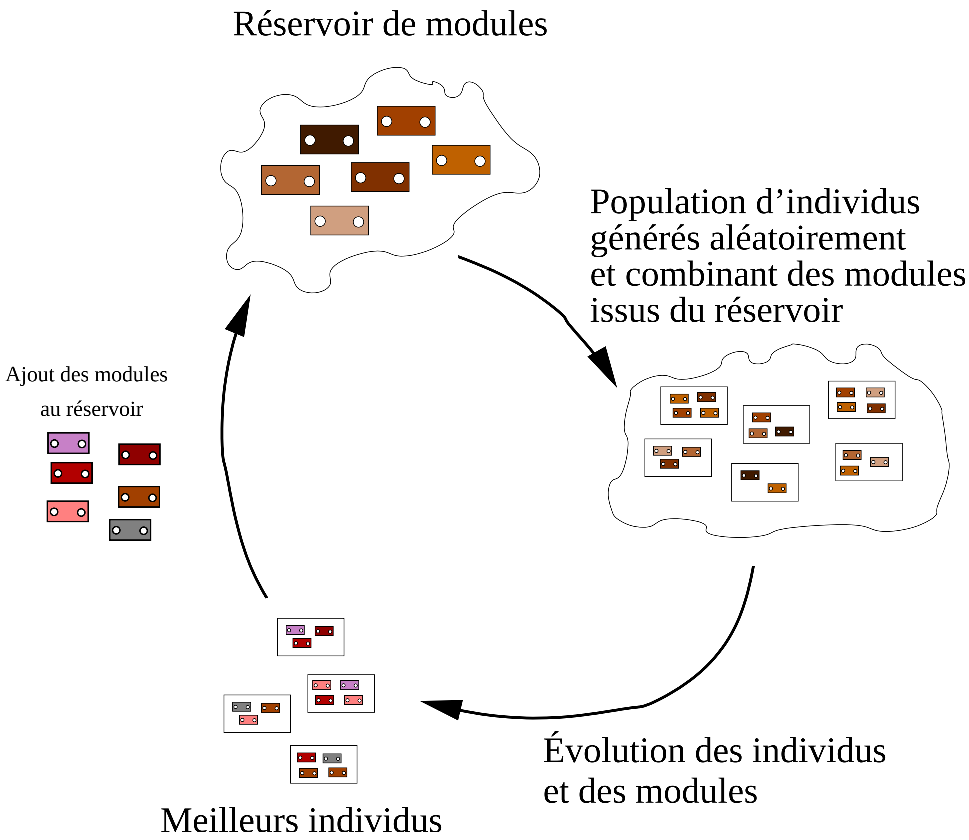


FIG. *5.1: Principe de fonctionnement de ModNet : boucle évolutionniste et réservoir de modules. Les individus exloitent les modules donnés dans le réservoir, l'évolution peut éventuellement les modifier et de nouveaux modules peuvent alors être extraits des meilleurs solutions générées et être intégrés au réservoir pour de futures expériences.*

ci-après, toute l'information paramétrique des réseaux de neurones est contenue dans les modules-modèles. Les vecteurs de modules et de liens ne contiennent qu'une description structurelle du réseau.

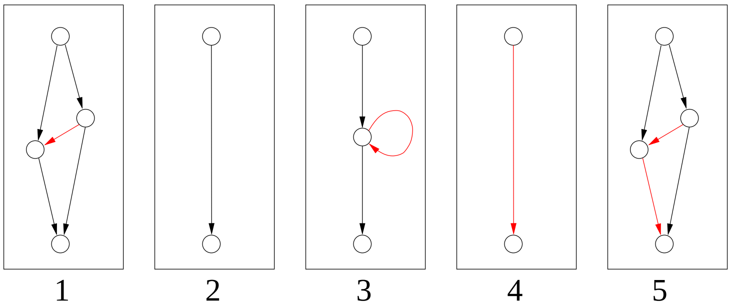
5.4 Opérateurs génétiques

5.4.1 Mutations

Mutations non structurelles

Outre les mutations de paramètres et d'éventuelles mutations structurelles à l'intérieur des modules, ModNet permet plusieurs types de mutations permettant de modifier aléatoirement les différents éléments d'un chromosome. La liste de modèles peut subir la suppression d'un module choisi aléatoirement ou l'ajout d'un nouveau module, copie d'un module du réservoir. La liste de modules peut aussi être soumise à des mutations. Comme précédemment, un module peut être supprimé ou ajouté. Une autre mutation permet de changer le modèle dont un module est

Liste de modules−modèles



Liste de modules

m1	m2	m3	m4	m5	m6	m7
3	2	1	1	5	4	2

Liste de connexions

m1−>m3	m1−>m2
m2−>o0	m6−>o2
m3−>o1	m7−>o1
i1−>m4	m4−>m6
m4−>m7	i0−>m1

Fig. 5.2: *Exemple de chromosome utilisé dans ModNet. Le chromosome est composé de trois parties : une liste de modules-modèles (deux modules de structure identique peuvent avoir des poids de connexions différents), une liste de modules et une liste de connexions. La liste de modules contient le numéro du modèle dont chaque module est la copie. Ainsi, les modules $m3$ et $m4$, qui sont tous les deux la copie du même module-modèle n°1, peuvent apparaître dans le génotype d'un individu. Le phénotype correspondant contiendra donc le module-modèle n°1 en deux endroits distincts (si les modules $m3$ et $m4$ sont référencés dans la liste des connexions). i_n est la n^{ieme} entrée et o_n est la n^{ieme} sortie.*

la copie[29].

Mutations structurelles

Les autres mutations vont influencer les connexions inter-modules qui définissent la structure du réseau. Un module peut être relié aux autres grâce à deux mutations différentes.

La première et la plus simple est le "branchement" direct. Chaque entrée et chaque sortie du module sont reliées à un neurone intermédiaire choisi aléatoirement. Ce branchement peut être sans autre effet sur le reste du réseau ou entraîner des transmissions de liens. Les liens transmis peuvent être les liens sortants des neurones intermédiaires en entrée du module. Ils sont transmis aux neurones intermédiaires en sortie du module. Les liens en entrée des neurones intermédiaires en sortie du module peuvent également être transmis aux neurones intermédiaires en entrée du module (figure 5.5).

[29]Le nouveau modèle choisi doit disposer du même nombre d'entrées et de sorties que l'ancien.

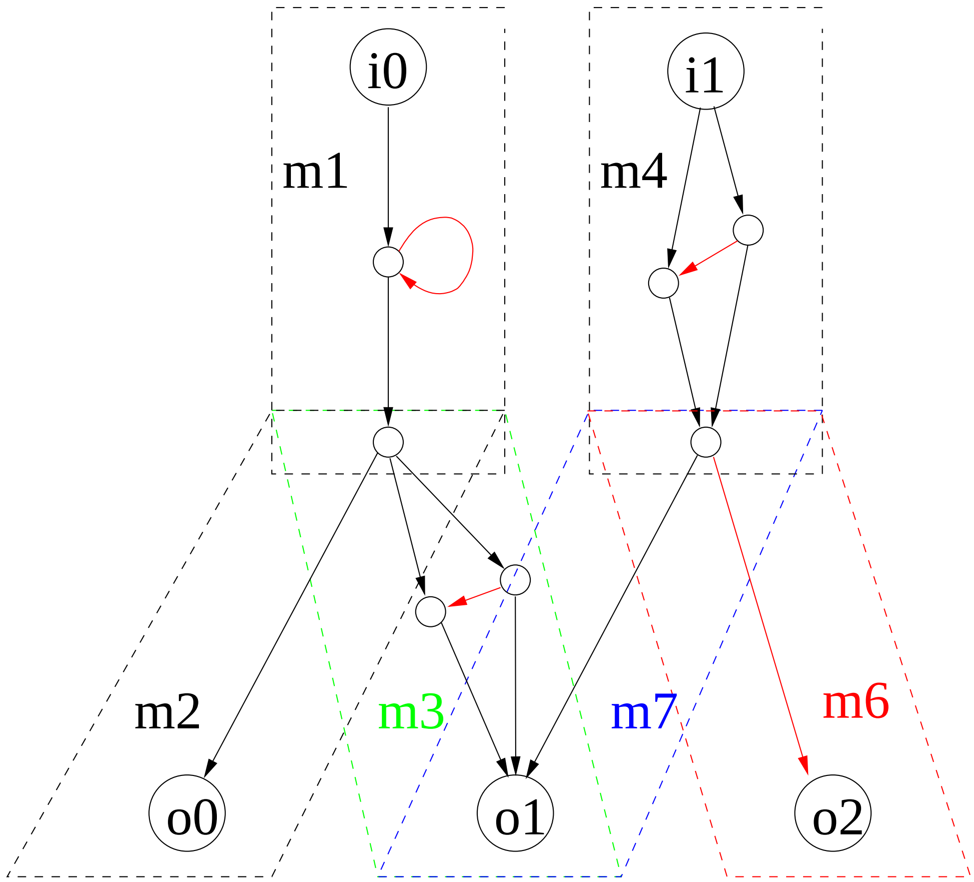


Fig. *5.3: Réseau de neurones correspondant au chromosome de la figure 5.2. Le module-modèle 1 est utilisé en deux endroits dans le réseau (m3 et m4). De même pour le module-modèle 2 (m2 et m7).*

La deuxième mutation est une insertion. Tant que toutes les entrées et les sorties du module n'auront pas été reliées, on tire un neurone intermédiaire, que l'on relie à une entrée (respectivement une sortie) du module tirée aléatoirement. Les liens sortants (resp. entrant) de ce neurone sont transmis à une sortie (resp. entrée) du module choisie aléatoirement (figure 5.6).

D'autres mutations vont permettre de simplifier la structure du réseau en supprimant des modules. Là encore, deux schémas sont possibles. Soit la suppression ne modifie rien d'autre, soit elle fusionne les neurones intermédiaires en entrée et en sortie du module à enlever (opération inverse de l'insertion telle qu'elle est représentée sur la figure 5.6).

5.4.2 Croisement

Le croisement dans ModNet revient à échanger des modules-modèles entre chromosomes. Deux variantes sont possibles :

- échange de modules 'compatibles', c'est-à-dire issus d'un même module-modèle (ils diffèrent par leurs paramètres, qui auront subi des générations aléatoires et des mutations différentes)
- échange de modules dotés du même nombre d'entrées/sorties.

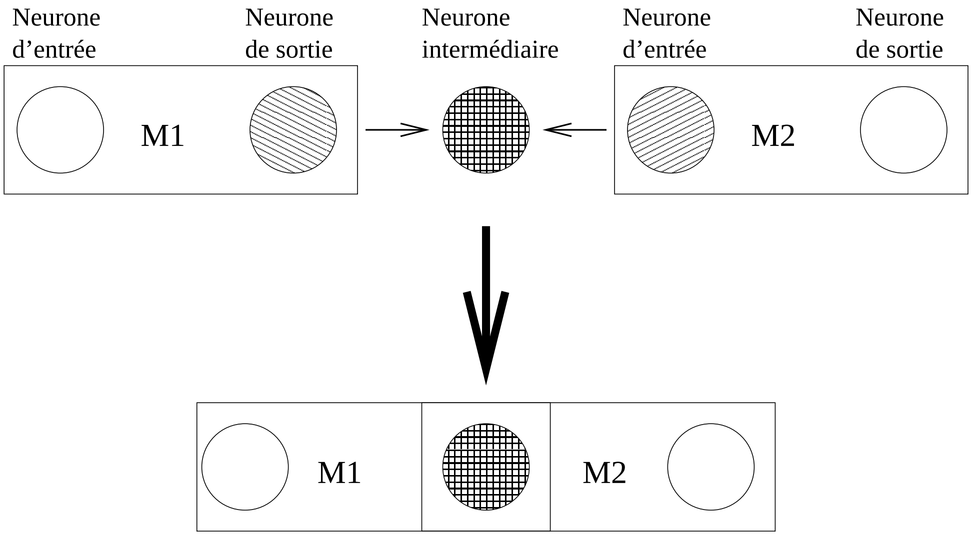


FIG. *5.4: Matérialisation d'un lien entre deux modules : le neurone de sortie du module source est fusionné avec le neurone d'entrée du module destination.*

Dans les expériences utilisant des modules complexes, le croisement permet ainsi d'échanger des éléments structurels entre réseaux.

5.5 Utilisation

ModNet offre une grande souplesse d'utilisation permettant d'inclure de la connaissance a priori à différents niveaux.

5.5.1 Réservoir de modules

Le réservoir de modules utilisé constitue le principal et le plus évident des biais que l'on peut imposer à l'évolution. Il peut contenir des modules dont la structure a été générée aléatoirement, des modules issus de précédentes expériences, ou bien des modules dont la structure a été préparée manuellement pour implémenter une fonction connue comme étant nécessaire à la résolution du problème considéré : calcul de dérivée ou oscillateur, par exemple.

De plus, selon la complexité du problème et le niveau de connaissance a priori, il est possible d'ajouter des contraintes sur les paramètres des modules (les poids des connexions principalement, mais aussi éventuellement les paramètres des neurones). L'approche par défaut consiste à faire évoluer tous les paramètres, mais le signe des connexions, par exemple, peut être imposé. Il est même possible de figer totalement les poids des connexions et de laisser à l'évolution le seul soin de choisir les modules et de les connecter. Il est également possible de laisser les connexions évoluer, mais en partant de valeurs prédéfinies avec, par exemple, de faibles taux de mutations pour que l'évolution ne procède qu'à de légères adaptations paramétriques.

5.5.2 “Patron” de réseau

La prédéfinition complète de la structure revient à faire évoluer les poids dans un réseau de structure fixe, une procédure courante dans la littérature (Blynel et D., 2002; Nolfi et Marocco, 2002; Bongard et Pfeifer, 2002). Cependant, ModNet permet de réaliser des approches intermédiaires entre de telles expériences à structure fixe et des expériences dont la structure est entièrement libre.

Il est ainsi possible de fixer un “patron” de réseau donnant un assemblage de modules. Ce patron sert pendant la phase de génération aléatoire. Il indique le nombre de modules ainsi que la façon de les relier entre eux. La génération aléatoire consiste alors à choisir un modèle pour chacun de ces modules (le modèle ayant obligatoirement le même nombre d'entrées et de sorties que le module). Les paramètres des modèles sont générés aléatoirement pendant cette phase. Les mutations fonctionnent ensuite comme décrit précédemment. Le patron ne sert que lors de la génération aléatoire.

Pasemann et al. ont exploité une stratégie similaire dans (Pasemann *et al.*, 2000). Pour résoudre des problèmes A et B, il se donnait la possibilité d'imposer dans le réseau la structure permettant de résoudre le problème A. L'évolution pouvait ensuite ajouter des neurones ou éventuellement modifier la structure proposée afin de résoudre les deux problèmes. Dans cette approche, c'était la structure du réseau qui était imposée, alors que dans ModNet, le patron indique comment les modules doivent être reliés entre eux et c'est ensuite à l'évolution de trouver les modules les plus efficaces. Cependant, dans une expérience n'ayant que des modules simples dans le réservoir initial de modules[30], le patron de ModNet est équivalent à l'approche employée par Pasemann.

5.5.3 Évolution de structures de composants hétérogènes

La plupart des expériences portant sur l'évolution de réseaux de neurones utilisent des éléments homogènes : un seul type de neurones, un seul type de connexions, les mêmes paramètres pour décrire chaque élément structurel du réseau.

ModNet propose un framework dans lequel les modules sont des unités fonctionnelles gérées indépendamment les unes des autres. Cette indépendance permet, par exemple, de mélanger des neurones intermédiaires ayant des fonctions de transfert différentes ou encore de mélanger des connexions dont le poids appartient à des intervalles de valeurs différentes.

Cela permet, par exemple, de réaliser des expériences dans lesquelles des neurones intégrateurs à fuites sont utilisés en même temps que des neurones traditionnels. Ces capacités peuvent être intéressantes pour l'étude de contrôleurs dans lesquels la gestion du temps est importante, comme le contrôle de la marche d'un robot hexapode, par exemple. Elles peuvent également permettre de réaliser des comparatifs sur les types de neurones utilisés. Nous ne nous sommes toutefois pas servi de ces capacités dans les expériences décrites ici, dans lesquelles tous les neurones utilisent une fonction de transfert de type *tanh*.

[30] Modules ayant une entrée et une sortie reliées par une connexion.

5.6 Évolution des paramètres

L'évolution de paramètres peut être réalisée de plusieurs façon. En fait, ModNet ne définit que la partie structurelle du réseau. Les paramètres peuvent être des chaînes binaires, comme dans les Algorithmes Génétiques, ou encore des réels, comme dans les Stratégies Évolutionnistes. Du point de vue de ModNet, ce sont des objets capables de subir une mutation. L'implémentation est choisie par l'expérimentateur[31]. Ces paramètres font partie des modules modèles et sont échangés avec eux lors du croisement.

5.7 Espace de recherche

Les réseaux susceptibles d'être générés par ModNet sont de structure quelconque. Des perceptrons multi-couches peuvent être générés, de même que des réseaux récurrents. Les réseaux générés vont être fortement influencés par l'algorithme permettant de choisir les neurones intermédiaires auxquels un module va se connecter.

Les probabilités de choix des neurones intermédiaires dans ModNet sont influencées par trois considérations. Lors du choix d'un neurone intermédiaire comme entrée d'un module (respectivement sortie d'un module), ce sont :

- le neurone intermédiaire est-il directement relié à une entrée (respectivement sortie) non encore connectée au réseau ?
- le neurone intermédiaire est-il relié à une entrée (respectivement sortie), directement ou indirectement (cette entrée étant déjà reliée au réseau) ?
- le neurone intermédiaire est-il "libre", c'est-à-dire relié à aucun autre module, entrée ou sortie ?

Dans les expériences décrites dans cette thèse, nous avons appliqué des probabilités de choix de 1 pour le premier cas, 0,1 pour le deuxième et 0 pour le troisième[32]. Cela nous permet de générer des réseaux reliant, en moyenne, toutes les entrées à toutes les sorties. Un jeu de paramètres de $(1, 1, 0)$, aura tendance à générer des réseaux ne prenant pas en compte toutes les entrées (ou toutes les sorties). Une valeur non nulle sur le troisième paramètre aura tendance à générer des sous-réseaux isolés, c'est-à-dire non connectés aux entrées ou aux sorties.

Ces trois paramètres nous permettent donc de modifier facilement la zone de l'espace de recherche dans laquelle l'évolution doit se concentrer.

---

[31] Dans les expériences décrites dans la suite de cette thèse, nous avons utilisé des chaînes binaires de 8 bits (dont un bit de signe).

[32] Avec ces valeurs, la somme des probabilités n'est pas de 1, aussi nous le normalisons avant de faire le choix.

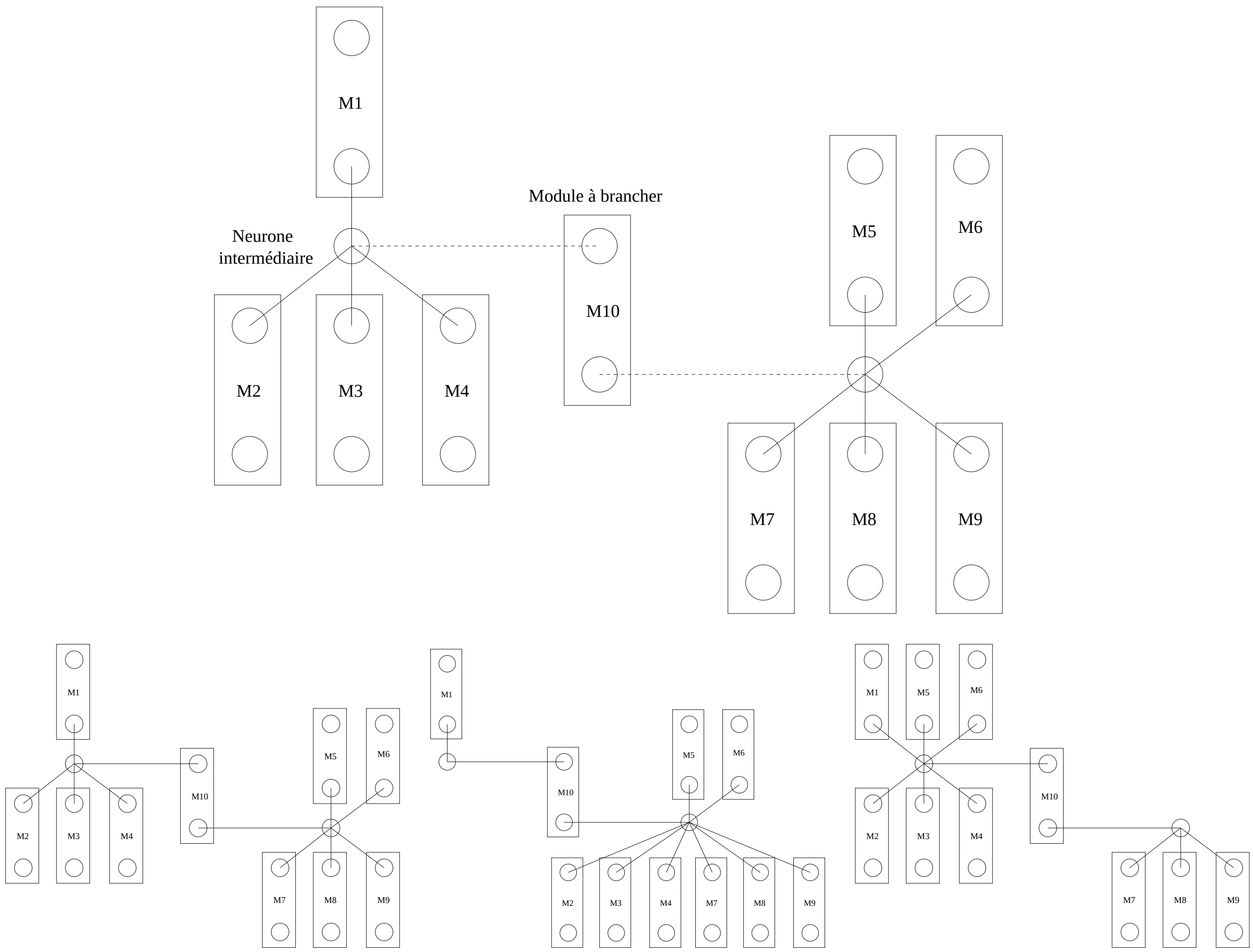


FIG. *5.5: Branchement d'un module (mutation structurelle). Les points de branchement étant choisis (schéma du haut), trois cas de figure sont possibles. On ne modifie pas le réseau (en bas à gauche), on transmet aux sorties des liens issus des entrées (en bas au milieu) ou on transmet aux entrées des liens issus des sorties (en bas à droite). Ces trois possibilités sont équiprobables dans les expériences décrites dans la suite de cette thèse.*

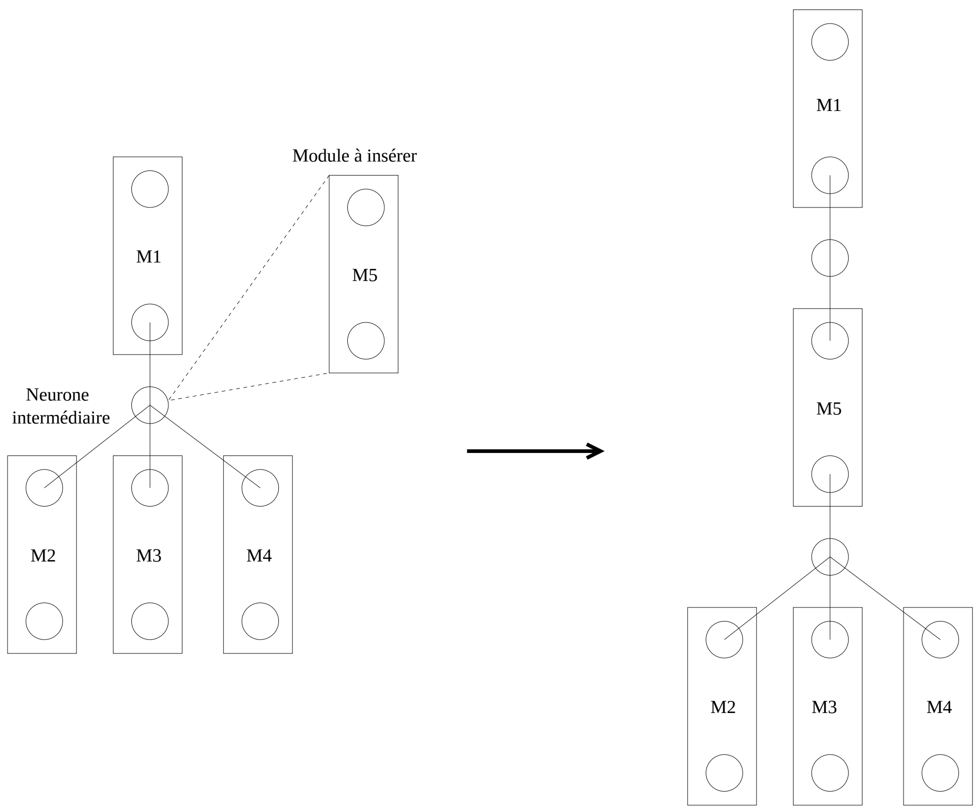


FIG. *5.6: Insertion d'un module (mutation structurelle).*

Chapitre 6

Plateformes logicielles développées

Afin d'implémenter ModNet et de réaliser aisément des comparaisons avec d'autres techniques, nous avons développé plusieurs plateformes logicielles complémentaires que nous allons présenter dans ce chapitre. Notre objectif était de disposer d'outils nous permettant de passer facilement d'une expérience à une autre et également de permettre à d'autres d'utiliser les mêmes outils sans avoir à tout re-implémenter[33].

6.1 SFERES : un framework pour l'évolution et la simulation

Devant la multiplicité et la généralité des algorithmes évolutionnistes existants, nous avons conçu et développé, en collaboration avec Samuel Landau, SFERES [34] une plateforme logicielle générique pour l'évolution et la simulation multi-agents. En effet, si de nombreuses biliothèques existent actuellement pour l'optimisation simple de paramètres - la GALib en est l'exemple sans doute le plus connu (Wall, 2000) - aucune plateforme ne permet de résoudre des problèmes liés à la conception de contrôleurs d'agents depuis la partie évolutionniste jusqu'à la simulation des agents. L'évaluation de contrôleurs d'agents nécessite, non pas une évaluation immédiate, comme dans le cas de l'optimisation de fonction, mais une évaluation temporellement plus étendue, laissant le temps à l'agent de prouver ses capacités (ou ses incapacités). De plus, si la conception de certains contrôleurs peut se ramener à une simple optimisation de paramètres, elle peut nécessiter dans d'autres cas la manipulation d'informations beaucoup plus structurées (par exemple la manipulation de graphes pour l'évolution de réseaux de neurones), ce que ne permettaient pas les bibliothèques existantes.

SFERES est constitué de deux parties. La première est la partie évolutionniste qui implémente les algorithmes évolutionnistes ; la seconde est la partie simulation qui implémente les simulations multi-agents. Les deux parties sont liées par l'évaluation des individus de l'algorithme évolutionniste grâce à la simulation. L'évaluation porte sur le comportement des agents

[33]SFERES est sous licence GPL, mais n'est pas encore distribué faute de documentation adaptée. SFERES et NeuroSFERES sont déjà utilisées par d'autres personnes au sein de notre équipe.

[34]Sferes is a Framework for Encouraging Research on Evolution and Simulation.

simulés. Ce comportement est décrit, plus ou moins directement selon les cas, dans le génome sur lequel travaille l'évolution.

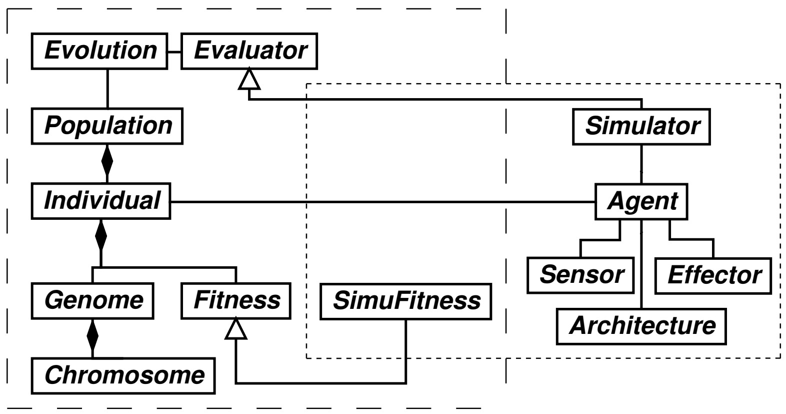


FIG. *6.1: Diagramme UML des classes de SFERES. Encadrée de tirets, la partie évolutionniste et, encadrée de pointillés, la partie simulation.*

La figure 6.1 rassemble les classes abstraites du framework en les regroupant par parties. Chacune de ces classes représente un module interchangeable prenant en charge une partie des caractéristiques d'une expérience.

La partie évolutionniste, encadrée de tirets sur la figure 6.1, est constituée de la gestion bas-niveau des calculs contenue dans `Evolution` (répartition éventuelle des calculs en réseau ou sur des machines multi-processeurs) ainsi que de tous les élements décrivant l'algorithme évolutionniste : l'algorithme de sélection, l'information génétique et la note qui lui est associée. L'algorithme de sélection et la structure de population associée sont contenus dans `Population`. L'information génétique, contenant la description d'une solution potentielle, est implémentée dans les classes `Genome` et `Chromosome`, et la note associée dans la classe `Fitness`. Ces deux dernières caractéristiques (information génétique et fitness), fortement liées, sont regroupées au sein d'une même structure, un `Individual`, structure de base manipulée par la `Population`. Une dernière partie, l'`Evaluator` a pour but d'attribuer une note au `Genome`.

L'`Evaluator` a été dérivé en un simulateur générique simple appelé `Simulator`. Celui-ci contient des `Agents` intéragissant avec leur environnement au moyen de capteurs et d'effecteurs - ce sont les classes `Sensor` et `Effector` - la partie contrôle étant contenue dans `Architecture`. Ce simulateur de base, relativement simple, n'a pas pour objectif d'être un simulateur multi-agents complet. Il est suffisant pour les problèmes de contrôle de robots qui nous intéressent ici mais, au besoin, il pourrait ne constituer qu'une interface vers des simulateurs multi-agents plus performants (Guessoum, 1998).

Le principal intérêt de SFERES réside ainsi dans la possibilité de réutiliser des modules d'une expérience à une autre en ne changeant que le strict minimum. Outre le considérable gain de temps occasionné, cela permet également de réaliser des comparaisons dans des conditions rigoureusement identiques. Ainsi, dans les expériences que nous avons réalisées, le passage d'un type de contrôleur à un autre ne nécessite que le changement de la partie architecture et de la partie génome. L'algorithme de sélection (partie `Population` et `Individual`), la simulation (`Simulator` et `Agent`), les capteurs, les effecteurs, la fonction d'évaluation des performances (partie `Fitness`) sont inchangés.

Pour plus de détails sur SFERES, voir (Landau *et al.*, 2001a; Landau *et al.*, 2002). L'annexe B présente un exemple de fichier de configuration.

6.2 Réseaux de neurones

6.2.1 Graphes et NeuroSFERES

Cette section présente NeuroSFERES, un framework intégré à SFERES pour l'évolution de réseaux de neurones. Ce framework est constitué de deux parties : une partie réseau de neurones et une partie développement de graphes qui permet de concevoir un graphe par évolution.

NeuroSFERES

Réseau. Le réseau de neurones est une classe dérivée de la classe `Architecture`. C'est un graphe orienté dont les noeuds sont les neurones et les liens sont les connexions. Il contient une liste de neurones et une liste de connexions. Le développement du réseau n'est pas décrit dans cette classe qui ne se charge que du fonctionnement "normal" du réseau, autrement dit de la propagation de l'information dans le réseau. Un réseau de neurones adapté devra être dérivé de la classe de base pour chaque type de développement : réseau à structure fixe, réseau à développement instantané depuis un graphe obtenu par évolution.

Éléments du réseau. Les neurones et les connexions présentent plusieurs points communs. Tous disposent d'entrées et de sorties[35], ainsi que d'une activation et d'une valeur de sortie. Nous avons vu précédemment à quoi correspondait l'activation et la sortie d'un neurone. Pour une connexion, c'est sensiblement la même chose : l'activation est la somme des entrées (généralement limitée à une entrée) et la sortie est une fonction de l'activation, le cas le plus courant étant une fonction linéaire du type $f(x) = w \times activation$ (w étant le poids de la connexion). Ces attributs sont regroupés dans la classe de base `Unit`.

La fonction d'activation et la fonction de transfert sont des méthodes d'une classe dédiée. Cela permet de changer facilement la partie fonctionnelle des connexions ou des neurones sans avoir à modifier quoi que ce soit d'autre.

La distinction entre neurones et connexions tient aux faits que les neurones ont, en entrée et en sortie, une liste de connexions et que, inversement, les connexions ont une liste de neurones en entrée et en sortie.

Un neurone d'entrée du réseau aura une fonction d'activation particulière qui ne calculera pas la somme des entrées du neurone, mais qui ira lire une entrée du réseau, et un neurone de sortie aura une fonction de transfert qui ira écrire la valeur de sortie du neurone dans la sortie

[35] Même si les connexions n'ont généralement qu'une seule entrée et une seule sortie.

correspondante du réseau. Les neurones d'entrée ou de sortie peuvent très bien être remplacés par des connexions d'entrée ou de sortie[36].

Graphes

Un module de SFERES permet la manipulation de graphes. Un graphe est une classe indépendante des autres classes citées précédemment. Il contient une liste de noeuds et une liste de liens, ainsi que des méthodes de manipulations de ce type de structure. Les noeuds et les liens peuvent être orientés ou non. Le graphe n'est ici, en fait, qu'une structure intermédiaire entre le génotype et le phénotype.

Développement de réseaux de neurones

Le développement de réseaux de neurones depuis un génome se passe en deux temps : on exprime le génome pour obtenir le graphe correspondant, puis on traduit le graphe en un réseau de neurones. Cette technique permet de travailler sur le développement de graphe et non directement sur les réseaux de neurones. Le génome contient une description de la structure du graphe avec les paramètres associés à chaque noeud et chaque lien. Il est utilisé pour obtenir un graphe équivalent qui peut ensuite être traduit vers une structure utilisable, un réseau de neurones dans notre cas. Cette procédure en deux étapes nécessite un surplus d'instructions de traduction. Cependant, une fois l'étape de traduction depuis un graphe vers un réseau de neurones achevée, elle peut être réutilisée pour toutes les techniques de développement de réseau. Le travail à réaliser consiste alors à reprogrammer uniquement le passage du génome vers le graphe, ce qui ne nécessite pas plus de travail qu'un développement direct et offre l'avantage d'être applicable à d'autres problèmes que l'évolution de réseaux neuronaux. Les techniques développées peuvent ainsi techniquement s'appliquer à des problèmes nécessitant la génération automatique de graphes, comme la génération d'automates à états finis ou encore de morphologie de robots, par exemple.

Cet environnement nécessite cependant un développement "instantané" du réseau dans lequel l'interaction avec l'environnement n'entre pas en compte, ce qui est le cas de la majorité des approches existantes actuellement.

Cet outil a été utilisé comme base pour l'implémentation de ModNet afin de réaliser les expériences décrites par la suite. Il a été implémenté en C++. Les fichiers de configuration et les traces générées par le déroulement d'une expérience sont en XML et des scripts PERL permettent de récupérer les informations intéressantes.

[36] il suffit d'affecter à ces connexion la fonction d'activation ou de transfert adéquate.

Chapitre 7

Le pendule inversé

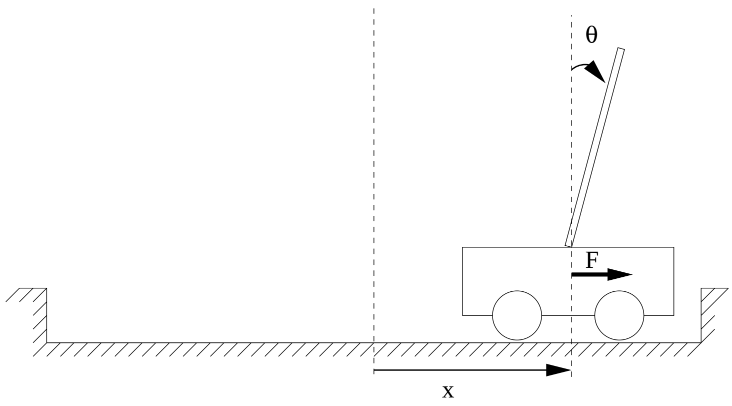


FIG. *7.1: Pendule inversé*

La stabilisation d'un pendule inversé est un problème classique dans le domaine du contrôle (Anderson et Miller, 1990; Wieland, 1991; Geva et Sitte, 1993). Un pendule est monté sur un chariot susceptible de se déplacer sur un axe (figure 7.1). Le pendule peut tourner autour d'un axe et l'objectif est de le maintenir dans une position verticale et dirigée vers le haut en faisant bouger uniquement le chariot dans certaines limites.

Si ce problème n'est manifestement pas lié aux robots volants, il appartient néanmoins à la même catégorie. Il a servi et sert encore de banc de test et a été traité à de nombreuses reprises avec des contrôleurs de types différents : contrôleurs flous (Karr, 1991; Cooper et Vidal, 1994) ou réseaux de neurones (Whitley *et al.*, 1995; Michel *et al.*, 1997; Pasemann et Dieckmann, 1997b; Gomez et Miikkulainen, 2002; Stanley et Miikkulainen, 2002a) notamment.

De plus, ce problème offre des variantes de complexité croissante. Les contrôleurs peuvent utiliser en entrée les erreurs et leur dérivée sur les deux degrés de liberté, à savoir x et θ, ou ils peuvent n'utiliser que les erreurs, sans les dérivées, ce qui complique le problème. C'est cette variante que nous utiliserons ici. Il est également possible d'étudier le cas de plusieurs pendules de caractéristiques différentes montés sur le même chariot (Whitley *et al.*, 1995) ou d'étudier le cas de pendules avec jointures (Cooper et Vidal, 1994). Nous n'étudierons pas ces versions plus complexes dans ce mémoire afin de nous concentrer, dans les chapitres suivants, sur les appareils volants.

Pour simuler le comportement du pendule inversé, nous avons utilisé le système d'équations suivant :

$$\ddot{\theta} = \frac{g\sin\theta + \frac{\cos\theta}{m_c+m} * \left(F + ml\dot{\theta}^2\sin\theta\right)}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c+m}\right)} \tag{7.1}$$

$$\ddot{x} = \frac{F + ml\left(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right)}{m_c+m} \tag{7.2}$$

avec, comme paramètres :

- longueur du pendule : $l = 0,5m$
- masse du pendule : $m = 0,1kg$
- masse du chariot : $m_c = 1,0kg$
- force maximale : $f_{max} = 10,0N$
- écart maximal autorisé pour le pendule par rapport à la verticale : $\theta_{max} = 0,2rad$
- écart maximal du chariot par rapport au centre : $x_{max} = 2m$

Nous avons intégré les équations selon la méthode d'Euler avec une constante de temps de $0,01s$.

Le problème du contrôle d'un pendule simple se résolvant très simplement à partir d'un correcteur de type PD (Proportionnel Dérivé), nous avons testé notre algorithme en ne fournissant en entrée que les positions x et θ, et non leurs dérivées[37].

Sachant que la dérivée est indispensable à la résolution de ce problème[38], ces expériences ont pour but de vérifier si l'évolution est capable de découvrir une solution implémentant un tel calcul par elle-même.

7.1 Conditions expérimentales

L'évaluation se fait sur deux expériences d'une minute chacune, pendant lesquelles on introduit des perturbations sur x et θ toutes les 20 secondes en moyenne. Les perturbations sur θ restent en valeur absolue inférieures ou égales à $0,14$ rad et celles sur x à 1 m, sachant que les limites autorisées sont de $0,2$ rad pour θ et 2 m pour x. Ces perturbations sont équivalentes à un test du fonctionnement du système relancé dans des conditions initiales différentes. Bien que non physiquement plausibles, elles permettent de s'assurer de la robustesse et de la stabilité des contrôleurs générés sans forcément réinitialiser le réseau de neurones.

La fitness est composée d'une partie entière et d'une partie décimale :

[37] Si les dérivées sont disponibles, une connexion directe de poids approprié des entrées vers la sortie suffit à résoudre le problème.

[38] Sans dérivée, le pendule peut, au mieux, osciller avec une amplitude constante, voir section 2.1.4.

$$f(g) = p(g) + fit_{dec}(g)$$

g étant un génoype.

La partie entière correspond au pourcentage de temps $p(g)$ pendant lequel le pendule a été maintenu (100 indique donc que le pendule n'est pas tombé pendant l'évaluation). La partie décimale indique l'erreur moyenne à chaque pas de temps sur chaque degré de liberté. Elle est obtenue par la formule suivante :

$$\begin{aligned}
\epsilon_\theta(g) &= \frac{\sum_{t=1}^{T} \epsilon_{norm\theta}(t,g)}{T} \\
\epsilon_x(g) &= \frac{\sum_{t=1}^{T} \epsilon_{normx}(t,g)}{T} \\
fit_{dec}(g) &= \frac{(1-\epsilon_\theta(g)) + (1-\epsilon_x(g))}{2}
\end{aligned}$$

ϵ_{norm} étant l'erreur normée par la valeur maximale qu'elle est susceptible de prendre. L'erreur sur θ et sur x est mesurée par différence entre la valeur réelle et 0. La fitness maximale est donc de 101 pour une expérience pendant laquelle le pendule a été maintenu pendant toute l'évaluation à la position souhaitée - c'est-à-dire $\theta = 0$, $x = 0$ - ce qui correspond à un pendule vertical, le chariot étant au centre de l'intervalle donné. De même, une fitness de $100,5$ correspond à un pendule qui n'est pas tombé pendant les évaluations réalisées et dont l'écart moyen sur θ et sur x est de $0,5$[39], ce qui signifie que, en moyenne, le pendule est resté à $\frac{\theta_{max}}{2}$ et le chariot à $\frac{x_{max}}{2}$[40]. Cette fonction de fitness récompense en priorité les contrôleurs capables de maintenir le système à l'intérieur des limites imposées pendant toute la durée de l'évaluation. Lorsque plusieurs individus en sont capables, l'individu qui aura la meilleure note sera celui qui maintiendra le système au plus proche de son point de repos. Cela favorise les réseaux de neurones stables effectuant un contrôle rapide.

L'algorithme de sélection utilisé opère sur une population de 100 individus. À chaque génération, il conserve 60% des meilleurs et remplit la population à partir de ces individus. Initialement, cependant, on crée une population 100 fois plus importante pour ne garder que les 100 meilleurs individus à partir desquels l'algorithme évolutionniste est lancé. Cette initialisation permet de réduire l'écart-type entre les différentes expériences d'une même série.

Contrairement à Whitley et al. (1995), nous ne faisons qu'un seul rafraîchissement du réseau par pas de temps. Cela évite au réseau d'avoir des entrées constantes la majeure partie du temps, ce qui pourrait fausser les calculs éventuels de dérivée ou les calculs de différence temporelle réalisés au sein du réseau. Les entrées fournies au réseau, à savoir x et θ, sont divisées par leurs valeurs maximales, à savoir respectivement 2m et 0,2rad.

[39] ou toute combinaison des deux erreurs permettant de générer une moyenne de $0,5$ sur ces deux degrés de liberté.

[40] C'est une valeur moyenne. En pratique, si le pendule n'est pas tombé, il a dû osciller car il ne peut pas rester à une position (θ, x) constante en dehors des couples pour lesquels θ vaut 0 ou π.

7.2 Expériences sans connaissance a priori

Nous avons lancé plusieurs séries d'expériences qui diffèrent seulement par le réservoir initial de modules donné à l'évolution. Dans les cas décrits ici, les modules mis à la disposition de l'évolution ne comptent qu'une entrée et une sortie. La première série, la série témoin, ne contient qu'un seul module qui implémente une connexion directe entre l'unique entrée du module et son unique sortie (module appelé par la suite module "P", pour proportionnel, car la sortie du module est proportionnelle à son entrée[41]). La deuxième série contient un seul module dont la structure est susceptible d'évoluer au cours des générations. Nous avons pour cela utilisé un codage direct des modules, avec la capacité d'ajouter ou de supprimer des neurones et des connexions avec une probabilité faible de 10^{-4}. Pour la génération aléatoire, on génère un nombre de neurones cachés variant entre 0 et 3 et un nombre de connexions variant entre 0 et 6. Les paramètres des connexions utilisées dans ces modules sont les mêmes que pour les autres modules, à savoir des chaînes binaires de 8 bits, et ont un taux de mutation identique de $0,1$.

Les neurones utilisés ont une fonction de transfert de type $tanh$ (fonction sigmoïde).

7.2.1 Statistiques

Nous avons réalisé vingt expériences dans chaque série. Les paramètres de ces séries d'expériences sont rigoureusement identiques, seul change le réservoir de modules disponibles.

Sur les 20 expériences lancées, 12 ont convergé après 2000 générations dans les expériences avec modules P uniquement, alors que 20 ont convergé dans les expériences avec modules à structure évoluable (figure 7.2). La progression de la fitness est également nettement en faveur de la deuxième série d'expérience, laquelle, de plus, présente la variabilité la plus faible. La première série reste bloquée à des fitness faibles avec une variabilité importante (figure 7.3).

Les expériences utilisant des modules se révèlent donc beaucoup plus performantes que les expériences témoin. Toutes parviennent à empêcher le pendule de tomber en moins de 400 générations (à la $60^{ème}$ génération, 75% des expériences empêchent le pendule de tomber).

7.3 Étude de solutions de la première série

La meilleure expérience de la première série a obtenu la note finale de 100,824. Si le réseau correspondant (figure 7.4) parvient à maintenir le cartpole à l'intérieur des limites imposées pendant la durée d'une évaluation, il ne parvient pas à stabiliser le pendule en position centrale et verticale (figure 7.5) : l'angle oscille de façon importante avec une amplitude constante en moyenne. La position du chariot oscille également, mais avec une période plus élévée et, surtout, avec une amplitude s'atténuant lentement. Le fait que le système ne soit pas stabilisé indique que le réseau n'est pas parvenu à calculer de dérivée.

Les solutions générées par Pasemann et Dieckmann (1997b) ont les mêmes caractéristiques : aucune n'a réussi à supprimer ces oscillations dont l'amplitude reste constante tout au long de

[41]Si l'on reste dans la zone linéaire de fonctionnement des neurones.

Avec module P uniquement

génération	min	médiane	max	convergées
20	7,72	45,46	100,79	4
100	8,69	84,56	100,81	7
500	9,11	89,98	100,81	9
1000	9,11	100,64	100,81	12
2000	9,11	100,65	100,82	12

Avec un module dont la structure est soumise à évolution

génération	min	médiane	max	convergées
20	81,18	88,18	100,85	8
100	84,50	100,81	100,86	16
500	100,70	100,84	100,94	20
1000	100,70	100,84	100,94	20
2000	100,79	100,86	100,94	20

FIG. *7.2: Statistiques des résultats des expériences avec module P seulement et avec module de structure variable. Dans chaque série, 20 expériences ont été réalisées. "convergées" représente le nombre d'expériences dans lesquelles le meilleur individu est capable de maintenir le pendule dans les limites imposées pendant toute la durée de l'expérience. Les valeurs des fitness inférieures à 100 indiquent le pourcentage du temps de l'évaluation pendant lequel le pendule a été maintenu. Les valeurs supérieures à 100 indiquent que le pendule a été maintenu dans les limites autorisées pendant toute l'évaluation ; la partie décimale correspond à* $1 - écart\_normé\_moyen$ *sur les deux degrés de liberté (x et theta) : plus il est proche de 1, plus le contrôle est efficace.*

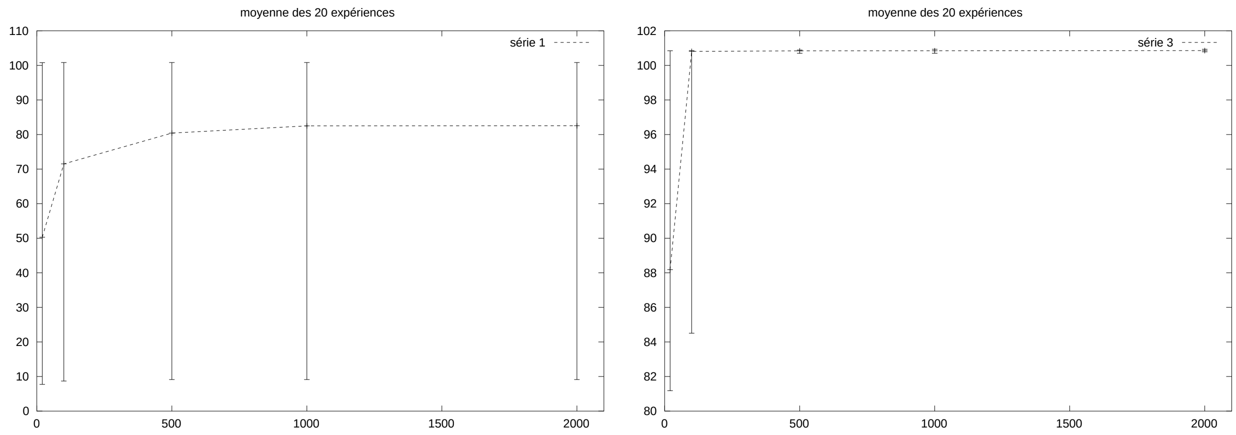


FIG. *7.3: Évolution de la fitness moyenne au fil des générations pour les deux premières séries d'expériences. Série 1 : modules P uniquement. Série 2 : module dont la structure est soumise à évolution. Les générations sont représentées sur l'axe des abscisses et les fitness sur l'axe des ordonnées.*

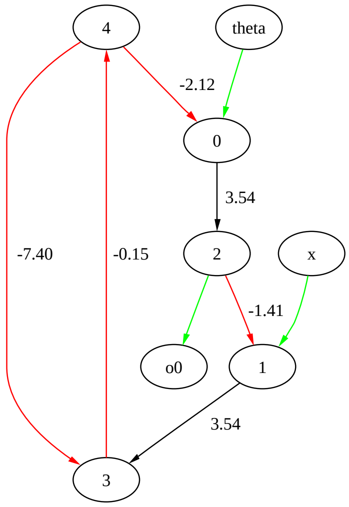


FIG. *7.4: Meilleur réseau obtenu par évolution pour le contrôle d'un pendule inversé simple après 2000 générations dans une expérience de la première série : le réservoir de modules ne contient qu'une connexion simple (module P). Les informations disponibles en entrée sont l'angle θ instantané ainsi que la position du chariot x. Les connexions en gris dont le poids n'est pas indiqué sont les connexions depuis les entrées ou vers les sorties : elles ont un poids unitaire.*

l'évaluation. La plupart des auteurs qui se sont intéressés à la question ne prennent en considération que les chutes et négligent la stabilité du contrôle. Ce point, pourtant, nous semble intéressant car il permet de distinguer les solutions stables des solutions instables. Ces dernières peuvent, certes, empêcher le pendule de tomber pendant le temps donné, mais peuvent égale-

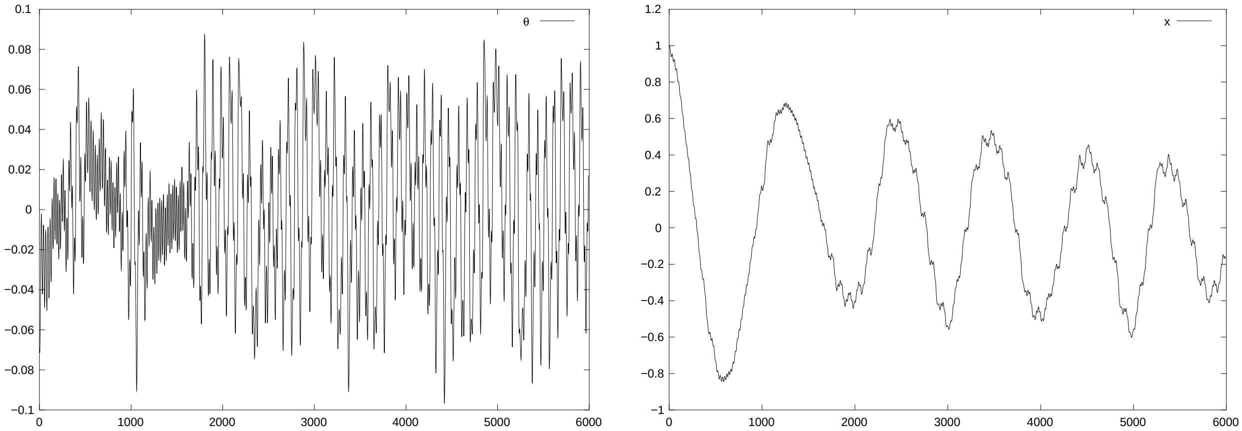


FIG. *7.5: Comportement du pendule inversé contrôlé par un réseau obtenu par évolution après 2000 générations. Expérience de la première série (modules P disponibles uniquement). L'angle θ est en radians, x est en mètres. Les abscisses sont en pas de temps (un pas de temps représente 0,01s). Le système est initialement placé à une position de $\theta = -0,07rad$ $x = 1m$.*

ment le laisser tomber peu après[42]. Whitley et al. (1995), par exemple, ne donnent pas de détails quant au comportement des solutions qu'ils ont obtenues (ils ont vérifié cependant les capacités de généralisation et de stabilité de ces solutions).

Les différents réseaux que nous avons observés présentent plusieurs caractéristiques communes (figures 7.6 et 7.4). Tout d'abord, tous disposent d'une boucle d'une taille de trois à cinq neurones. De plus, dans tous les réseaux observés, le capteur de x est relié au neurone de la boucle le plus éloigné de la sortie, alors que le capteur de θ est relié au neurone de la boucle qui est directement relié à la sortie. Cela signifie que l'action de x sur la sortie a un décalage de 3 à 5 pas de temps, alors que l'action de θ est beaucoup plus directe (2 neurones intermédiaires traversés dans tous les cas). De plus, l'influence du capteur de x sur la sortie sera d'autant plus faible que des connexions de poids inférieurs à 1 seront traversées avant d'atteindre la sortie. De ces observations, on peut déduire que l'évolution a logiquement donné la priorité au maintien de θ, indispensable pour la "survie" du pendule, le maintien de x n'intervenant qu'en second.

Nous avons testé les capacités de généralisation de ces différents réseaux (figure 7.7). Les contrôleurs ont été testés pour des positions initiales variant de $-1,8m$ à $+1,8m$ par incrément de $0,2$ pour x et de $-0,17rad$ ($-10deg$) à $+0,17rad$ ($+10deg$) par incréments de $0,034rad$ ($2deg$) pour θ. Pour chaque contrôleur, nous avons relevé les capacités de généralisation sous la forme d'un pourcentage d'évaluations réussies[43]. Les capacités de généralisation de ces réseaux sont faibles : 34%, 68%, 52% et 67%.

[42]Nous avons observé ce comportement à plusieurs reprises pour des fitness de l'ordre de 100,7.

[43]Une évaluation est réussie si le pendule ne tombe pas

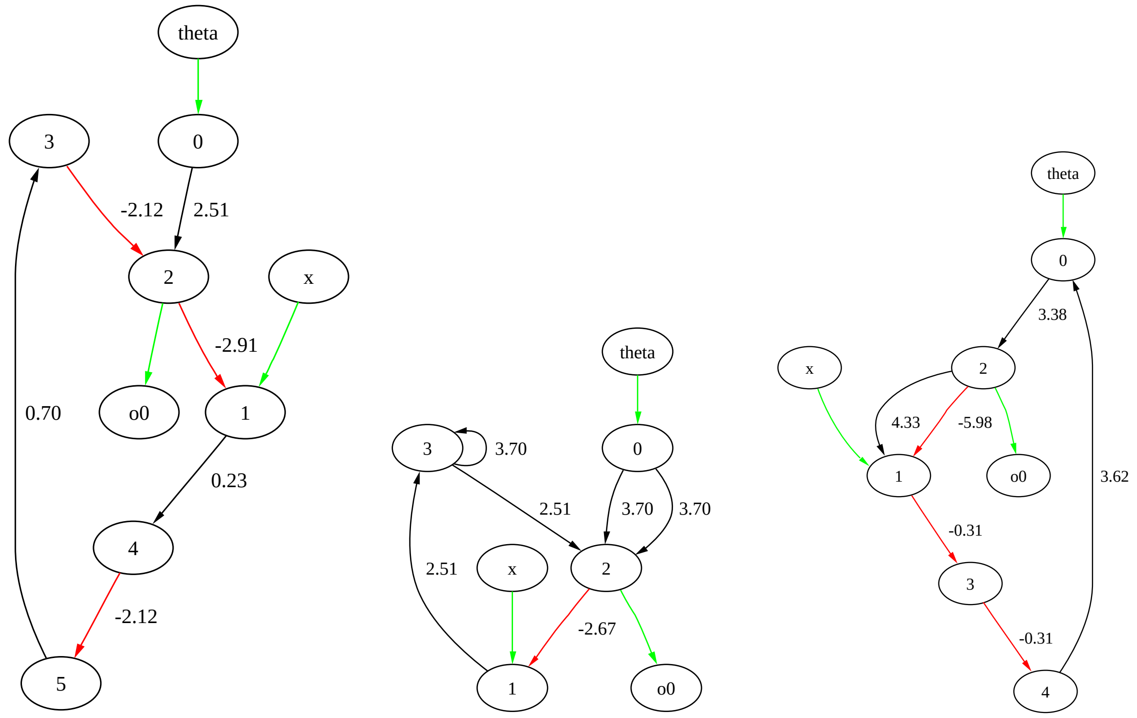


FIG. *7.6: Autres réseaux obtenus par évolution pour le contrôle d'un pendule inversé simple après 2000 générations (expériences de la première série : le réservoir de modules ne contient qu'une connexion simple (module P)). Les réseaux sont issus des meilleurs individus des expériences ayant obtenu les meilleures fitness.*

7.4 Étude de solutions de la deuxième série

Les expériences de la deuxième série ont un taux de réussite beaucoup plus élevé que la série témoin : 100% des expériences ont convergé. De plus, les fitness atteintes sont beaucoup plus élevées : 100,94, contre 100,82 pour la première série. Si l'écart entre les deux n'est que $0,12$, ce qui est faible en apparence, il correspond en fait à une différence importante dans le comportement (figure 7.8) : alors que le pendule n'était pas stabilisé dans les expériences témoin, dans cette expérience le pendule est très rapidement ramené à la position désirée : $x = 0$ et $\theta = 0$.

Le meilleur individu obtenu lors des expériences de cette série est ainsi capable de parfaitement maintenir le pendule dans la position désirée (figure 7.8), rapidement et sans la moindre oscillation. Le réseau correspondant (figure 7.9), ainsi que les autres bons réseaux observés (figure 7.10), contiennent tous des structures “D” permettant de calculer une différence temporelle. Cette structure contient un neurone caché et une connexion reliant directement l'entrée à la sortie. Lorsque les signes des poids sont correctement positionnés, cela permet de faire la différence entre le signal correspondant à un pas de temps et le signal du pas de temps précédent (nous y reviendrons à la section 7.5). À chaque fois, la structure “D” est incluse dans un module (exemple figure 7.11), plutôt que répartie sur plusieurs modules.

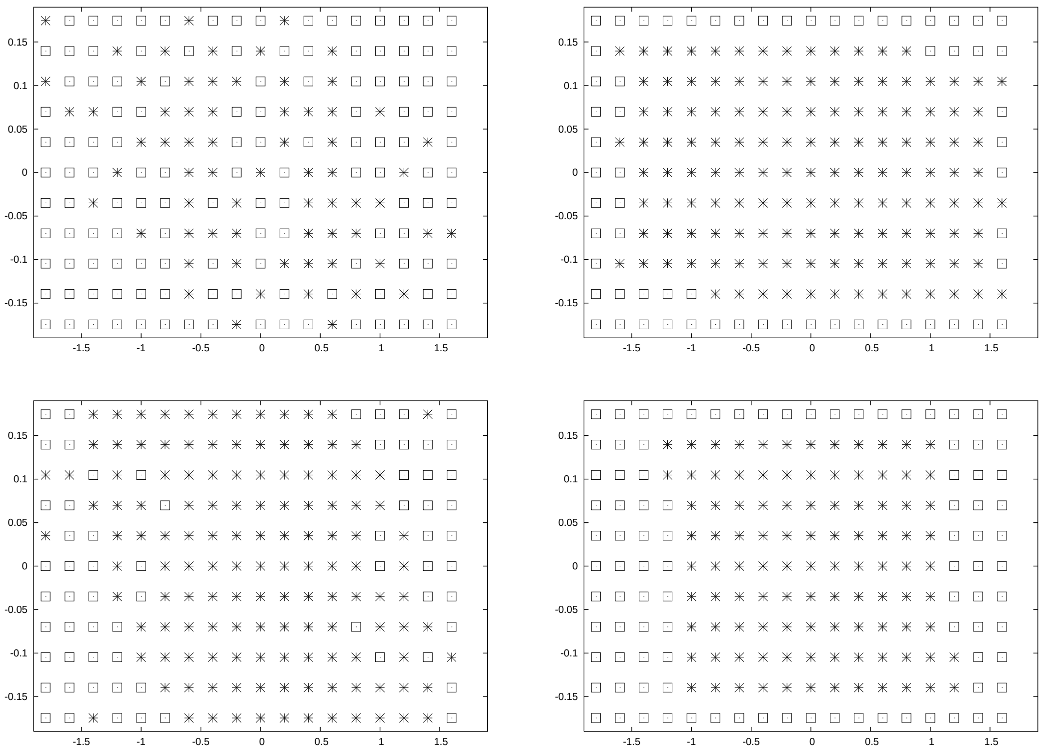
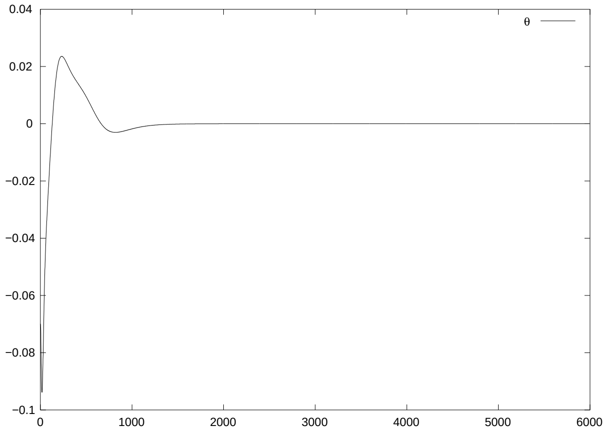


FIG. 7.7: *Capacité de généralisation des meilleurs réseaux de la première série (en haut à gauche, réseau de la figure 7.4, ensuite réseaux de la figure 7.6). Les points indiquent la réussite (étoile) ou l'échec (carré) du contrôle pendant une expérience de deux minutes, la position initiale étant donnée par les coordonnées du point (x en abscisse en mètres et θ en ordonnée en radians).*

Les capacités de généralisation sont, à une exception près, nettement supérieures aux expériences de la série témoin (figure 7.12).

Ainsi, dans ces deux séries d'expériences, dont aucune n'exploite de connaissances a priori, la simple manipulation de modules permet à l'évolution de converger plus vite et vers des solutions bien plus efficaces que lorsque les modules ne sont pas disponibles. Cette différence est liée au croisement qui permet de propager rapidement au reste de la population les modules efficaces découverts par un des individus. En effet, dans les expériences de la première série, si un individu découvre la structure 'D', il est dans l'incapacité de la transmettre par croisement à d'autres individus. De plus, cette structure n'étant pas protégée dans un module, elle pourra facilement être détruite par les différentes mutations structurelles.

Pasemann et Dieckmann, dont nous nous sommes inspirés pour déterminer les conditions de ces expériences, ne donnent pas de détails quant à la vitesse de convergence de leur algorithme (Pasemann et Dieckmann, 1997b). Sans donner en entrée la vitesse, leur algorithme n'a pas réussi à générer de réseau stabilisant complètement le pendule.



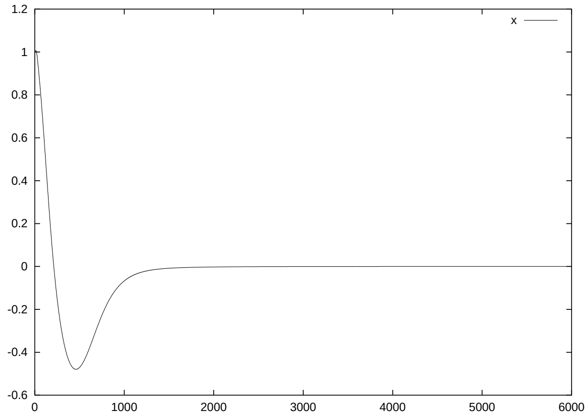


FIG. *7.8: Comportement du pendule inversé contrôlé par un réseau obtenu par évolution après 2000 générations. Expérience de la deuxième série (modules dont la structure est soumis à évolution uniquement). En ordonnée, l'angle θ est en radians, x est en mètres. Les abscisses sont en pas de temps (un pas de temps représente 0,01s). Le système est initialement placé à une position de $\theta = -0,07rad$ $x = 1m$.*

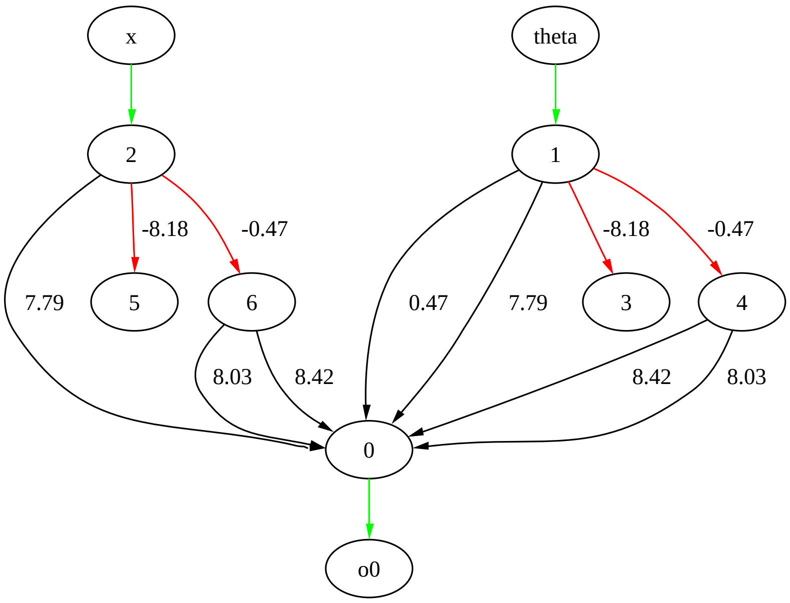


FIG. *7.9: Meilleur réseau obtenu par évolution pour le contrôle d'un pendule inversé simple après 2000 générations dans une expérience de la deuxième série : le réservoir de modules contient un seul module dont la structure est susceptible de varier au cours de l'évolution. Les informations disponibles en entrée sont l'angle θ instantané ainsi que la position du chariot x. Les connexions en gris dont le poids n'est pas indiqué sont les connexions depuis les entrées ou vers les sorties, elles ont un poids unitaire.*

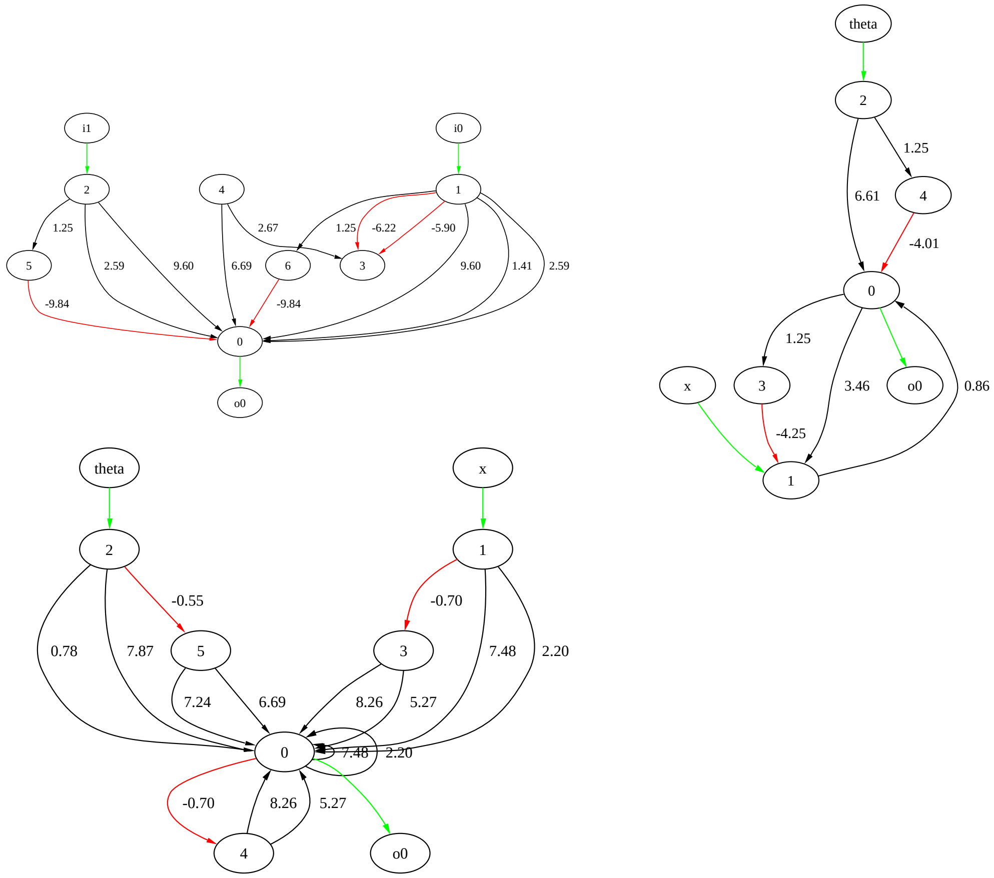


FIG. *7.10: Autres réseaux obtenus par évolution pour le contrôle d'un pendule inversé simple après 2000 générations (expériences de la deuxième série : le réservoir contient un seul module dont la structure est susceptible de varier au cours de l'évolution). Les réseaux sont issus des meilleurs individus des expériences ayant obtenu les meilleures fitness.*

7.5 Expériences avec connaissance a priori

La troisième série d'expériences contient, en plus des modules P, des modules "I" et "D" dont la structure (figure 7.13) peut permettre l'émergence de fonctions de type intégration ou dérivation si les poids des connexions sont ajustés convenablement. Ici, seule la structure des modules est donnée, les poids devant être obtenus par évolution. L'appellation module D ou I est donc quelque peu abusive car, avant évolution, ces modules n'implémentent en rien des calculs de type dérivée ou intégrale. L'évolution a donc un double objectif dans cette série d'expériences : trouver un assemblage de modules convenable compte tenu de la tâche à résoudre et optimiser les paramètres du réseau (les poids des connexions) pour que le réseau soit efficace. Toutefois, dans la mesure où les poids des connexions ne sont pas contraints, les modules D ou I peuvent très bien être utilisés par l'évolution à des fins autres que le calcul d'une approximation de dérivée ou d'intégrale. La quatrième série d'expériences contient les mêmes modules que la série précédente,

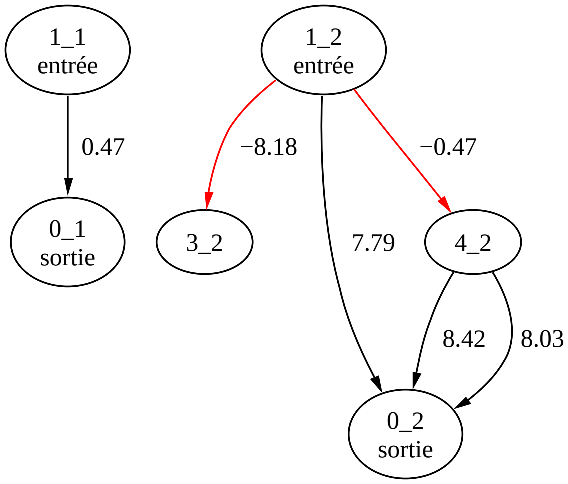


FIG. *7.11: Modules utilisés par le réseau représenté sur la figure 7.9.*

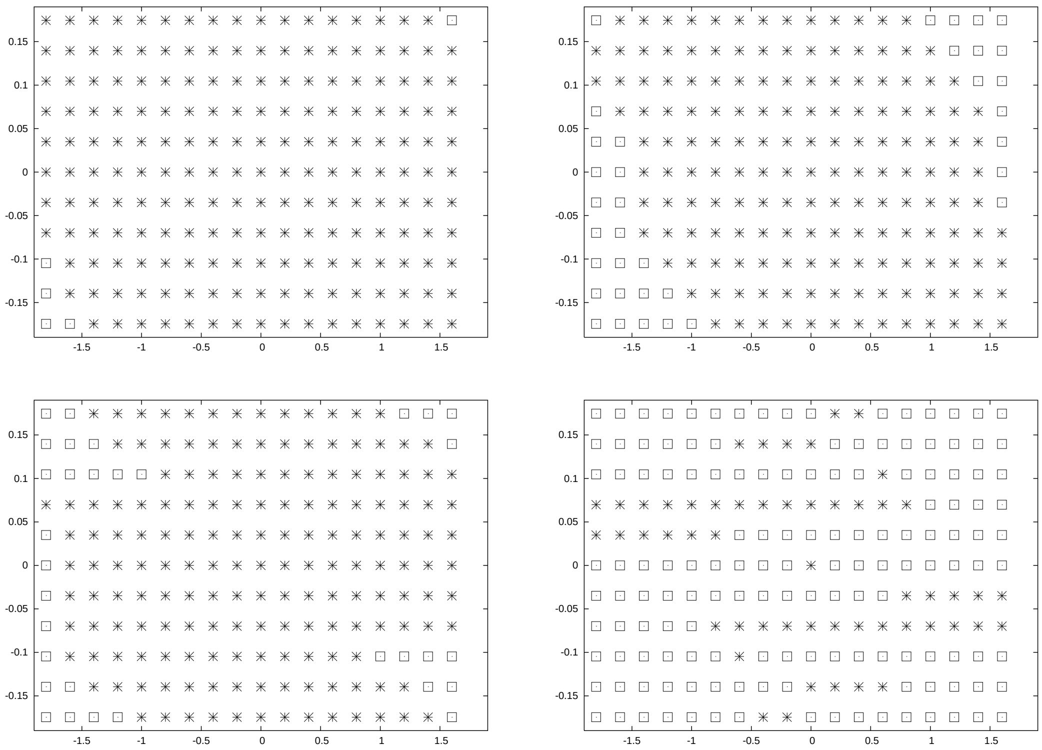


FIG. *7.12: Capacité de généralisation des meilleurs réseaux de la deuxième série (en haut à gauche, réseau de la figure 7.9, ensuite réseaux de la figure 7.10). Les points indiquent la réussite (étoile) ou l'échec (carré) du contrôle pendant une expérience de deux minutes, la position initiale étant donnée par les coordonnées du point (x en abscisse en mètres et θ en ordonnée en radians). Les taux de réussite sont de 97%, 84%, 82% et 27%.*

mais certains signes de poids de connexions des modules D et I ont été figés de façon à forcer les modules à avoir une fonctionnalité proche de ce pour quoi ils ont été définis, c'est-à-dire le calcul approximatif de dérivée ou d'intégrale (figure 7.14). Cela permet d'orienter l'évolution et de réduire l'espace de recherche.

D'après les résultats des expériences précédentes, nous savons que des modules D sont suffisant pour résoudre le problème. Cependant, dans la plupart des solutions présentées dans la littérature, apparaissent des connexions récurrentes comme celles qui sont présentes dans le module 'I' (Whitley *et al.*, 1995; Pasemann et Dieckmann, 1997b). Ces expériences nous permettront ainsi de voir si d'autres solutions, plus efficaces que celles découvertes précédemment, peuvent être générées en utilisant ce type de connexion.

Toutefois, dans la mesure où le module D est le plus fréquemment utilisé pour résoudre ce problème, nous avons également réalisé des expériences dans lesquelles nous n'avons utilisé qu'un seul module D dans le réservoir, soit avec des signes libres, soit avec des signes en partie imposés.

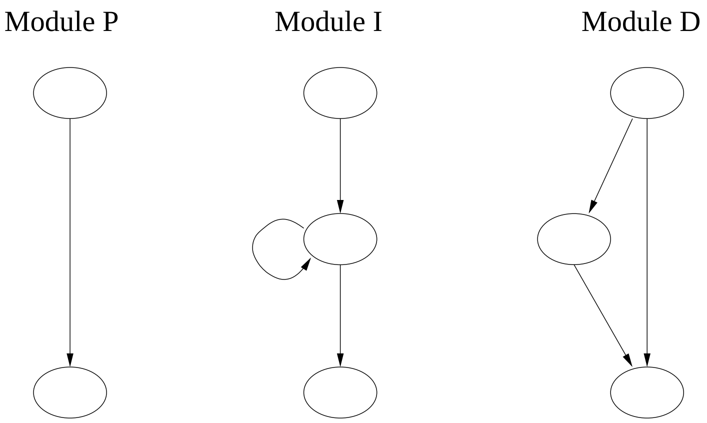


FIG. *7.13: Modules P, I et D. Le module P représente un bloc proportionnel. Le module I représente, au moins potentiellement, un bloc intégral (si le poids de la connexion récurrente est positif) et le module D représente, au moins potentiellement, un module dérivée (si un des poids est négatif et tous les autres positifs, par exemple).*

7.5.1 Statistiques

Dans ces séries d'expériences, les performances sont nettement supérieures à la série témoin (figure 7.15 et 7.16) et sont du même ordre que celles des expériences de la deuxième série. Les expériences avec modules P, I et D avec signes imposés convergent, cependant, nettement plus rapidement que les expériences de cette deuxième série. Les performances finales sont similaires. Les expériences n'utilisant qu'un seul module D ont un comportement similaire aux expériences utilisant des modules P, I et D avec signes imposés. Les expériences avec module D dont les signes sont imposés sont nettement supérieures aux autres expériences. Dès la $20^{ème}$ génération, toutes les vingt expériences ont généré des individus capables de maintenir le pendule. De plus la médiane est également très élevée. La fitness finale est également plus importante que dans

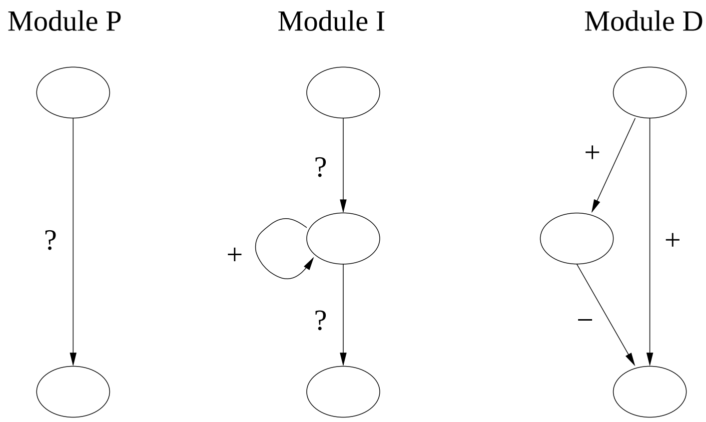


FIG. *7.14: Modules P, I et D dans lesquels les signes des poids de certaines connexions ont été imposés de façon à restreindre l'espace de recherche.*

les autres expériences. Ces résultats ne sont pas surprenants, car la solution idéale au problème est un assemblage de modules D dont les connexions ont des signes correspondant à ceux que nous avons imposés. Une grande partie des réseaux générés ayant toutes chances d'être efficace, l'évolution a pu concentrer son activité dans la zone de l'espace de recherche la plus prometteuse. À titre de comparaison, nous avons défini une architecture composée de deux PID reliant les deux capteurs à la commande du moteur. Nous avons fait évoluer les six paramètres de ce contrôleur avec une Stratégie Évolutionniste pour maintenir le pendule et le chariot. Toutes les expériences ont rapidement convergé vers une fitness optimale de 100,97.

7.5.2 Étude de solutions de la troisième série

Le comportement généré par la meilleure solution est similaire à ce qui a été obtenu dans la deuxième série d'expériences : le pendule est maintenu à la verticale avec le chariot centré (figure 7.17).

Le réseau permettant de générer ce comportement est très simple (figure 7.18) et, en particulier, il ne contient pas de récurrence (autre que celle du neurone intermédiaire du module 'I'). Il utilise deux modules différents : un module 'I' et un module 'D' apparaissant à deux reprises dans le réseau. Dans les deux cas, les poids ont été positionnés par l'évolution pour affecter aux modules la fonctionnalité pour laquelle ils ont été définis. Le module 'I', en effet, dispose d'une connexion récurrente positive - implémentant ainsi une approximation d'intégrale - et le module 'D' contient une connexion négative qui permet de soustraire, au niveau du neurone de sortie du module, le signal du pas de temps t et celui du pas de temps $t-1$ - implémentant ainsi une différence temporelle proche de la dérivée.

Les différents réseaux efficaces observés pour cette série d'expériences utilisent tous des modules 'D' avec un jeu de poids compatible avec le calcul de différence temporelle (figure 7.19).

Les capacités de généralisation des différents réseaux observés sont bien meilleures que pour les résultats de la première série et sont légèrement meilleurs que ceux de la deuxième série (figure

Avec modules P, I et D (signes libres)

gén	min	médiane	max	conv
20	61,10	90,13	100,84	8
100	84,30	100,71	100,88	15
500	85,75	100,80	100,91	19
1000	96,11	100,80	100,91	19
2000	96,11	100,83	100,92	19

Avec modules P, I et D (signes en partie imposés)

gén	min	médiane	max	conv
20	86,61	100,75	100,85	18
100	100,63	100,81	100,90	20
500	100,63	100,81	100,91	20
1000	100,72	100,84	100,93	20
2000	100,77	100,85	100,94	20

Avec modules D (signes libres)

gén	min	médiane	max	conv
20	84,49	100,76	100,82	18
100	100,72	100,82	100,87	20
500	100,75	100,84	100,95	20
1000	100,76	100,86	100,95	20
2000	100,77	100,88	100,95	20

Avec modules D (signes en partie imposés)

gén	min	médiane	max	conv
20	100,78	100,84	100,88	20
100	100,78	100,88	100,93	20
500	100,79	100,91	100,94	20
1000	100,79	100,91	100,95	20
2000	100,87	100,93	100,96	20

FIG. *7.15: Statistiques des résultats des expériences avec module P, I et D ou D seulement avec et sans signes imposés à différentes générations. Dans chaque série, 20 expériences ont été réalisées. La colonne "conv" représente le nombre d'expériences dans lesquelles le meilleur individu est capable de maintenir le pendule dans les limites imposées pendant toute la durée de l'expérience. Les valeurs de fitness inférieures à 100 indiquent le pourcentage du temps de l'évaluation pendant lequel le pendule a été maintenu. Les valeurs supérieures à 100 indiquent que le pendule a été maintenu dans les limites autorisées pendant toute l'évaluation, la partie décimale correspond à* $1 - écart\_normé\_moyen$ *sur les deux degrés de liberté* (x *et* θ) *: plus il est proche de 1, plus le contrôle est efficace.*

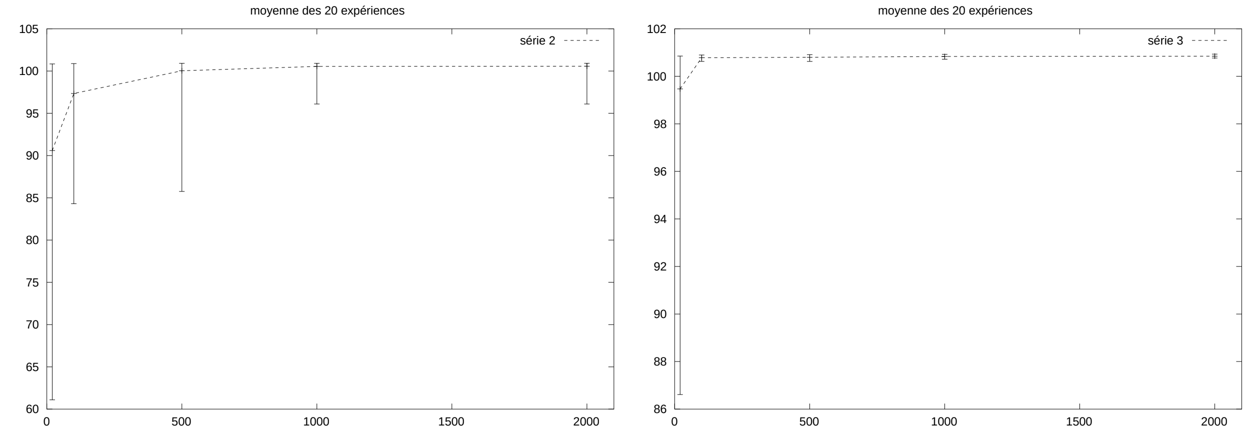


Fig. 7.16: Évolution de la fitness moyenne au fil des générations pour les séries d'expériences 3 et 4. Série 3 : modules P, I et D, série 4 : modules P, I et D avec signes partiellement imposés. Les générations sont représentées sur l'axe des abscisses et les fitness sur l'axe des ordonnées.

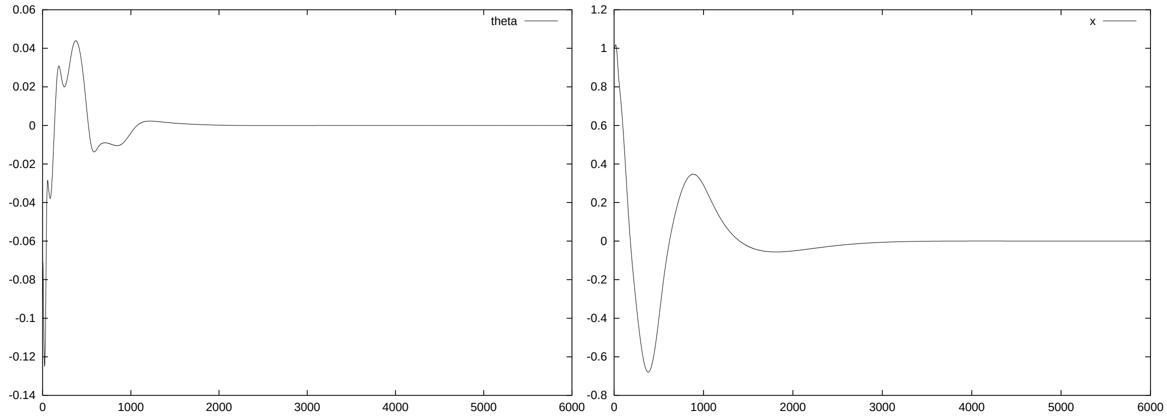


Fig. 7.17: Comportement du pendule inversé contrôlé par un réseau obtenu par évolution après 2000 générations. Expérience de la troisième série (modules P, I et D sans contrainte sur les connexions). En ordonnée, l'angle θ est en radians, x est en mètres. Les abscisses sont en pas de temps (un pas de temps représente $0,01s$). Le système est initialement placé à une position de $\theta = -0,07rad$, $x = 1m$.

7.20) : 87%, 89%, 73% et 94%.

7.5.3 Étude de solutions de la quatrième série

Les solutions étudiées dans cette section ont été obtenues après 2000 générations dans des expériences ayant pour réservoir initial des modules P, I et D dont les signes sont partiellement imposés.

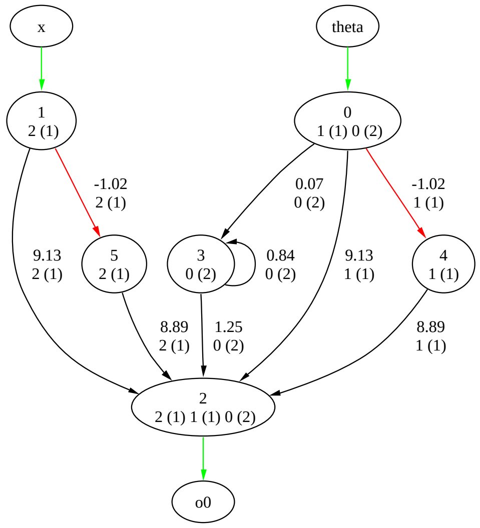


Fig. 7.18: *Meilleur réseau obtenu par évolution pour le contrôle d'un pendule inversé simple après 2000 générations dans une expérience de la troisième série : le réservoir de modules contient des modules P, I et D. Les poids des connexions évoluent entièrement. Les informations disponibles en entrée sont l'angle θ instantané ainsi que la position du chariot x. Les nombres en dessous du numéro de neurone ou en dessous du poids d'une connexion correspondent aux numéros de modules auquel le neurone ou la connexion appartiennent. Les nombres entre parenthèses indiquent le module source du réservoir : 0 pour un module P, 1 pour un module D et 2 pour un module I.*

Les meilleurs réseaux de cette série d'expériences sont plus efficaces que ceux de la troisième série. Ils parviennent à stabiliser le pendule plus rapidement et avec un dépassement plus faible (figure 7.21). Les réseaux correspondants (figure 7.22) utilisent principalement des modules D, mais aussi parfois des modules I. L'utilisation des modules I ne correspond pas, à une exception près, à la fonctionnalité initiale prévue pour ce module. Dans un cas, la connexion récurrente est nulle, dans l'autre, le module est utilisé pour créer une boucle récurrente. Les modules D sont utilisés de la même manière que dans la troisième série d'expériences, le capteur x étant toujours plus éloigné de la sortie du réseau que le capteur θ.

Les capacités de généralisation des réseaux sont encore plus élevées que pour les séries précédentes (figure 7.23) : 97,5%, 71%, 99% et 87%.

7.6 Discussion

L'utilité des modules, qu'ils soient prédéfinis ou évolués, apparaît clairement dans ces différentes séries d'expériences. Tous les réseaux efficaces utilisent plusieurs modules de type D. L'apparition de ce type de module nécessite à la fois de trouver la bonne structure, mais aussi le bon paramétrage. Dans les expériences de la première série, l'apparition d'un tel module est possible mais, statistiquement, peu de réseaux disposeront de cette structure. Parmi ces réseaux,

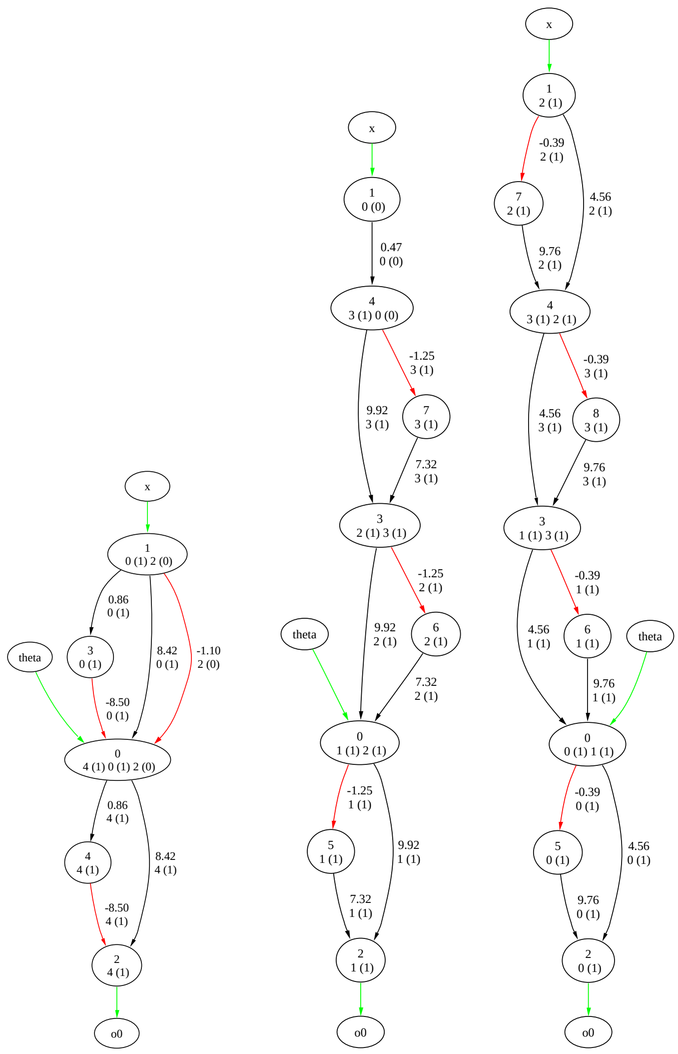


Fig. *7.19: Autres réseaux obtenus par évolution pour le contrôle d'un pendule inversé simple après 2000 générations (expériences de la deuxième série). Les réseaux sont issus des meilleurs individus des expériences ayant obtenu les meilleures fitness.*

il en restera encore moins qui présenteront un jeu de paramètres assurant la fonctionnalité qui convient, c'est-à-dire qui procure un avantage sélectif. Malheureusement, parmi ces derniers réseaux, qui ont donc un bon potentiel, mais dont la probabilité d'apparition est faible, beaucoup contiendront, en plus du module D, d'autres neurones et connexions capables de détériorer fortement les performances de l'ensemble et donc de faire rejeter une solution pourtant prometteuse.

L'évolution pourrait tout de même trouver la bonne solution si elle pouvait procéder de manière incrémentale. Une solution efficace doit pouvoir être trouvée par améliorations successives. Si certains éléments structurels de cette solution ne peuvent être obtenus progressivement - c'est-à-dire si un réseau de structure très proche obtient une mauvaise fitness, ne reflétant pas cette proximité structurelle - l'évolution, de même que tout autre algorithme de recherche dans ce cas,

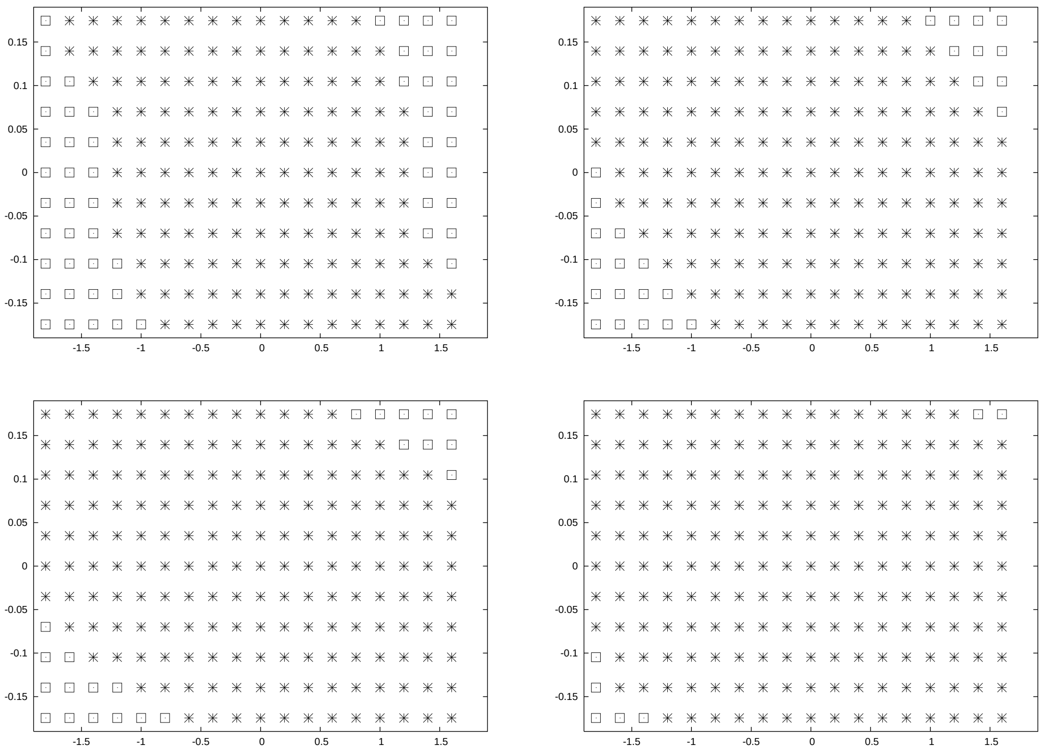


FIG. *7.20: Capacité de généralisation des meilleurs réseaux (en haut à gauche réseau de la figure 7.18, puis réseaux de la figure 7.19). Les points indiquent la réussite (étoile) ou l'échec (carré) du contrôle pendant une expérience de 2 minutes, la position initiale étant donnée par les coordonnées du point (x en abscisse en mètres et θ en ordonnée en radians).*

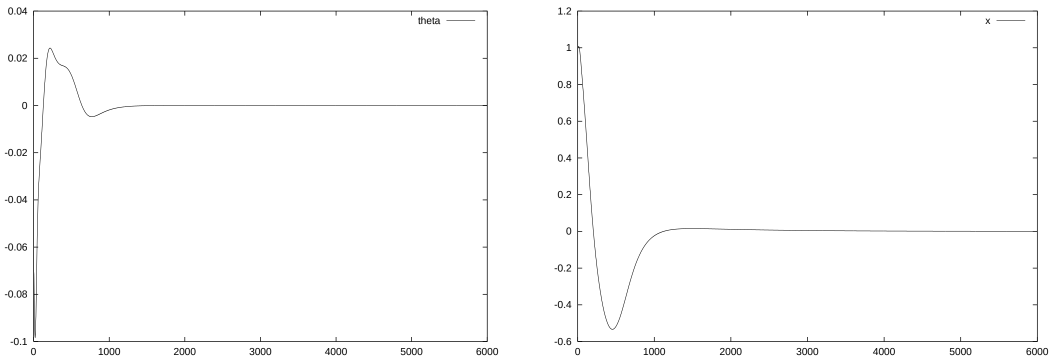


FIG. *7.21: Comportement du pendule inversé contrôlé par un réseau obtenu par évolution après 2000 générations (réseau de gauche de la figure 7.22). L'angle θ en radians, à x est en mètres. Les abscisses sont en pas de temps (un pas de temps représente 0.01s). Le système est initialement placé à une position de $\theta = -0.07rad$ $x = 1m$.*

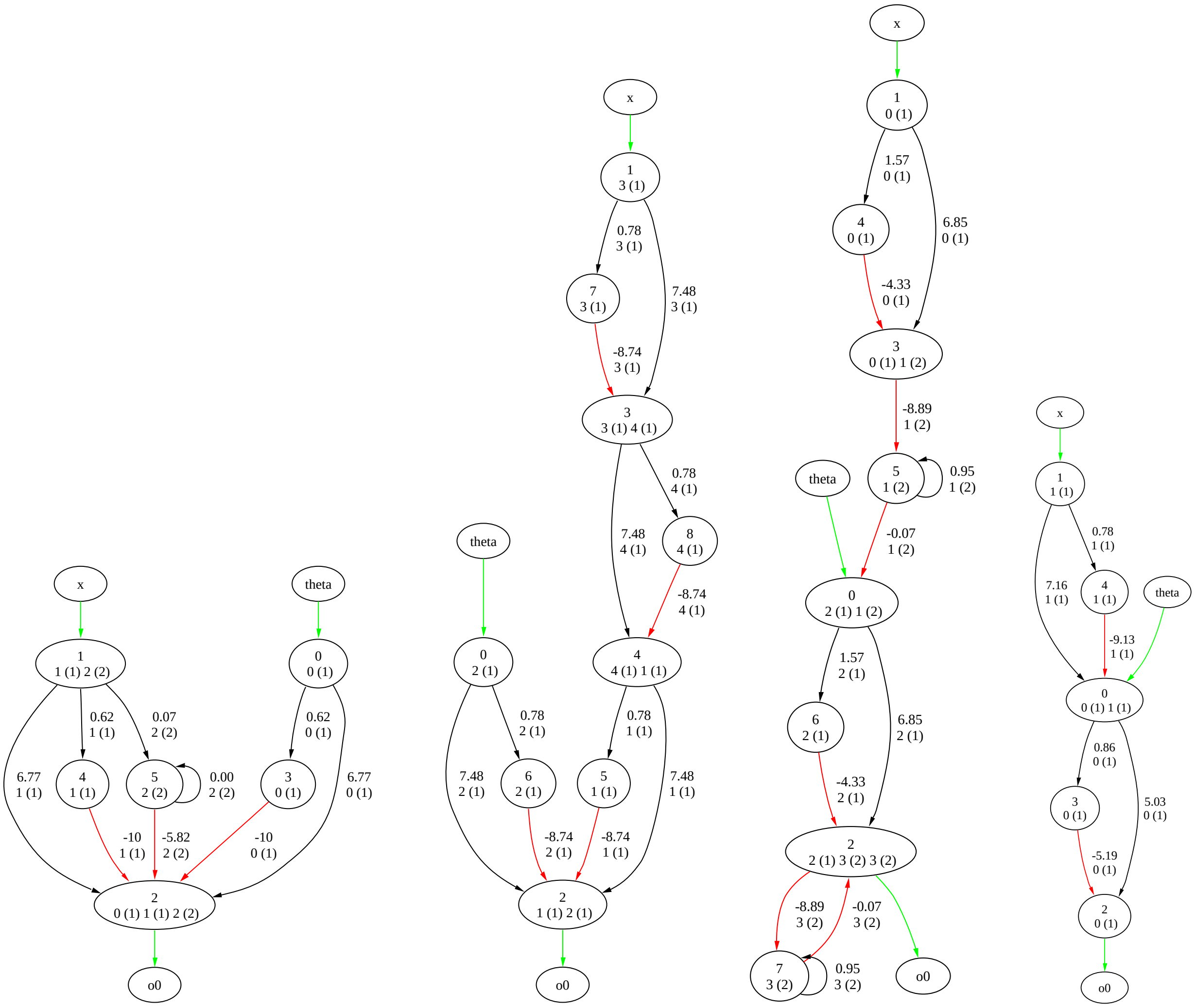


FIG. *7.22: Réseaux obtenus par évolution pour le contrôle d'un pendule inversé simple à l'issue d'une expérience de la troisième série (modules P, I et D avec poids partiellement imposés). Les informations disponibles en entrée sont l'angle θ instantané ainsi que la position du chariot x.*

ne pourra pas converger efficacement. De plus, les réseaux doivent pouvoir échanger des éléments fonctionnellement significatifs de leur génome. Lorsque nous donnons à l'évolution la possibilité d'utiliser des modules, nous lui permettons de protèger des élements structurels des mutations (les modules ayant un taux de mutation structurelle fixe très faible ou bien une structure fixe donnée à l'avance) et nous lui permettons également d'échanger ces éléments avec d'autres individus lors du croisement. Dans ces expériences, cela a permis de propager rapidement des modules susceptibles de calculer une dérivée au sein de la population.

Parmi les différentes séries d'expériences que nous avons réalisées, nous pouvons identifier plusieurs conditions donnant des résultats similaires. Ainsi les expériences avec modules soumis

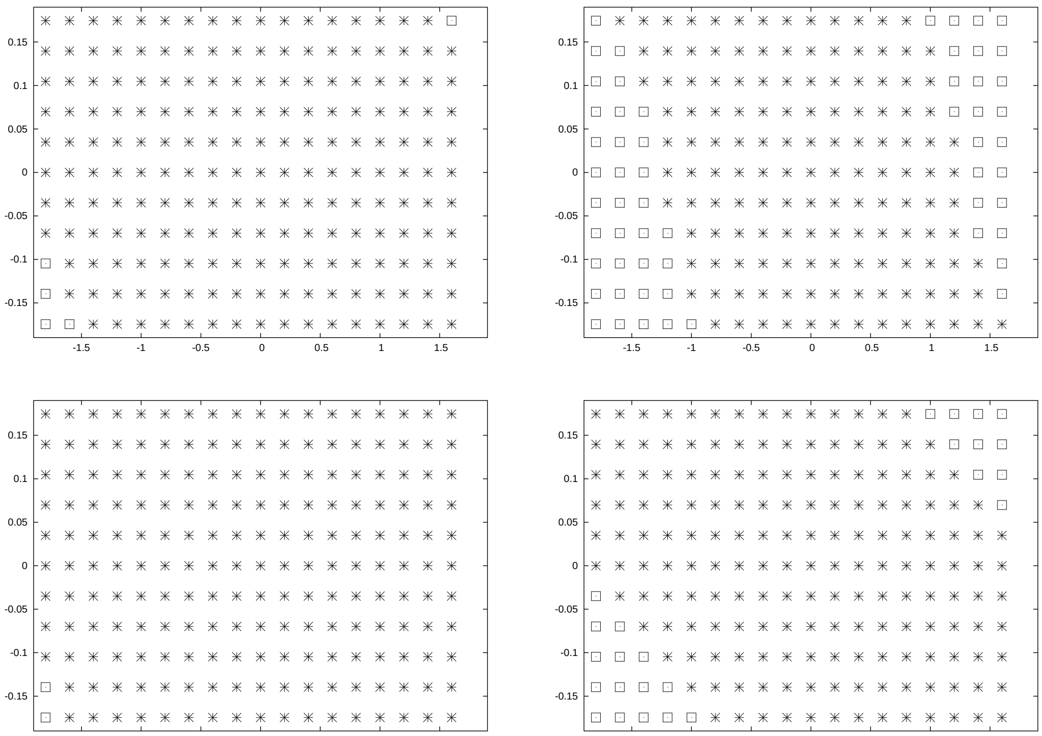


FIG. *7.23: Capacité de généralisation des meilleurs réseaux de la troisième série (réseau de gauche sur la figure 7.22). Les points indiquent la réussite (étoile) ou l'échec (carré) du contrôle pendant une expérience de deux minutes, la position initiale étant donnée par les coordonnées du point (x en abscisse en mètres et θ en ordonnée en radians). Le contrôleur a été testé pour des positions initiales variant de -1,8 à +1,8 par incrément de 0,2 pour x et de -0,17 radians à +0,17 radians par incréments de 0,034 radians. Les taux de généralisation sont de 97,5%, 71%, 99% et 87%.*

à évolution, ou avec modules P, I et D de structure fixe, ont donné des résultats similaires. De même, les expériences avec modules P, I et D dont certains poids sont imposés ont donné des résultats semblables aux expériences avec modules D uniquement. Les expériences avec modules D dont les poids ont été préparés ont nettement donné les meilleurs résultats. Ces trois catégories d'expériences définissent en quelque sorte des classes d'équivalence du point de vue de l'évolution. En effet, du point de vue de l'évolution, parvenir à générer une solution à partir de modules soumis à évolution est un problème similaire à celui de générer une solution en partant de modules P, I et D. Dans un cas, l'évolution doit trouver la bonne structure pour un module. Cette structure étant relativement simple, elle va apparaître assez régulièrement. Dans le deuxième cas, l'évolution doit choisir la bonne structure parmi les trois choix qui lui sont proposés. Compte tenu des résultats que nous avons observé, ces deux cas de figures semblent similaires. Lorsque les poids des modules I et D sont imposés, l'espace de recherche se réduit, de même que lorsqu'il n'y a que le module D dans le réservoir. Enfin, les expériences avec module D dont les poids sont imposés sont celles qui sont, par définition, le plus proches de la solution optimale. L'évolution a peu de chemin à parcourir : assemblage des modules (sachant que l'assemblage conduisant à la solution optimale est le plus simple compte tenu de la structure du module) et choix des valeurs absolues des poids.

Donner des modules est une autre façon de concevoir l'incrémentalité. Cela permet à l'évolution de se construire ou de réutiliser des éléments fonctionnels utiles pour résoudre un problème. Ainsi, plutôt que de décomposer le problème en tâches plus simples et de résoudre le problème étape par étape en superposant ou en combinant les solutions trouvées aux différentes étapes (comme l'ont fait, par exemple, Kodjabachian et Meyer (1998)), nous pouvons donner à l'évolution des unités computationnelles plus puissantes qu'un simple neurone. En lui permettant de les assembler et de les paramétrer convenablement pour le problème à résoudre, nous pouvons lui éviter de repartir de zéro à chaque nouvelle expérience.

Le problème du pendule inversé est toutefois un problème faisant intervenir un faible nombre d'entrées et de sorties. Nous allons à présent étudier les capacités de notre approche évolutionniste sur des problèmes plus complexes, faisant intervenir plus de degrés de liberté.

Chapitre 8

Contrôle d'un dirigeable lenticulaire : le projet ALPhA

Le projet ALPhA[44] s'inscrit dans des études à long terme dont l'objectif est la construction d'un dirigeable de type lenticulaire de grande taille en vue de transporter du frêt ou des passagers. Les travaux présentés ici s'inscrivent dans une première étape du projet, celle de l'utilisation d'une maquette de 10 mètres de diamètre (figure 8.1) destinée à tester la dynamique de ce type d'engin, ainsi que ses possibilités de contrôle. L'intérêt d'une forme lenticulaire est qu'elle offre une résistance au vent bien moindre qu'une forme de cigare classique. En revanche, elle entraîne une dynamique plus complexe.

Le projet

Le projet ALPhA regroupe Pierre Balaskovic, consultant expert en dirigeables, la société AirStar de Grenoble, spécialisée dans les ballons éclairants, l'équipe infres de l'ENST (Télécom Paris) et l'AnimatLab du LIP6.

Pierre Balaskovic et la société AirStar ont pris en charge la réalisation de la maquette. L'équipe système de l'ENST s'occupe de la partie électronique et informatique embarquée.

Notre contribution à ce projet

Nous nous sommes intéressé à la partie logicielle du projet et, plus spécifiquement, au contrôle de la plateforme. Dans un premier temps, nous avons développé le logiciel de mixage permettant au pilote d'agir directement sur les commandes pour les séances de test préliminaires. Dans un deuxième temps, nous avons développé la simulation du dirigeable qui a servi, pendant la phase de conception de la maquette, à étudier différentes configurations des empennages ainsi que l'effet du déplacement des gaz à l'intérieur de l'enveloppe. À l'issue de cette phase, il a été retenu d'utiliser des parois permettant de séparer l'enveloppe en quatre parties de façon à limiter les déplacements gazeux qui avaient tendance à amplifier les oscillations de l'appareil. La position

[44] Aéronef Léger pour la Photographie Aérienne

FIG. *8.1: Maquette du dirigeable ALPhA*

de l'empennage arrière, ainsi que sa taille, ont également été influencées par les résultats de la simulation.

Outre ces contributions, relativement annexes par rapport aux objectifs de notre travail de recherche, notre rôle principal était d'étudier la faisabilité d'un système d'aide au pilotage, puis d'un système de pilotage automatique de la plateforme au moyen de réseaux neuronaux.

Le premier objectif est de maintenir le dirigeable horizontal à une altitude déterminée par le pilote. Ce système doit alléger la tâche du pilote en prenant en charge la partie la plus critique de l'opération : le maintien d'assiette. En effet, comme nous le verrons plus tard, la forme lenticulaire dote le dirigeable d'une portance importante dès que l'engin n'est pas horizontal[45], ce qui rend les manœuvres délicates - spécialement lorsque l'on se trouve près du sol.

Le second objectif est d'effectuer un contrôle complet du dirigeable : maintien de l'assiette, de l'altitude et de la position. La position est déterminée à partir d'une caméra et d'un système de traitement d'images qui renvoie les coordonnées relatives d'un point de repère au sol, point au-dessus duquel le dirigeable doit se maintenir.

Description du dirigeable

Le dirigeable est de type lenticulaire et dispose d'un diamètre de dix mètres. Sept moteurs et trois gouvernes assurent sa stabilisation et sa propulsion.

[45]En fait dès que son axe longitudinal n'est pas aligné avec celui du vent.

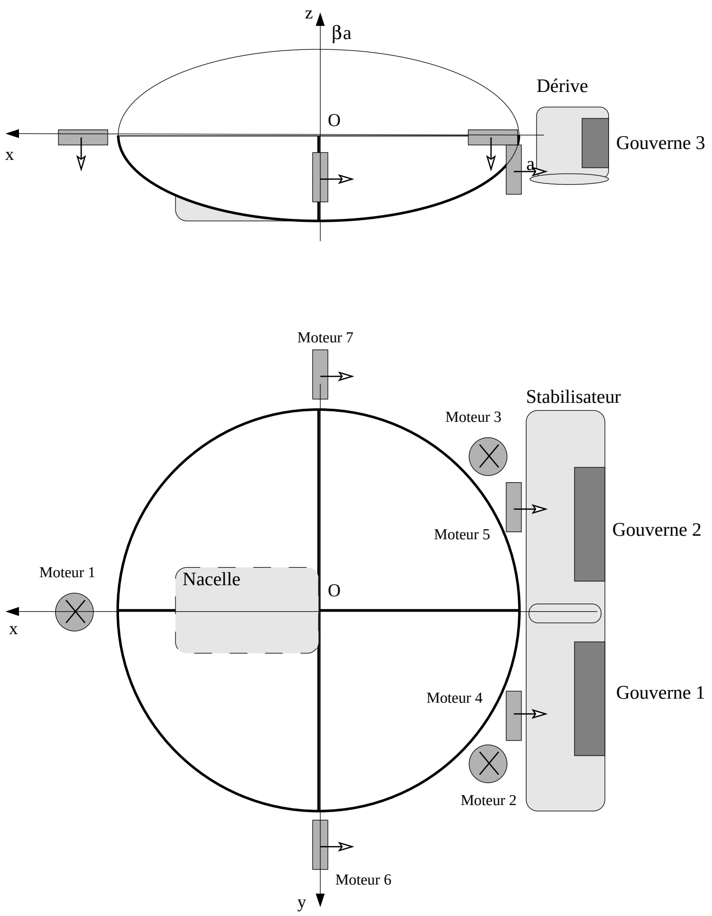


Fig. *8.2: Caractéristiques du dirigeable objet de l'étude*

Trois de ces moteurs, notés 1 à 3 sur la figure 8.2, servent uniquement au maintien de l'assiette et de l'altitude, leur axe étant vertical.

Les moteurs 4 et 5 servent uniquement à la propulsion et les moteurs 6 et 7 servent à la propulsion et au maintien du cap lorsqu'ils sont utilisés en différentiel.

Le dirigeable dispose également de deux gouvernes horizontales qui, utilisées simultanément, donnent un couple de tangage[46] et, en différentiel, un couple de roulis[47]. La gouverne verticale permet de donner un couple de lacet[48]. Nous ne les utiliserons pas dans les travaux décrits ici.

La première étape de notre travail sur ce projet a consisté à mettre au point un simulateur (voir annexe A) destiné à évaluer les contrôleurs générés par évolution et à faciliter l'étude et la conception de la maquette en testant différentes configurations (avec empennage versus sans empennage, par exemple).

L'annexe B contient le fichier de configuration utilisé pour une expérience portant sur le contrôle du dirigeable. Elle contient tous les paramètres de l'appareil que nous avons utilisés par

[46]Le tangage est l'angle de rotation autour de l'axe des y.
[47]Le roulis est l'angle de rotation autour de l'axe des x.
[48]Le lacet est l'angle de rotation autour de l'axe des z.

la suite.

Problématiques

Le contrôle de ce dirigeable a permis d'aborder de nombreux problèmes de difficulté variée en fonction de la configuration et des objectifs retenus.

Le maintien simultané de l'assiette et de l'altitude avec les seuls moteurs pose le problème de l'utilisation d'un effecteur pour atteindre deux objectifs potentiellement concurrents : les moteurs verticaux peuvent être utilisés pour le maintien d'assiette, s'ils sont utilisés en différentiel, et pour le maintien d'altitude, s'ils sont utilisés en conjonction. Le contrôleur devra donc utiliser le couplage le plus efficace pour mener à bien ces deux tâches.

De plus, du fait de la forme de l'engin, il est également possible de prendre de l'altitude ou d'en perdre rapidement en donnant une assiette à l'appareil : il prend alors une portance qui le fait monter ou descendre très rapidement[49], un phénomène sur lequel nous reviendrons dans la suite de ce chapitre.

Enfin, si l'on souhaite réaliser un pilotage automatique complet - pour un problème de suivi d'objet par exemple - il faut ajouter le contrôle de position dans le plan horizontal, ce qui impose de contrôler cinq degrés de liberté[50] et d'exploiter efficacement les couplages et les redondances d'effecteurs.

8.1 Maintien d'assiette et d'altitude

Cette série d'expériences concerne le maintien simultané de l'assiette et de l'altitude du dirigeable malgré les conditions de vent changeantes. Cette tâche requiert l'utilisation de trois capteurs, fournissant des informations sur les positions relatives de l'appareil quant à ces trois degrés de liberté. Le roulis et le tangage sont fournis par un inclinomètre, tandis que l'erreur sur l'altitude, par rapport à une altitude désirée, est donnée par la différence entre une consigne-cible donnée par un opérateur au sol et la valeur retournée par un altimètre. Les effecteurs utilisés sont les moteurs à poussée verticale, c'est-à-dire les moteurs 1 à 3 sur la figure 8.2. La structure reliant ces capteurs à ces effecteurs sera un réseau de neurones dont la structure et les paramètres seront soumis à évolution selon les outils et concepts expliqués précédemment.

8.1.1 Évaluation

La fitness utilisée est similaire à celle que nous avons employée pour le pendule inversé. Elle prend en compte la somme des erreurs au carré sur les différents degrés de liberté à maintenir. Les paramètres considérés sont donc le roulis, le tangage et l'erreur par rapport à l'altitude désirée. Après discussion avec l'expert du projet, nous avons imposé une valeur limite de 0,35 radians (soit approximativement 20 degrés) au roulis et au tangage, valeur au delà de laquelle

[49]Ce qui a d'ailleurs provoqué quelques frayeurs pendant les premiers tests en vol manuel !

[50]Le roulis, le tangage, l'altitude ainsi que la position dans le plan horizontal.

nous considérions que le contrôle avait échoué. Ces limitations définissent le domaine de vol de l'appareil et donnent les valeurs maximales utilisées ensuite pour normaliser les erreurs sur ces différents degrés de liberté. En revanche, comme le dirigeable est en équilibre au sein de la masse d'air grâce à l'hélium, il n'est pas nécessaire de générer de portance pour contrer son poids. C'est pourquoi le domaine de vol ne contient pas de limitation sur la vitesse relative du dirigeable par rapport à la masse d'air.

De façon à accélérer l'évolution, l'évaluation d'un génome est arrêtée dès que la limite d'assiette est dépassée. Pour prendre en compte cet aspect, nous avons ajouté un terme prenant en compte la durée de l'évaluation. La fitness globale est alors la somme de deux termes :

$$f(g) = p(g) + fit_{dec}(g)$$

où g est un génotype et $p(g)$ est le pourcentage de temps pendant lequel le contrôleur défini par le génotype g a réussi à maintenir l'assiette (un temps maximal étant imposé). $fit_{dec}(g)$ évalue l'éloignement moyen du système par rapport à la valeur de consigne pour chacun des degrés de liberté. C'est la moyenne arithmétique des éloignements moyens $f_i(g)$ pour chacun des degrés de liberté i :

$$f_i(g) = \left(1 - \frac{\sum_t (d_i(g,t)^2)}{T}\right)$$

où $d_i(g)$ est l'erreur normée par sa valeur maximale sur un degré de liberté. Nous avons comparé l'influence du choix de la fonction utilisée pour agréger les critères à la section 8.3.

Dans cette série d'expériences, trois paramètres sont à maintenir : $d_1(g,t) = \frac{\theta_l(g,t)}{\theta_{l_{max}}}$, $d_2(g,t) = \frac{\phi_1(g,t)}{\phi_{1_{max}}}$ et $d_3(g,t) = \frac{z(g,t)}{z_m ax}$. $\theta_{l_{max}}$ et $\phi_{1_{max}}$ valent 0,35 rad et $z_m ax$ a été choisi arbitrairement à 100m. $T = \sum_t 1$ est la durée totale de l'évaluation, N est le nombre de degrés de liberté. Nous avons ajouté un critère supplémentaire : l'énergie consommée (énergie moyenne consommée par chaque moteur normée par l'énergie maximale qu'un moteur est susceptible de consommer), de façon à minimiser également la consommation énergétique de l'appareil.

Un contrôleur stable - autrement dit, un contrôleur qui permet au dirigeable de rester dans son domaine de vol pendant toute la durée de l'évaluation - aura alors une fitness strictement supérieure à 100 (qui correspond à un maintien dans le domaine de vol pendant 100% du temps), le maximum étant de 101 (100% du temps dans le domaine de vol, auquel s'ajoute une note de 1 correspondant à un maintien 'parfait' de chaque degré de liberté).

L'évaluation du comportement d'un contrôleur se fait sur 4 expériences distinctes réalisées dans des conditions différentes. Elles durent en tout 1000 secondes simulées, temps suffisamment long pour s'assurer de la stabilité du contrôle, et soumettent le dirigeable à des vents variables en intensité et en direction, ainsi qu'à des perturbations régulières d'assiette. L'altitude désirée change également régulièrement de façon à s'assurer de l'efficacité du contrôle de ce degré de liberté. Les conditions d'évaluations sont fixes et identiques pour tous les individus.

L'algorithme évolutionniste utilise une population de 100 individus tout en générant initialement 10.000 individus (dont seuls les 100 meilleurs seront retenus à l'issue de la première génération). Les calculs prennent plusieurs jours (de 3 à 10 en fonction de la puissance du processeur et de la charge du système).

8.1.2 Connaissances a priori et informations évoluées

Nous avons réalisé cinq types d'expériences exploitant ModNet avec des réservoirs de modules-modèles différents :

- expériences sans connaissance a priori : un seul module dont la structure peut être générée et modifiée par évolution
- expériences avec modules : trois modules de structure fixe : module P (proportionnel), module I (intégral) et module D (dérivée) en laissant le poids des connexions évoluer
- expériences mixtes : trois modules de structure fixe P, I et D ainsi qu'un module dont la structure est soumise à évolution
- expériences avec connaissance a priori : trois modules P, I et D dont certains poids de connexions sont imposés pour garantir que les modules remplissent la fonction pour laquelle ils ont été conçus (I pour intégrer le signal et D pour le dériver, figure 8.3)
- expériences témoins : un seul module simple : une connexion relie directement l'entrée du module à sa sortie (module P)

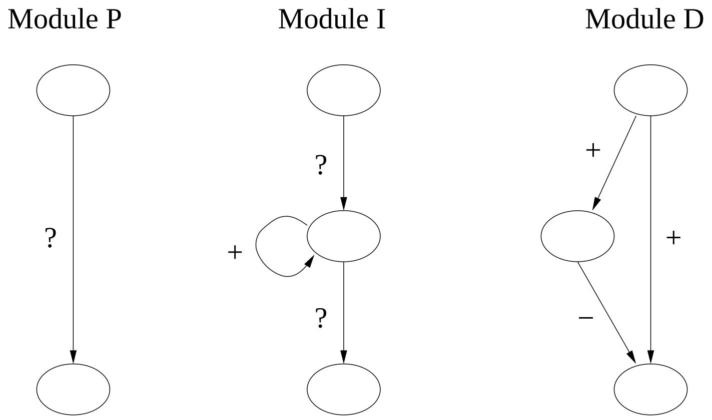


FIG. 8.3: *Modules P, I et D avec poids partiellement imposés. Le module P représente un bloc proportionnel, (il remplira cette fonction quelle que soit la valeur du poids). Le module I représente un bloc intégral. Pour ce faire, nous avons imposé à la connexion récurrente d'être positive. Le module D représente un bloc dérivé, ce qu'il ne peut être que si le signal passant par le neurone intermédiaire (signal du pas de temps précédent) est soustrait au signal passant en direct. Les " ?" indiquent des connexions dont le signe n'est pas imposé, les "+" un signe positif, les "-" un signe négatif. La valeur absolue des poids (et éventuellement le signe) sont soumis à évolution en utilisant un Algorithme Génétique.*

Ces différentes variantes ont pour but de tester les différentes conditions de fonctionnement et de vérifier si l'évolution est capable d'exploiter les connaissances mises à sa disposition, que ce soit sous forme de modules de structure ou, éventuellement, de paramètres prédéfinis.

Gén	Donnée	Témoin	Mod. evo.	P, I et D	mixte	P, I et D avec signe
20	Maximum	100,882	100,839	100,881	100,885	100,869
	Moyenne	100,826 (1)	100,803 (4)	100,804 (3)	98,267 (5)	100,818 (2)
	Minimum	100,789	100,767	100,747	70,14	100,777
100	Maximum	100,882	100,87	100,889	100,898	100,888
	Moyenne	100,838 (3)	100,823 (4)	100,845 (1)	98,755 (5)	100,844 (2)
	Minimum	100,789	100,784	100,793	75,79	100,794
500	Maximum	100,891	100,879	100,891	100,9	100,895
	Moyenne	100,854 (2)	100,846 (5)	100,853 (3)	100,851 (4)	100,865 (1)
	Minimum	100,807	100,791	100,8	100,792	100,829
1000	Maximum	100,893	100,883	100,891	100,9	100,906
	Moyenne	100,859 (4)	100,855 (5)	100,862 (2)	100,860 (3)	100,873 (1)
	Minimum	100,812	100,791	100,801	100,793	100,844

FIG. *8.4: Statistiques des expériences lancées sur le contrôle de roulis, tangage et altitude. 12 expériences ont été réalisées dans chaque conditions. La valeur entre parenthèses indique le rang de la série d'expérience pour une génération donnée. "Mod. evo." représente les expériences dans lesquelles les modules sont soumis à évolution.*

8.1.3 Résultats préliminaires

Statistiques

Nous avons réalisé 12 expériences différentes dans chacune des conditions décrites précédemment. Les fitness ont été relevées à la $20^{ème}$ génération, à la $100^{ème}$, la $500^{ème}$ ainsi que la $1000^{ème}$ génération pour les douze expériences réalisées dans chacune des conditions.

La figure 8.4 contient les statistiques de ces résultats obtenus sur les cinq types d'expériences décrites précédemment. Les expériences témoins se comportent, en moyenne, mieux que les expériences dans lesquelles la structure du seul module est évoluée. Les expériences "mixtes" avec des modules P, I et D ainsi qu'avec un module dont la structure peut évoluer, peinent dans les premières générations, mais rattrapent ensuite les expériences témoin. Les expériences avec modules P, I et D donnent des résultats similaires à ceux des expériences témoins. Seules les expériences avec modules P, I et D dont certains signes sont imposés montrent une différence notable par rapport aux expériences témoin[51]. L'évolution a donc utilisé ces modules pour générer des solutions plus efficaces. Par contre, l'ajout de modules moins contraints (P, I et D sans imposer de signes) ou de modules dont la structure est soumise à évolution n'a pas engendré d'amélioration notable sur cette expérience par rapport l'expérience témoin. Une cause possible serait le trop grand nombre de paramètre des modules I et D qui, associé à un vaste espace de recherche, ralentirait la convergence de l'évolution.

[51]Une différence d'un centième dans la fitness marque la différence entre un comportement oscillant à faible amplitude et une stabilisation complète.

Comportement observé

Les meilleurs individus des différentes expériences sont tous capables de maintenir l'altitude après 1000 générations. Le maintien d'assiette engendre des performances plus variables. Si le roulis et le tangage restent toujours à l'intérieur des limites fixées, la qualité du contrôle peut varier notablement d'une expérience à l'autre. Elle va du maintien avec oscillations entretenues, au maintien efficace avec des oscillations rapidement amorties. Cette disparité entre les expériences peut s'expliquer par le fait que 1000 générations ne sont probablement pas suffisantes pour permettre à toutes les expériences de converger[52].

Cependant, une caractéristique est présente dans tous les résultats observés : à certains instants, le tangage n'est pas maintenu autour d'une valeur nulle, mais à une valeur non nulle de signe variable d'une période à l'autre. Ce phénomène est un peu surprenant car la "consigne" est de toujours maintenir cette valeur autour de 0, consigne qui ne varie pas contrairement à la consigne d'altitude. Pourtant, les contrôleurs obtenus par évolution semblent la maintenir à une valeur non nulle précise dans certains contextes. En observant de plus près le comportement généré, nous avons observé que ces consignes "fantômes" apparaissaient lorsque l'appareil n'est pas à l'altitude désirée. De plus, le signe de ces consignes est lié au signe de l'erreur sur l'altitude. Lorsque le dirigeable doit monter, la nouvelle consigne d'incidence est négative et, dans le cas contraire, elle est positive. L'interprétation de ces résultats demande de revenir sur la dynamique du dirigeable et sur une de ses caractéristiques aérodynamiques.

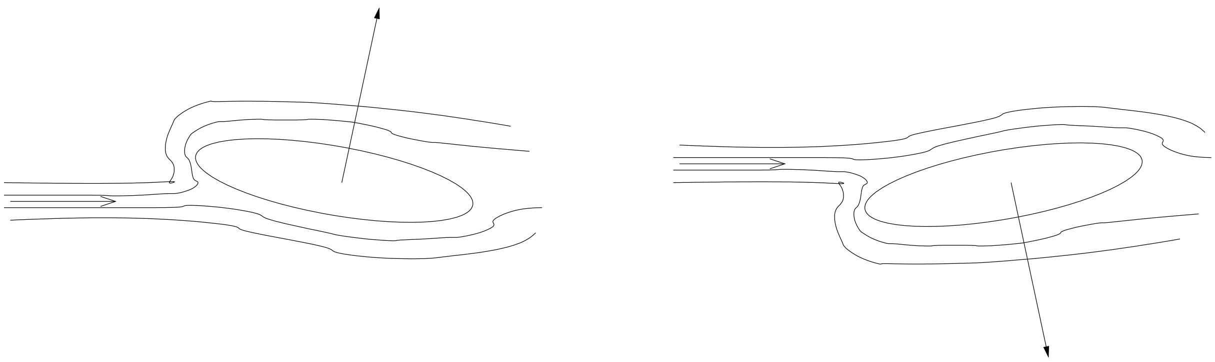


Fig. 8.5: *Phénomène de portance. L'air doit parcourir un chemin de longueur différente entre le dessus et le dessous de l'aile. Il va créer une surpression du côté où le chemin est le plus court et une dépression de l'autre côté. C'est ce qui crée la portance d'une aile d'avion. À gauche, l'aile est cabrée, le chemin est plus long pour passer au dessus de l'aile, la portance est donc dirigée vers le haut. À droite, c'est le contraire, la portance est donc dirigée vers le bas.*

Il apparaît en effet que le dirigeable utilisé dans ces expériences dispose, grâce à sa forme lenticulaire, d'une portance importante lorsque son incidence n'est pas nulle. L'enveloppe du dirigeable se comporte en fait comme une aile d'avion. Lorsque l'air n'a pas le même chemin à parcourir pour passer au-dessus ou en dessous de l'enveloppe, il se crée une dépression du côté

[52] Cette différence se joue sur la fitness au niveau des deuxième et troisième décimales.

où le chemin est le plus long et une surpression de l'autre, ce qui crée des forces perpendiculaires à l'enveloppe (figure 8.5). Lorsque le dirigeable fait face au vent et qu'il se cabre (autrement dit lorsque son assiette longitudinale est négative), il se crée une portance dirigée vers le haut : le dirigeable monte. Lorsque le dirigeable pique (autrement dit lorsque l'incidence est positive), la portance créée est dirigée vers le bas, le dirigeable descend. Dans cette série d'expériences, le vent est de direction variable, cependant la dérive arrière permet au dirigeable de se positionner passivement face au vent. Ainsi, quelle que soit la direction du vent, le dirigeable se retrouve systématiquement, après un temps très court, face au vent.

Ce phénomène de portance, créée par la forme de l'enveloppe lenticulaire, permet de faire varier l'altitude de façon beaucoup plus rapide qu'avec les moteurs uniquement. Pour le vérifier, nous avons réalisé un contrôleur n'exploitant pas cette caractéristique et se servant uniquement des moteurs verticaux pour maintenir à la fois l'assiette et l'altitude de l'appareil. Chaque degré de liberté est maintenu séparément et indépendamment des autres. L'assiette longitudinale est maintenue grâce à un PID commandant le moteur 1 et les moteurs 2 et 3 en différentiel[53]. L'assiette latérale est maintenue par un autre PID commandant les moteurs 2 et 3 en différentiel[54]. L'altitude enfin, est maintenue par un troisième PID dont les ordres sont envoyés de façon identique aux trois moteurs. Les ordres provenant des différents PID sont additionnés avant d'être envoyés aux moteurs. Les coefficients des différents PID ont été optimisés par une Stratégie Évolutionniste. L'assiette est parfaitement maintenue autour d'une valeur constamment nulle. L'altitude est également bien maintenue, mais à une vitesse plus de trois fois plus faible qu'avec les neurocontrôleurs obtenus par évolution ($3,2 m.s^{-1}$ avec les neurocontrôleurs contre $0,96 m.s^{-1}$ pour le PID), voir figure 8.6. Cette différence est dûe à la faible puissance des moteurs par rapport à la trainée crée par la taille importante de l'enveloppe. En exploitant les caractéristiques du dirigeable en interaction avec son environnement, l'évolution a ainsi pu obtenir un maintien d'altitude beaucoup plus efficace. Pour cela, elle a exploité le couplage existant entre le maintien du tangage et celui de l'altitude (Doncieux et Meyer, 2003).

Analyse de réseaux générés par l'évolution

Analyse du réseau 1. Le réseau 1 a été généré dans une expérience témoin (avec uniquement des modules P dans le réservoir de modules modèles). Il contient 8 neurones cachés et une connexion récurrente allant du neurone 2 au neurone 3 (figure 8.7). L'erreur sur l'assiette longitudinale commande directement le moteur 1. L'erreur sur l'assiette latérale commande directement le moteur 3. Le contrôle d'altitude commande les trois moteurs, permettant ainsi de faire varier l'altitude même en absence de vent. L'influence du capteur d'incidence et du capteur d'altitude est précisément calibrée par l'évolution pour permettre l'exploitation de la portance de l'appareil (figure 8.8).

Lorsque l'altitude est correcte, le neurone 2 oscille entre $+1$ et -1 à une période fixe de

[53] lorsque le moteur 1 reçoit la consigne o, les moteurs 2 et 3 reçoivent la consigne $-\frac{o}{2}$

[54] le moteur 3 reçoit $-o$ lorsque le moteur 2 reçoit o

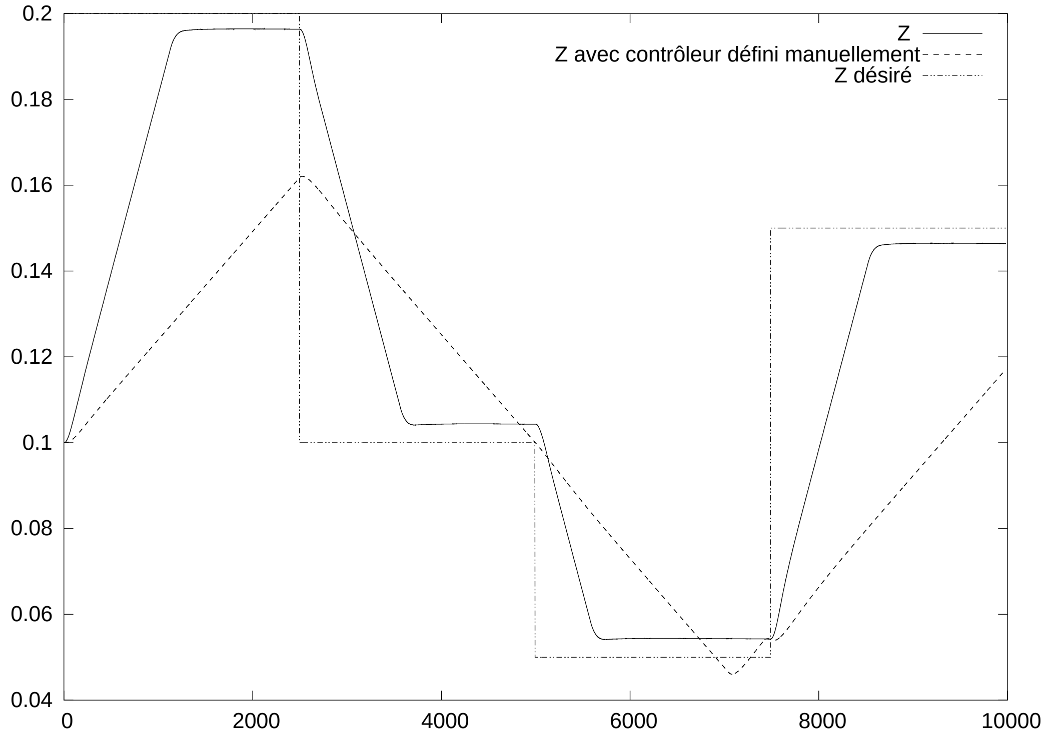


Fig. *8.6: Comparaison entre un neurocontrôleur entièrement conçu par évolution et un contrôleur défini manuellement. Le neurocontrôleur maintien l'altitude trois fois plus vite que le contrôleur défini manuellement. Un pas de temps (en abscisse) correspond à 25ms, l'altitude (en ordonnée) est en mètres.*

100ms. Cette oscillation, très précise, entraîne une oscillation du neurone 1, modulée par l'entrée de l'assiette longitudinale, ce qui permet de maintenir l'incidence à une valeur nulle de façon robuste (figure 8.9). Lorsque l'altitude est incorrecte, le neurone 2 sature et agit comme un biais sur le neurone 1, forçant ainsi le maintien de l'incidence autour d'une valeur non nulle. L'évolution a donc exploité les caractéristiques non-linéaires des neurones.

L'oscillation est exploitée pour le maintien de l'assiette latérale, qui est opéré par une succession périodique d'ordres de valeur maximale puis minimale (stratégie "bang-bang"). Étant donné que ces ordres semblaient constants au cours du temps, même lorsque nous ajoutions des perturbations sur l'assiette, nous avons supprimé le capteur i_1, pour vérifier s'il avait de l'influence sur le contrôle de ce paramètre. Il s'est avéré que, dans les conditions de test, cette suppression n'a absolument aucun effet sur le comportement du dirigeable (courbes rigoureusement identiques avec et sans le capteur).

Ce comportement n'est pas très surprenant, car nous avons pu observer en simulation que le roulis était auto-stabilisé par la forme du dirigeable et surtout la présence des empennages arrières. Cependant, afin de comprendre comment fonctionne cet asservissement, nous l'avons étudié plus en détail. La période des oscillations est de 100ms (oscillations imposées par le neurone 3, qui forme un oscillateur avec le neurone 2). Le signal envoyé sur les moteurs 2 et

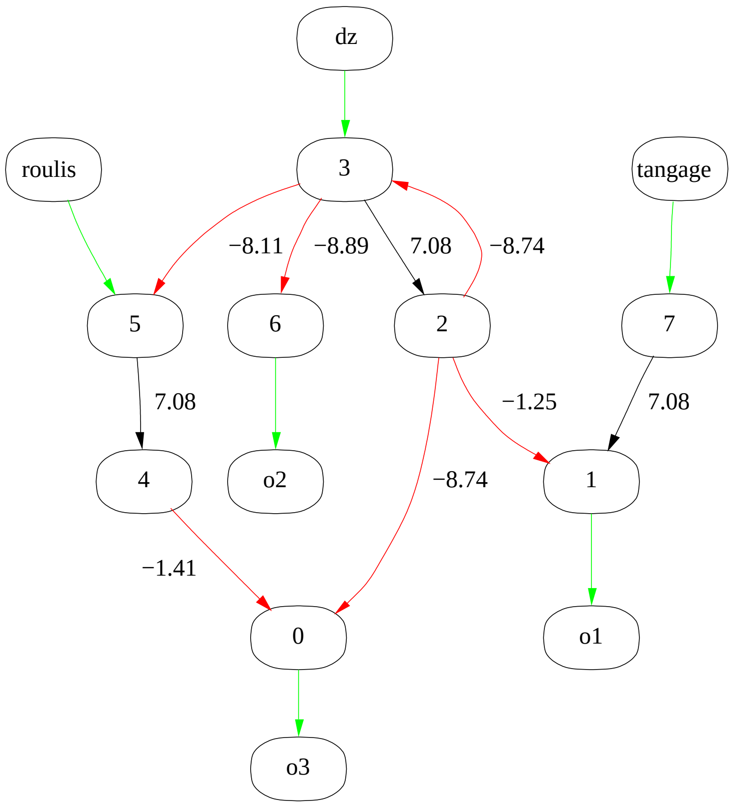


Fig. *8.7: Réseau 1. Meilleur réseau obtenu dans une expérience témoin (un seul module disponible : module contenant uniquement une connexion directe, module P). Les sorties o_1, o_2 et o_3 sont reliées aux moteurs 1, 2 et 3, qui sont les trois moteurs à poussée verticale (le moteur 1 étant le moteur avant). La fonction de transfert des neurones est la fonction tanh.*

3 est déphasé de $\frac{\pi}{2}$, autrement dit de 50ms. Dans un premier temps, nous avons pensé que ce déphasage était responsable du maintien de l'assiette latérale. C'est pourquoi, nous avons ajouté des neurones intermédiaires de façon à synchroniser les commandes envoyées sur les moteurs 2 et 3. Cette modification a diminué légèrement les performances du contrôleur, mais pas de façon significative (figure 8.10).

Le déphasage ne semblant pas être un paramètre prépondérant, nous avons étudié l'effet de variations dans la fréquence de l'oscillateur sur le contrôle du roulis et du tangage (figure 8.11). Le résultat de ces observations est que, avec le contrôle du tangage, le roulis est "passivement" maintenu : même en maintenant les moteurs 2 et 3 éteints, le roulis revient rapidement à la valeur nulle. Lorsque les moteurs 2 et 3 sont constamment éteints, le maintien du tangage est cependant moins performant. Le maintien du roulis se détériore progressivement en même temps que l'augmentation de la période des oscillations appliquées aux moteurs 2 et 3. Dans le même temps, le maintien du tangage, peu performant lorsque les moteurs 2 et 3 sont éteints, commence par s'améliorer avant de se dégrader à nouveau. La période de 8 pas de temps - période choisie

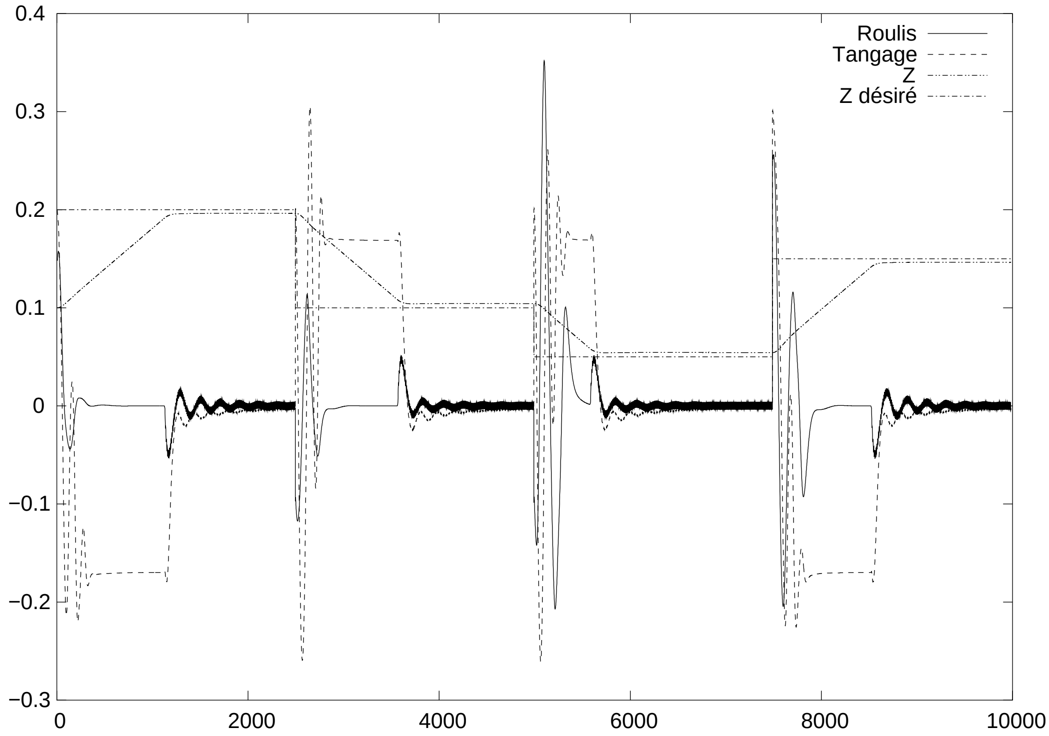


Fig. 8.8: *Comportement du dirigeable contrôlé par le réseau 1. Un pas de temps (en abscisse) correspond à 25 ms, les angles (en ordonnée) sont en radians et les altitudes (en ordonnée également) en mètres (elles sont divisées par 1000). Les conditions d'observation diffèrent des conditions d'évaluation appliquées pendant l'évolution afin de s'assurer de la robustesse du contrôle.*

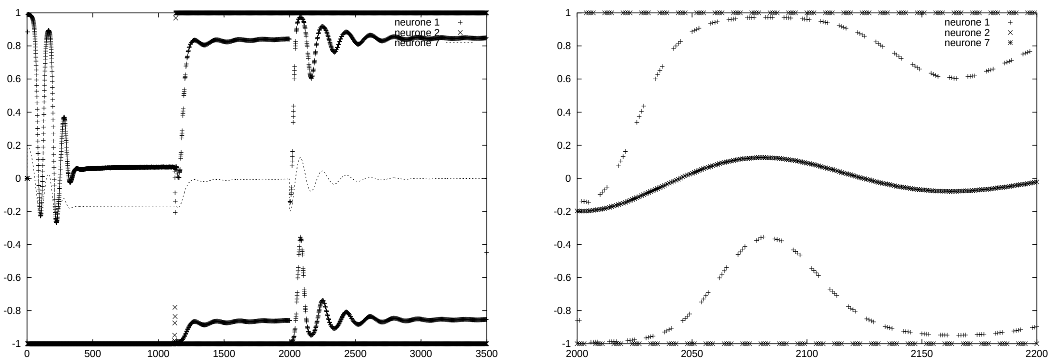


Fig. 8.9: *Évolution de la sortie des neurones 1, 2 et 7 au cours du temps au cours de l'évaluation représentée sur la figure 8.8. La figure de droite représente un zoom sur l'intervalle de temps compris entre les pas de temps 2000 et 2200. Le temps est en abscisse et l'amplitude de la sortie des neurones en ordonnée.*

par l'évolution - offre le meilleur compromis entre contrôle du roulis et contrôle du tangage.

Physiquement, les commandes de période trop faibles sont filtrées par l'inertie du dirigeable,

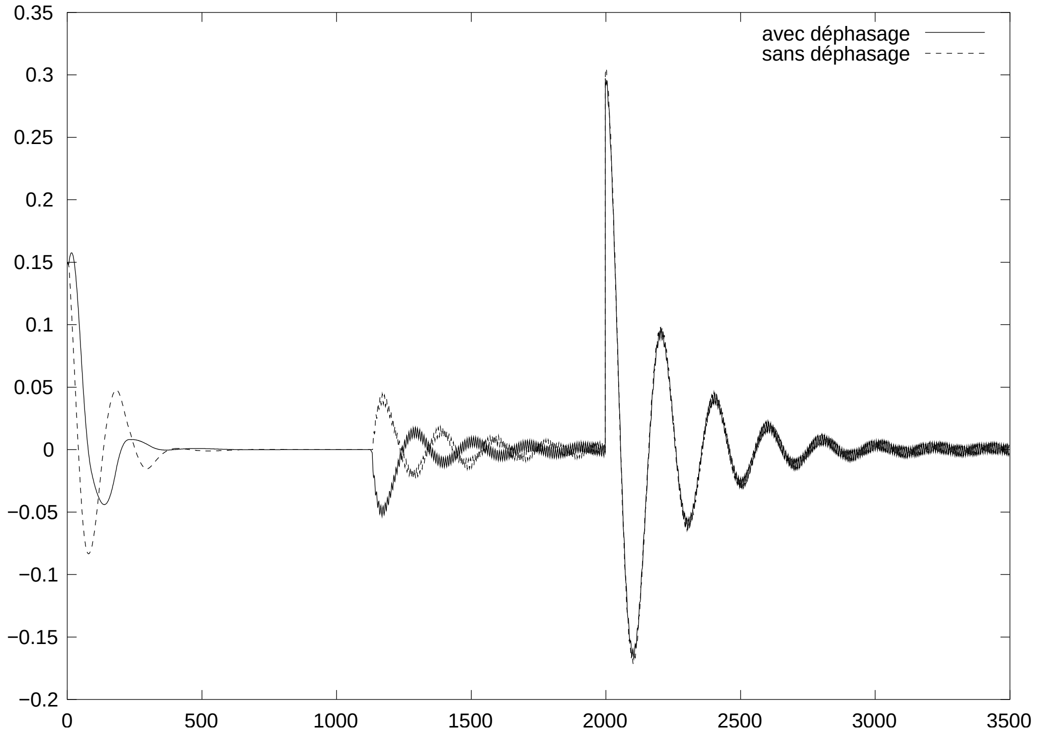


FIG. 8.10: *Valeur de l'assiette latérale avec et sans déphasage sur les moteur 2 et 3. Avec le déphasage, le contrôle est meilleur lors du maintien dans les premiers pas de temps, mais la différence est ensuite négligeable.*

ce qui explique la faible influence des commandes de période trop faible. Inversement, les commandes de période plus élevées ne sont pas du tout filtrées et ont une influence importante sur le comportement du dirigeable[55].

La présence d'une connexion récurrente dans un contrôleur a priori réactif nous a surpris et nous avons cherché à comprendre son utilité. Nous avons observé l'effet de sa suppression sur le maintien d'assiette (figure 8.12). Le maintien d'assiette dans les phases de montée ou de descente est inchangé mais, lorsque l'altitude est correcte, l'assiette montre des oscillations non amorties si l'on enlève la connexion allant de 2 vers 3. Les oscillations, auparavant de période fixe et ne contenant que des valeurs saturées, existent toujours lorsque la connexion est supprimée, mais à une période plus erratique et avec des valeurs intermédiaires non saturées. Cela démontre que la boucle sensori-motrice a naturellement tendance à osciller. Cependant, en imposant la période des oscillations à une valeur plus faible (0.1s), le réseau parvient à supprimer les oscillations d'assiettes.

[55] Le dirigeable en interaction avec son environnement se comporte comme un filtre passe-bas du fait de son inertie : les basses fréquences, autrement dit les périodes importantes, "passent" alors que les hautes fréquences sont coupées.

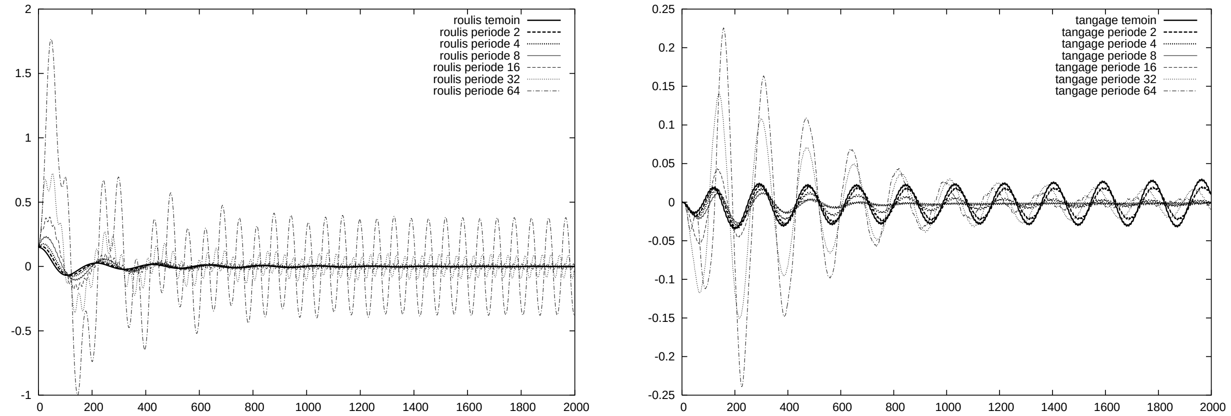


FIG. *8.11: Comportement du dirigeable lorsque l'on remplace les ordres allant des moteurs 2 et 3 par des ordres générés par un oscillateur défini manuellement. Le roulis figure sur les courbes de gauche, le tangage sur les courbes de droite. Nous avons fait varier la période de l'oscillateur (donnée en pas de temps) et nous avons observé le résultat sur le maintien du roulis et du tangage de l'aéronef. Les ordres envoyés au moteur 1 proviennent toujours du réseau de neurones de la figure 8.7. Dans l'expérience témoin, les moteurs 2 et 3 sont constamment éteints. Avec les ordres envoyés au moteur 1, le roulis est "passivement" maintenu, comme le montre l'expérience témoin. Cependant, dans ce cas, le tangage n'est plus aussi bien maintenu. Parmi toutes les combinaisons illustrées ici, l'évolution a choisi le meilleur compromis pour maintenir à la fois le roulis et le tangage aussi proches de 0 que possible.*

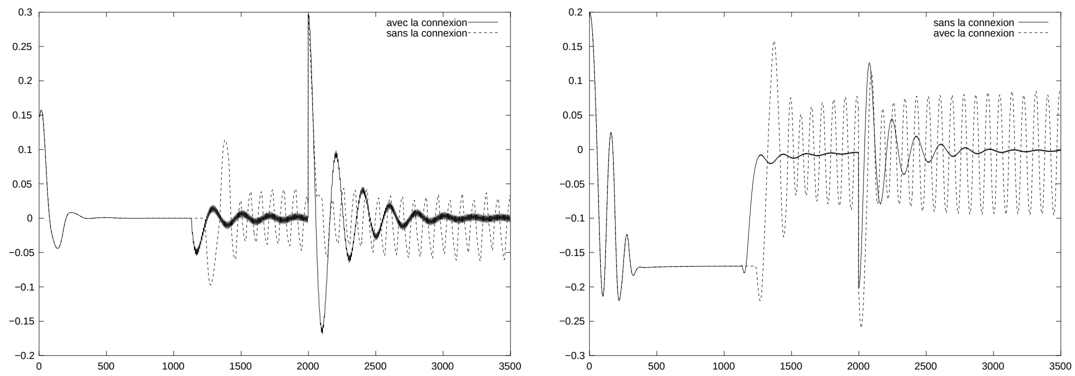


FIG. *8.12: Maintien de l'assiette du dirigeable avec et sans la connexion récurrente allant du neurone 2 au neurone 3. Le roulis figure sur les courbes de gauche et le tangage sur les courbes de droite. Pendant les 1000 premiers pas de temps, le dirigeable monte pour atteindre l'altitude désirée. Au 2000ème pas de temps, on modifie extérieurement l'assiette.*

Analyse du réseau 2. Le réseau 2 (figure 8.14) est le meilleur réseau obtenu dans des expériences avec modules P, I et D dont certains signes de connexions sont imposés. Avec 16 neurones cachés, il est beaucoup plus complexe que le réseau 1. Bien que des modules proportionnels

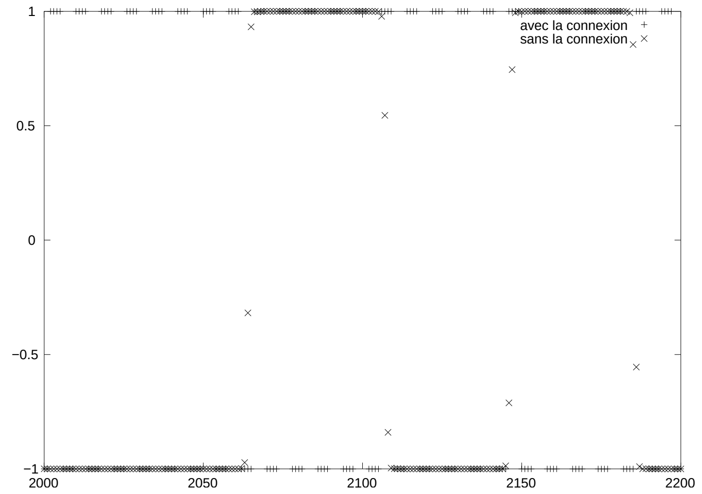


FIG. 8.13: *Sortie du neurone 0 avec et sans la connexion récurrente (la valeur de sortie des autres neurones oscillants est similaire). Avec la connexion, la période est fixe ; sans, elle est plus erratique et certains ordres intermédiaires sont non saturés.*

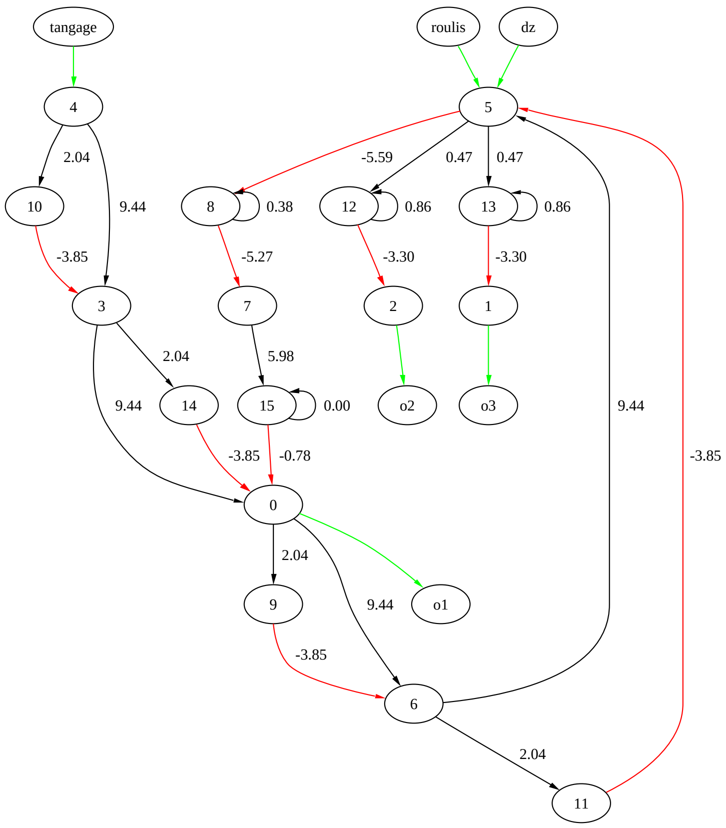


FIG. 8.14: *Réseau 2. Meilleur réseau obtenu dans une expérience avec modules P, I et D dans le réservoir (avec certains signes de connexions imposés).*

simples soient disponibles dans le réservoir, ce module ne contient que des modules I ou D (figure 8.15). Le module 4 contient cependant une connexion récurrente nulle, il est donc équivalent à deux modules proportionnels en série.

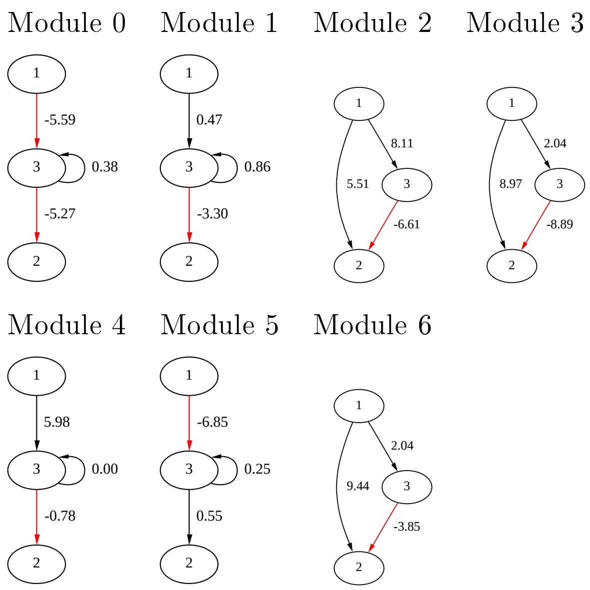


FIG. *8.15: Modules modèles utilisés dans le réseau 2.*

Le comportement généré par le réseau 2 présente deux différences par rapport au comportement généré par le réseau 1.

La première différence intervient dans la transition entre montée ou descente du dirigeable et maintien à altitude stable. Le dirigeable contrôlé par le réseau 1 montrait des oscillations de roulis parasites pendant cette période[56] alors qu'aucune perturbation externe n'avait été appliquée sur ce degré de liberté (figure 8.8). Le dirigeable maintenu par le réseau 2 voit son roulis maintenu rigoureusement à 0, même pendant cette étape de transition (figure 8.16). Le passage d'un régime de fonctionnement à un autre n'influe plus sur la qualité du contrôle de l'assiette latérale. Aucune perturbation n'apparaît pendant la transition.

La deuxième différence concerne l'erreur statique dans le contrôle d'altitude. L'erreur statique est l'erreur entre la valeur désirée et la valeur réelle en régime permanent[57]. Les deux réseaux sont capables de maintenir l'altitude du dirigeable sans oscillations. En revanche, tous les deux génèrent une erreur statique, mais cette erreur est 4 fois plus faible avec le réseau 2 (figure 8.17).

[56]Cette période de transition correspond au passage d'un mode de fonctionnement saturé constant à un mode de fonctionnement cyclique, dans lequel les moteurs sont commandés avec un signal en créneau, valeur maximale - valeur minimale à une période fixe.

[57]Le régime permanent intervient après le régime transitoire dans lequel la valeur réelle varie. Dans le régime permanent, la valeur réelle soit est constante soit oscille avec une amplitude constante.

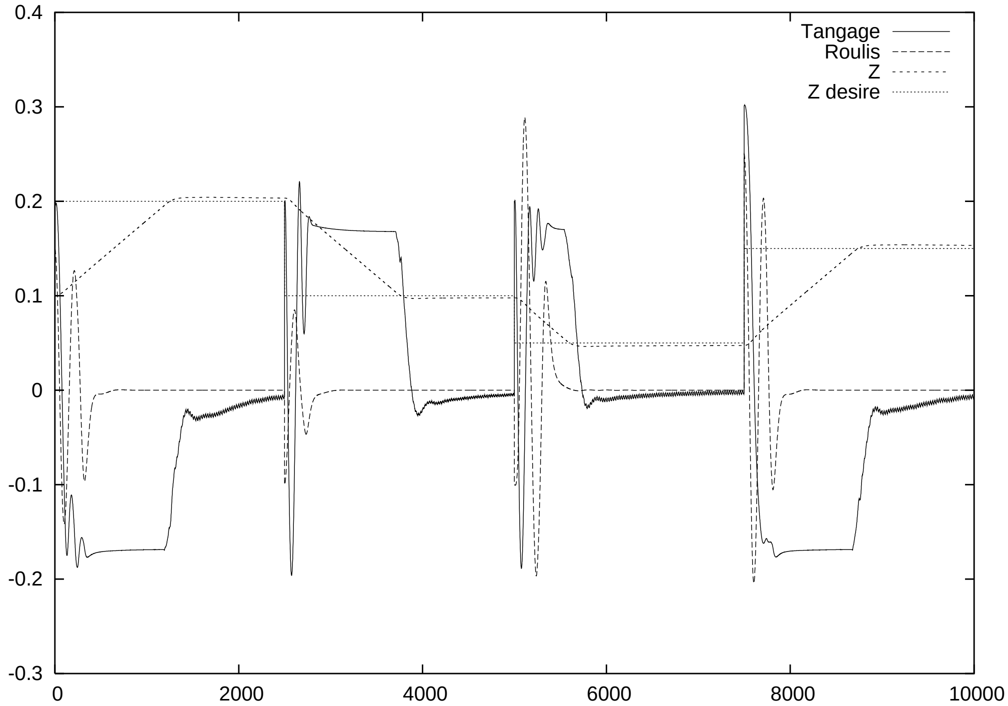


FIG. *8.16: Comportement du dirigeable contrôlé par le réseau 2. Un pas de temps correspond à 25 ms, les angles sont en radians et les altitudes en mètres (elles sont divisées par 1000).*

8.1.4 Résultats avec un contrôle continu

Les résultats des expériences citées précédemment ne sont pas applicables directement sur le dirigeable réel à cause de la stratégie de contrôle utilisée : l'alternance rapide de commandes maximales, dans un sens puis dans l'autre, serait très rapidement dommageable pour les moteurs et ce type de commande "bang-bang" serait, de toute façon, filtré par les variateurs des moteurs dont l'électronique veille à préserver l'intégrité en bloquant toute commande dangereuse.

L'utilisation de ce mode de fonctionnement s'est révélé fortement lié au fait que les capteurs n'étaient pas normalisés. En effet, le capteur de différence d'altitude prenait alors régulièrement des valeurs de sorties très élevées qui avaient tendance à saturer rapidement les neurones. L'évolution a exploité cette caractéristique pour générer des contrôleurs dont le fonctionnement était basé sur ces saturations.

Afin de favoriser l'émergence de contrôleurs plus continus, nous avons donc normalisé les différents capteurs. Les inclinomètres ont été normalisés à partir de la valeur maximale autorisée pour le roulis et le tangage, c'est-à-dire $0,35rad$, et le capteur d'erreur d'altitude a été normalisé arbitrairement pour atteindre la valeur maximale de 1 lorsque la différence d'altitude atteint $10m$ (ce qui correspond à un diamètre de l'appareil). Les autres paramètres sont identiques à ceux de la série précédente.

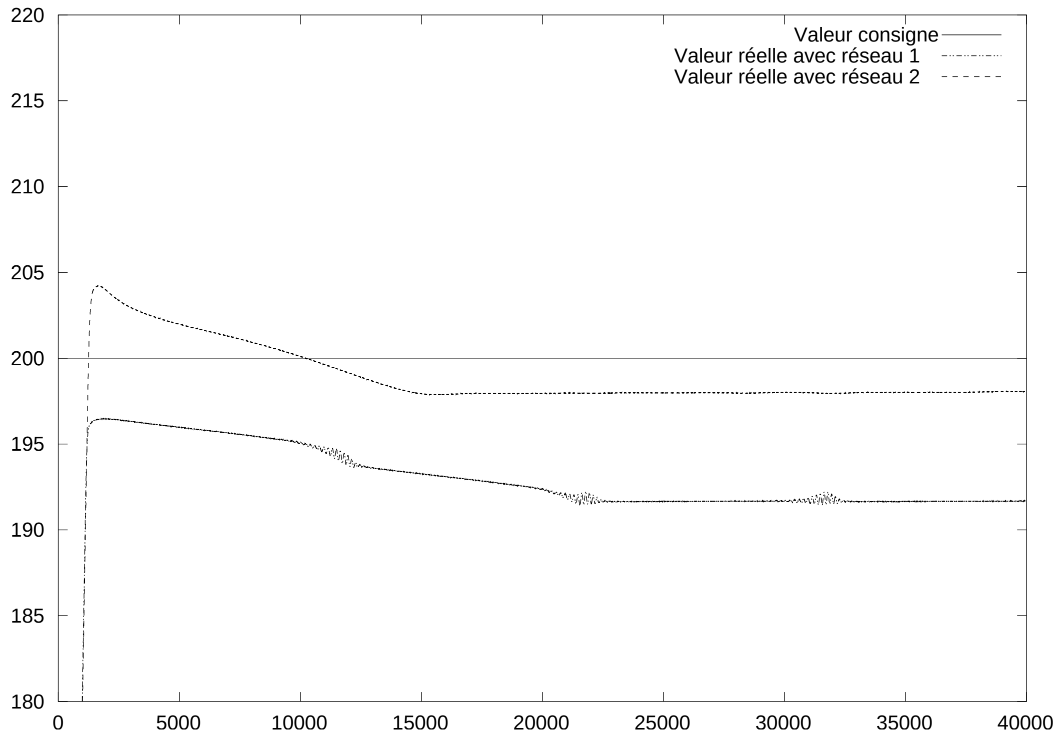


FIG. 8.17: *Comparaison entre le maintien d'altitude avec le réseau 1 et avec le réseau 2. Les deux réseaux permettent d'augmenter l'altitude à la même vitesse (en exploitant la portance du dirigeable), mais ils génèrent tous les deux une erreur statique. Cependant, cette erreur est environ quatre fois plus faible avec le réseau 2 (le dirigeable est maintenu à 198 mètres pour une valeur désirée de 200 mètres, soit une erreur de 1% avec le réseau 2 contre 191,7 mètres, soit une erreur de 4,15% pour le réseau 1).*

Statistiques

Nous avons lancé quatre expériences avec module P seulement et quatre avec modules P, I et D. Les fitness finales après 2000 générations sont stockées dans le tableau suivant :

module P	modules P, I et D
100,841	100,858
100,882	100,869
100,912	100,886
100,916	100,918

De même que précédemment, l'ajout des modules I et D ne semble pas particulièrement faciliter l'évolution. La courbe d'évolution de la fitness ne favorise aucun des deux jeux d'expériences non plus.

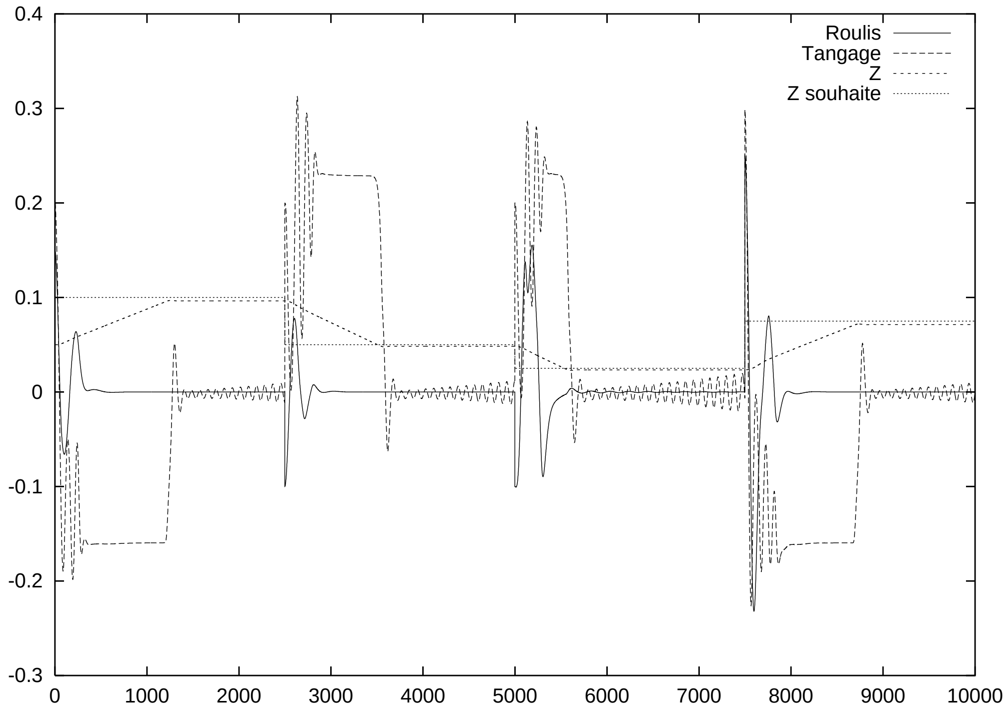


Fig. *8.18: Comportement du dirigeable contrôlé par le réseau 3 (deuxième série d'expériences). Un pas de temps correspond à 25 ms, les angles sont en radians et les altitudes en mètres (elles sont divisées par 1000). Des perturbations d'assiette sont imposées tous les 2500 pas de temps. Le vent change à plusieurs reprises d'orientation et d'amplitude.*

Comportement observé

Le comportement généré par les contrôleurs obtenus est similaire à celui obtenu dans les expériences précédentes : la portance de la forme lenticulaire est exploitée pour prendre ou perdre plus rapidement de l'altitude (figure 8.18). Si le contrôle de tangage génère des oscillations d'amplitude croissante, on peut constater sur la figure 8.19 que l'amplitude atteint une valeur maximale de $0,034rad$, soit moins de $2deg$, ce qui reste très faible[58]. La période des oscillations est de $2,25s$, la vitesse angulaire est alors de $3,5deg.s^{-1}$.

Hormis ces oscillations, une autre différence peut être remarquée : l'angle d'attaque utilisé pour prendre de l'altitude ou en perdre n'est pas le même en valeur absolue. Il est de $-0.16rad$ pour la montée[59] et de $0.23rad$ pour la descente. Cela implique une différence importante entre la vitesse de montée et la vitesse de descente : le dirigeable parvient à monter de $100m$ en $30s$ et il descend de $100m$ en $25s$ seulement. Nous reviendrons sur cette différence lors de l'étude du réseau, dans le paragraphe suivant.

Une dernière différence peut être remarquée par rapport aux expériences précédentes. Les commandes envoyées aux moteurs sont ici compatibles avec les caractéristiques des moteurs réels

[58]La limite maximale autorisée étant de $20deg$.

[59]Par convention, l'angle est positif lorsque le dirigeable est en piqué, autrement dit le nez vers le bas.

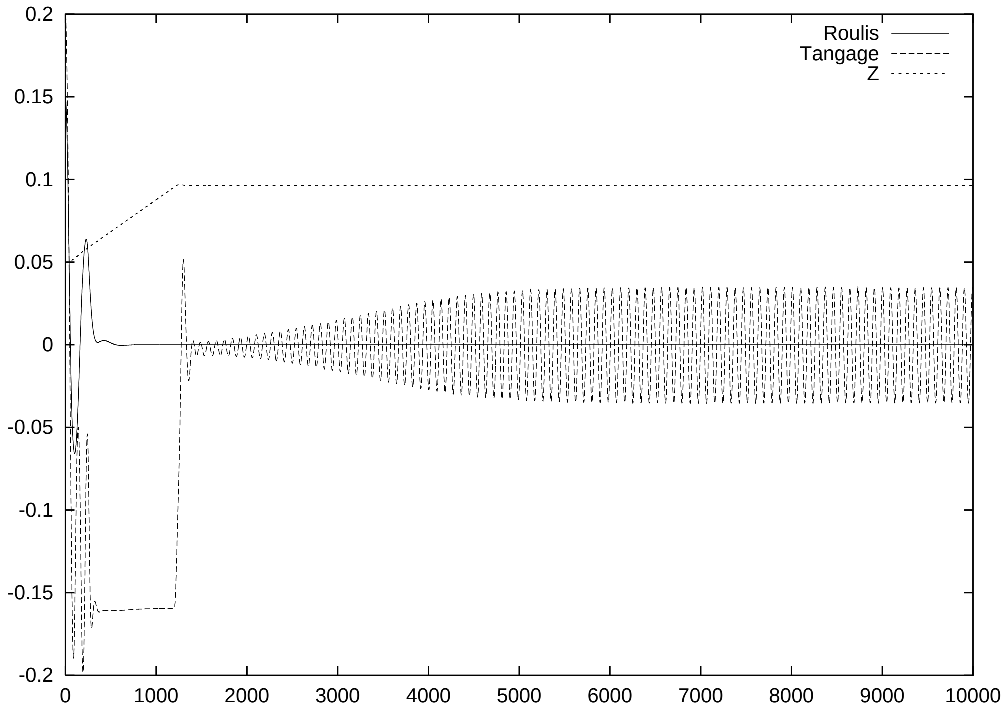


FIG. *8.19: Comportement du dirigeable contrôlé par le réseau 3 (deuxième série d'expériences). Un pas de temps correspond à 25 ms, les angles sont en radians et les altitudes en mètres (elles sont divisées par 1000). Évaluation réalisée pour tester la stabilité du contrôle en tangage. La consigne d'altitude est constante et aucune perturbation d'assiette n'est appliquée excepté au début de l'évaluation. Le tangage suit une oscillation d'amplitude* $0,034 rad$ *et de période* 90 *pas de temps, soit* $2,25s$.

(figure 8.20 et 8.21). En effet, ce réseau de neurones n'utilise pas de stratégie de contrôle de type “bang-bang” : les commandes moteurs varient de manière continue et généralement lente. Les oscillations entretenues les plus importantes se produisent pendant la phase “stable” de contrôle de tangage, l'amplitude est de 100W (soit moins de la moitié de la puissance disponible) et la période des oscillations est de $2,25s$.

Analyse de réseaux générés par évolution

Le réseau utilisé pour générer le comportement de la figure 8.19 exploite des modules 'D' ainsi que des modules 'I' (figure 8.22). Le neurone intermédiaire de ces modules 'I' sature très rapidement du fait de la connexion récurrente qui a un poids élevé de 9,21. Ces neurones agissent ensuite comme des biais sur les neurones auxquels ils sont reliés (neurone 2). Cette observation permet de simplifier le réseau en remplaçant les boucles récurrentes arrivant sur le neurone 2 par un seul neurone “biais” (figure 8.23) dont la sortie est constante au cours d'une évaluation : soit à $+1$, soit à -1, en fonction des conditions initiales. Lorsque le neurone “B” reste bloqué à -1, le dirigeable est plus lent à monter qu'à descendre. Il modifie le point de fonctionnement

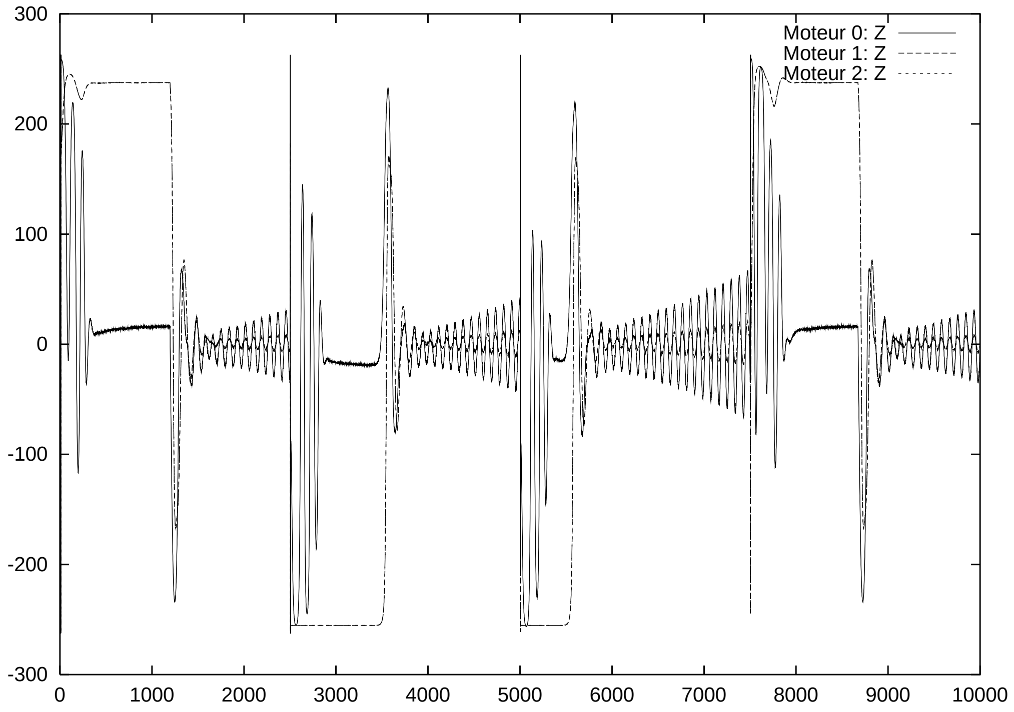


FIG. 8.20: *Commandes envoyées aux moteurs pour générer le comportement représenté sur la figure 8.18. Un pas de temps correspond à 25 ms. L'ordonnée représente la puissance du moteur (en Watt), la puissance maximale de tous les moteurs étant de 260W (en prenant en compte le rendement mécanique et celui de l'hélice).*

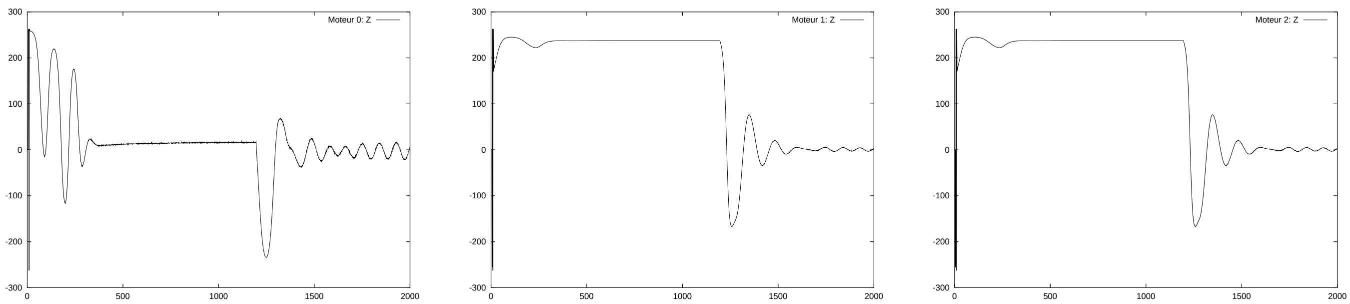


FIG. 8.21: *Commande envoyées aux moteurs pour générer le comportement représenté sur la figure 8.18. Zoom sur les 2000 premiers pas de temps. Un pas de temps correspond à 25 ms. L'ordonnée représente la puissance du moteur (en Watt).*

du neurone 2. Ce comportement asymétrique donne de bonnes performances en moyenne sur les quatre expériences réalisées pour évaluer un individu. Il correspond donc à une spécialisation du contrôleur pour les conditions particulières dans lesquelles il était évalué.

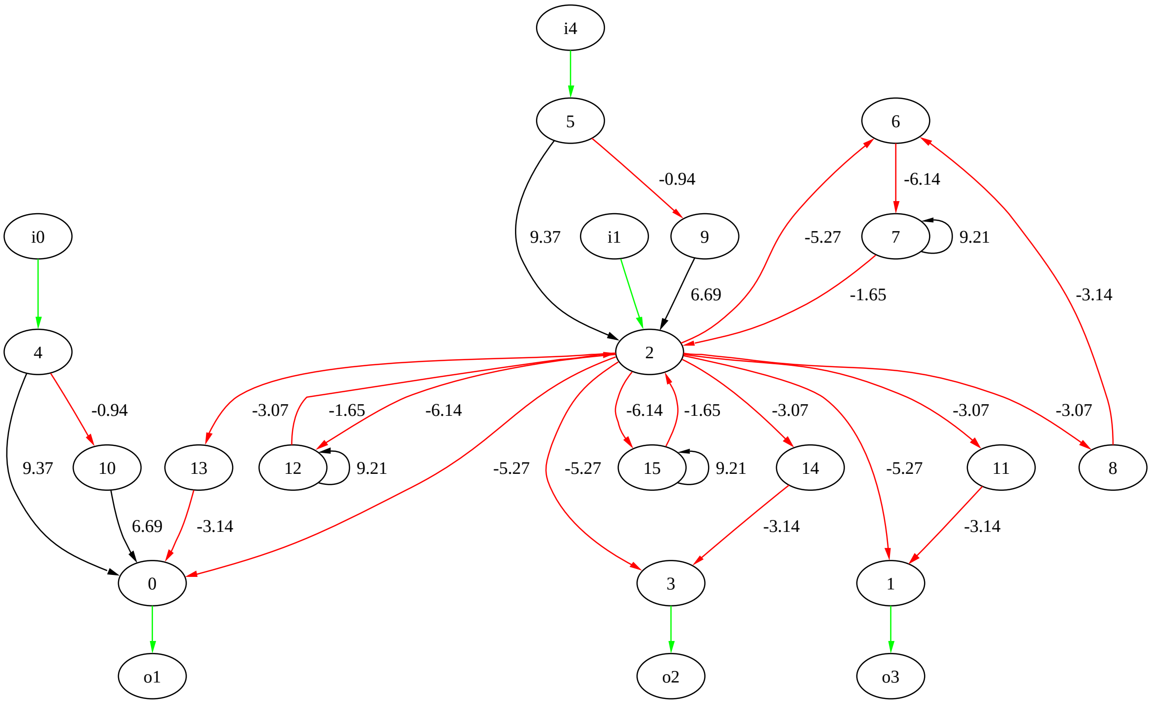


FIG. *8.22: Réseau 3. Meilleur réseau obtenu dans une expérience avec modules P, I et D (deuxième série d'expériences).*

Utilisation des connaissances a priori

Le réseau 3 est principalement constitué de modules 'D', dont la structure était imposée, mais pas les poids (figure 8.23). Dans les deux modules utilisés, le jeu de poids découvert par évolution réalise bien en sortie du module la différence entre un pas de temps et le pas de temps précédent à une constante multiplicative près, si l'on suppose que l'on reste dans la zone de fonctionnement linéaire des neurones.

Le module a cependant un deuxième mode de fonctionnement. Lorsque l'altitude est incorrecte, le neurone 2 sature à $+1$ ou -1. La sortie du module devrait logiquement être nulle (la différence entre deux pas de temps étant nulle), mais ce n'est pas le cas : elle sature également. En effet, le neurone intermédiaire (le neurone 14 sur la figure 8.23) sature lui aussi et l'entrée du neurone de sortie du module (le neurone 3) vaut alors

$$-5.27 * x - 3.14 * (-x) = -2.13 * x$$

où x est la sortie du neurone 2. La sortie du neurone est donc de signe opposé à la sortie du neurone 2 et proche de la saturation puisque $x = \pm 1$.

Le module dispose donc d'un point de fonctionnement saturé lorsque l'altitude est incorrecte et d'un point de fonctionnement non saturé, proche de 0, lorsque l'altitude est correcte (figure 8.24). L'évolution a donc exploité à la fois les capacités de calcul de dérivée de ce type de structure

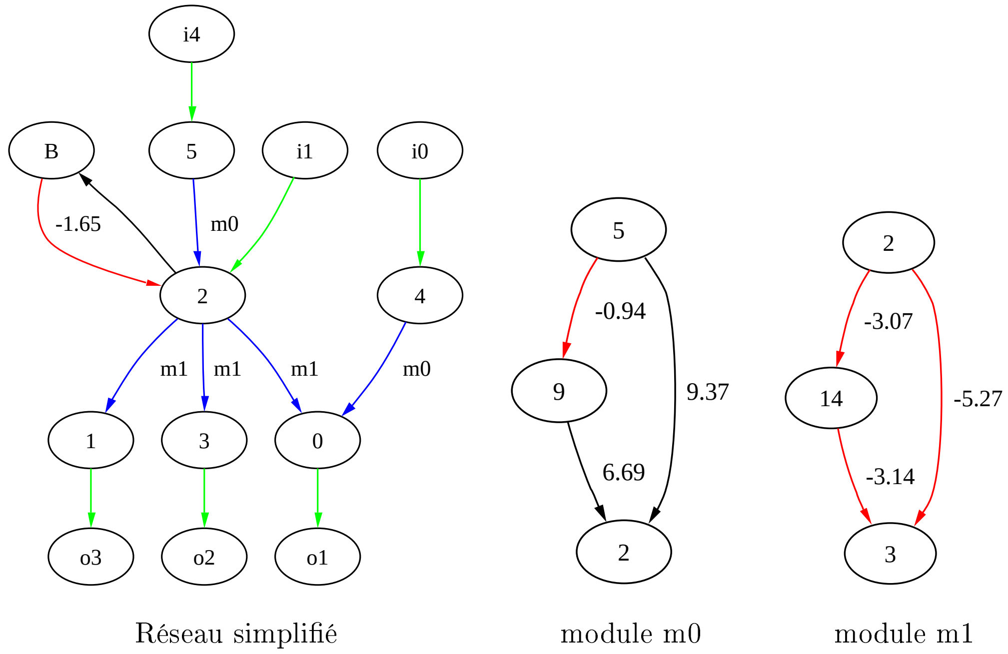


Fig. 8.23: *Version simplifiée (c'est-à-dire expurgée des neurones ou des connexions qui ne servent à rien) du réseau 3 (à gauche). m0 (au milieu) et m1 (à droite) sont des modules "D", "B" est un neurone de sortie constante au cours d'une évaluation (neurone biais). La sortie de ce neurone vaut +1 ou -1 en fonction des conditions initiales (dépendance représentée par la connexion allant du neurone 2 à B).*

ainsi que les effets de bord dûs aux non linéarités des fonctions de transfert des neurones.

Dans ces expériences, l'évolution a donc su exploiter des couplages non-linéaires entre les différents degrés de liberté pour générer un contrôle optimal.

Nous allons à présent nous intéresser au problème complet du contrôle du dirigeable.

8.2 Maintien complet : assiette, altitude et position

Dans cette série d'expériences, le dirigeable doit toujours se maintenir à l'horizontale à une altitude donnée. Il doit également se maintenir à la verticale d'un point particulier. En simulation, on fournit directement au système de contrôle les coordonnées du point cible dans le repère relatif au dirigeable. Dans le système réel, ces informations seront données par le système GVPP développé par la société BEV, système de traitement d'images intégré qui permet de suivre des

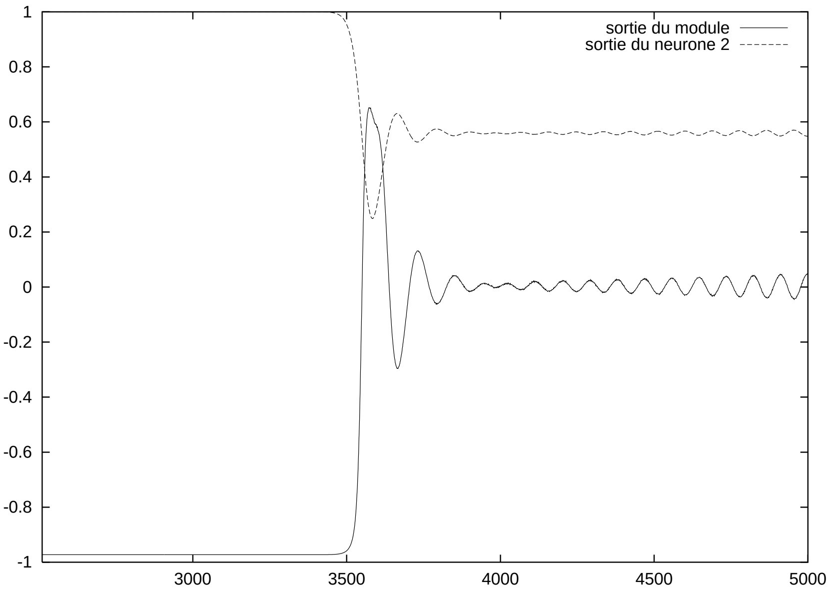


FIG. *8.24: Sortie du module m1 au niveau du neurone 3 et sortie du neurone 2. Mesures réalisées entre les pas de temps 2500 et 5000 de l'évaluation représentée sur la figure 8.18*

objets en temps réel[60].

Nous avons utilisé la même fonction de fitness que dans la section précédente en prenant en compte l'erreur de position (erreur sur x et y) en plus de l'erreur de roulis de tangage et d'altitude.

Le réseau dispose de 5 entrées indiquant les erreurs sur les différents degrés de liberté à maintenir : roulis, tangage, x, y[61] et z. Il dispose d'une sortie pour chacun des sept moteurs.

8.2.1 Résultats

Parmi toutes les expériences que nous avons lancées[62], aucune n'a permis de contrôler tous les degrés de liberté en même temps. Certaines expériences ont généré des réseaux capables de contrôler l'assiette et la position[63], d'autres expériences ont généré des réseaux capables de contrôler l'altitude et l'assiette[64]. Afin de laisser de plus grandes chances aux individus lors de l'évolution, nous avons doublé la valeur de l'angle limite sur le roulis et le tangage - de $0,35 rad$ sur les expériences précédentes - il passe à $0,70 rad$.

Nous avons lancé des expériences dans trois séries différentes : une série témoin avec des modules P uniquement, une série avec modules P, I et D (signes non imposés) et une série avec des modules recopieurs et inverseurs. Cette série dispose, en plus du module P, d'un module

[60]un exemple d'utilisation de GVPP est fourni dans Gourichon et al. (2002)
[61]x et y sont donnés dans le repère relatif au dirigeable.
[62]En comptant toutes les variantes que nous avons essayées, on atteint le chiffre de 150 expériences.
[63]L'altitude restant dans ces expériences à une valeur quasiment constante, sans suivre les consignes demandées.
[64]La position ne se rapprochant pas de la cible.

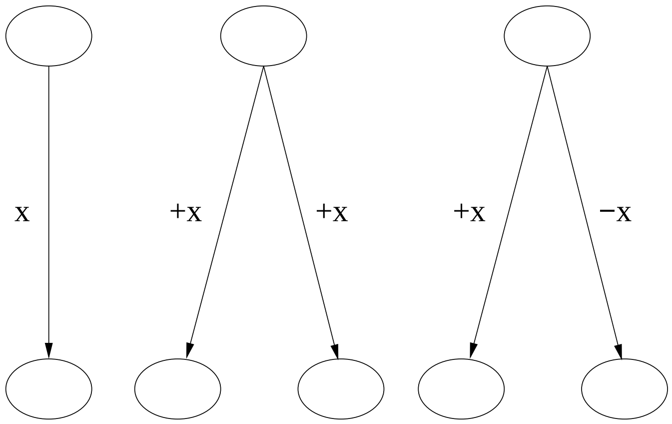


FIG. *8.25: Modules P à une ou deux sorties (égales ou opposées). À gauche, module P simple. Au milieu, module P recopieur à deux sorties. À droite, module P recopieur/inverseur à deux sorties. Les x sont des paramètres réels soumis à évolution (ils font partie de l'intervalle* $[-10;10]$*).*

recopieur à deux sorties[65], d'un recopieur à trois sorties et d'un recopieur/inverseur[66] (figure 8.25). Ces différents modules étaient supposés faciliter la découverte de symétries et d'anti-symétries inhérentes au contrôle du dirigeable complet. En effet, le contrôle de roulis peut être réalisé en envoyant deux commandes opposées aux moteurs verticaux situés de part et d'autre de l'axe longitudinal de l'appareil (moteurs 2 et 3). Le contrôle d'altitude, quant à lui, peut-être réalisé en envoyant des commandes identiques aux moteurs verticaux (soit aux trois, soit aux deux moteurs verticaux arrières[67]). L'erreur en position longitudinale peut être corrigée en envoyant les mêmes commandes aux deux moteurs de propulsion arrières et l'erreur en position latérale peut être corrigée par rotation autour de l'axe de lacet, à l'aide de commandes opposées sur les moteurs horizontaux situés de part et d'autre de cet axe.

Nous avons réalisé six expériences dans chacune de ces séries. Aucune expérience n'est parvenue à maintenir tous les degrés de liberté. L'évolution de la fitness maximum ne favorise aucune des trois séries (figure 8.26). Les expériences avec modules P, I et D sont parties avec un retard qu'elles ont toutefois rattrapé avant la $500^{ème}$ génération. La fitness minimum est plus élevée sur les deuxième et troisième séries (sauf à la $500^{ème}$ génération pour la troisième série). La différence la plus marquée porte sur l'évolution de la fitness moyenne qui est nettement plus élevée pour les expériences avec modules P, I et D (elle est plus élevée de 1 à 4 centièmes de points[68]). L'utilisation de modules recopieurs a provoqué un résultat moins marqué, mais elle a tout de

[65] Module ayant une entrée reliée à ses deux sorties par le biais de deux connexions de même poids.

[66] Module ayant une entrée reliée à ses deux sorties par le biais de connexions de signe opposé

[67] Fonctionne si le tangage est correctement contrôlé.

[68] Avec la fitness utilisée, des différences de centièmes de point correspondent à des différences comportementales importantes. Sur les angles, cette différence correspond à des écarts de valeurs moyenne de $0,5$ à 4 degrés. Sur les positions, cela correspond simultanément à des écarts de valeur moyenne allant de $0,2$ à $0,8$ mètres pour x et y et 1 à 4 mètres pour l'altitude. Les notes considérées étant des valeurs moyennes, l'écart peut être plus faible sur un critère et plus important sur un autre.

Génération	**Donnée**	**Témoin**	**P, I et D**	**recopieurs, inverseurs**
20	Maximum	100,775	100,724	100,775
	Moyenne	100,616	100,646	100,630
	Minimum	100,523	100,553	100,561
100	Maximum	100,775	100,737	100,792
	Moyenne	100,663	100,675	100,663
	Minimum	100,576	100,598	100,58
500	Maximum	100,802	100,803	100,802
	Moyenne	100,683	100,723	100,708
	Minimum	100,608	100,626	100,602

FIG. *8.26: Statistiques des expériences lancées sur le contrôle de roulis, tangage, altitude et position. Valeurs maximum, moyenne et minimum de la fitness sur 6 expériences.*

même entraîné une augmentation de la fitness moyenne.

Dans ces séries d'expériences, nous pouvons constater que l'ajout de connaissances a priori au travers de modules prédéfinis ne permet pas, du moins dans la limite des générations calculées, d'exhiber de différence importante de performances au niveau des meilleurs individus générés. En particulier, l'ajout de ces modules ne permet pas de résoudre le problème complet de l'asservissement du dirigeable. Par contre, les performances moyennes des différentes expériences sont plus élevées pour les séries avec modules prédéfinis.

Réseaux capables de maintenir la position

Le réseau 4 (figure 8.27) est l'un des réseaux les plus efficaces. Il a été obtenu après 3500 générations. Il maintient efficacement la position (figure 8.28), mais l'assiette est moins bien contrôlée que dans les expériences précédentes (portant sur le contrôle de l'assiette et de l'altitude uniquement). Le tangage a une valeur moyenne de $0,04rad$ pendant l'évaluation et le roulis $0,06rad$, ce qui demeure cependant très faible et largement suffisant en pratique.

La structure du réseau est loin de correspondre à ce que nous aurions défini manuellement. La connexion directe entre le neurone 12 et 5 permet le contrôle du tangage. Les moteurs 4, 5 et 6 reçoivent les mêmes commandes depuis le neurone 6 au travers d'un même module I. Le moteur 7 dispose de commandes différentes, issues toutefois du même neurone, mais traversant un module différent.

Les commandes envoyées sont de type “bang-bang” (figure 8.29). Celles qui sont envoyées aux moteurs 4 et 5 ont une période approximative de $50ms$. L'enveloppe de cette commande suit une courbe en créneaux périodique dont la fréquence varie en fonction des conditions (sa valeur moyenne est de l'ordre de $2,5s$). La commande envoyée au moteur 6 est également de type “bang-bang”, mais avec une période plus importante variant entre $1,7s$ et $2,2s$.

Le réseau 5 (figure 8.30) parvient également à contrôler efficacement la position du dirigeable

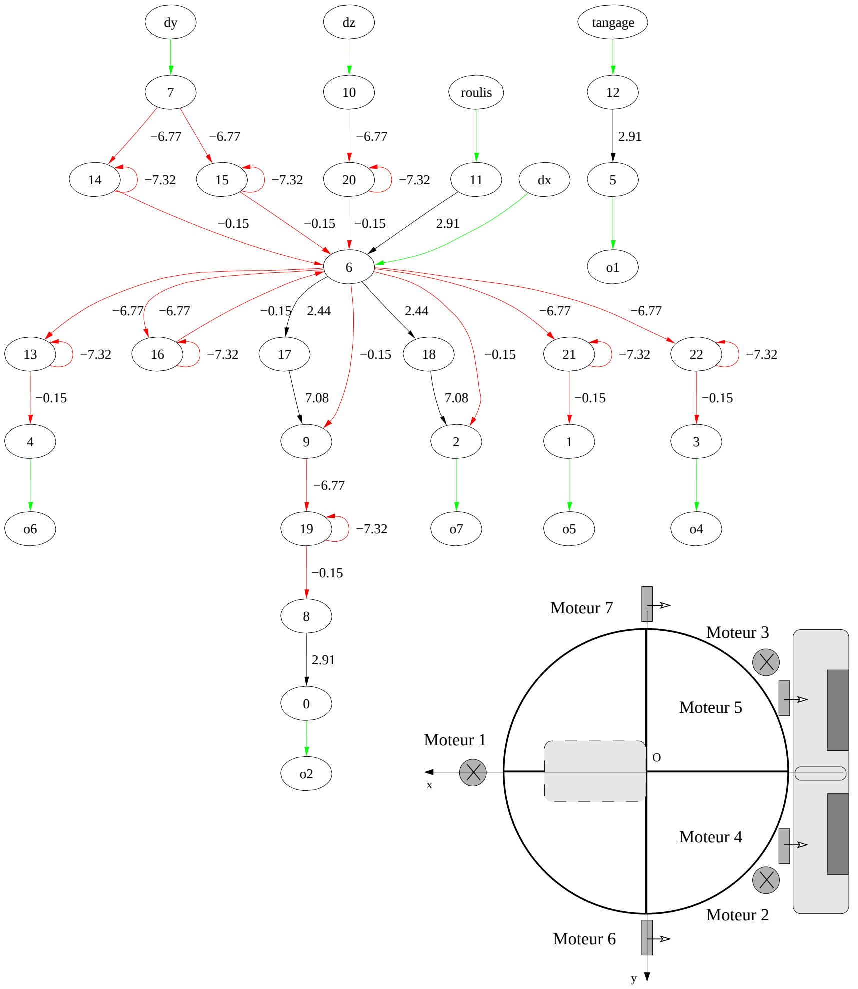


FIG. *8.27: Réseau 4 généré après 3500 générations. Le réseau utilise trois modules différents, un module I utilisé à 8 reprises, un module D utilisé à deux reprises et un module P à trois reprises. La sortie o_i du réseau est reliée au moteur numéro i.*

(figure 8.31). Il ne contient pas de récurrences et correspond à une solution plus proche de ce que l'on aurait attendu intuitivement. De plus, contrairement à la solution précédente, celle-ci utilise une stratégie compatible avec le fonctionnement des moteurs (figure 8.32) : les commandes varient de manière continue, sauf pour les moteurs 4 et 5, qui ont un fonctionnement de type “bang-bang”, mais à des periodes suffisamment importantes pour pouvoir être appliquées sur les moteurs réels sans risquer de les détériorer. Cette solution est donc exploitable sur le dirigeable réel.

Aucune des expériences n'a permis de contrôler tous les degrés de liberté. L'ajout de modules dans le réservoir initial mis à la disposition de l'évolution n'ayant pas été suffisant, nous allons donc présenter des expériences exploitant un autre type de connaissances : le patron de réseau.

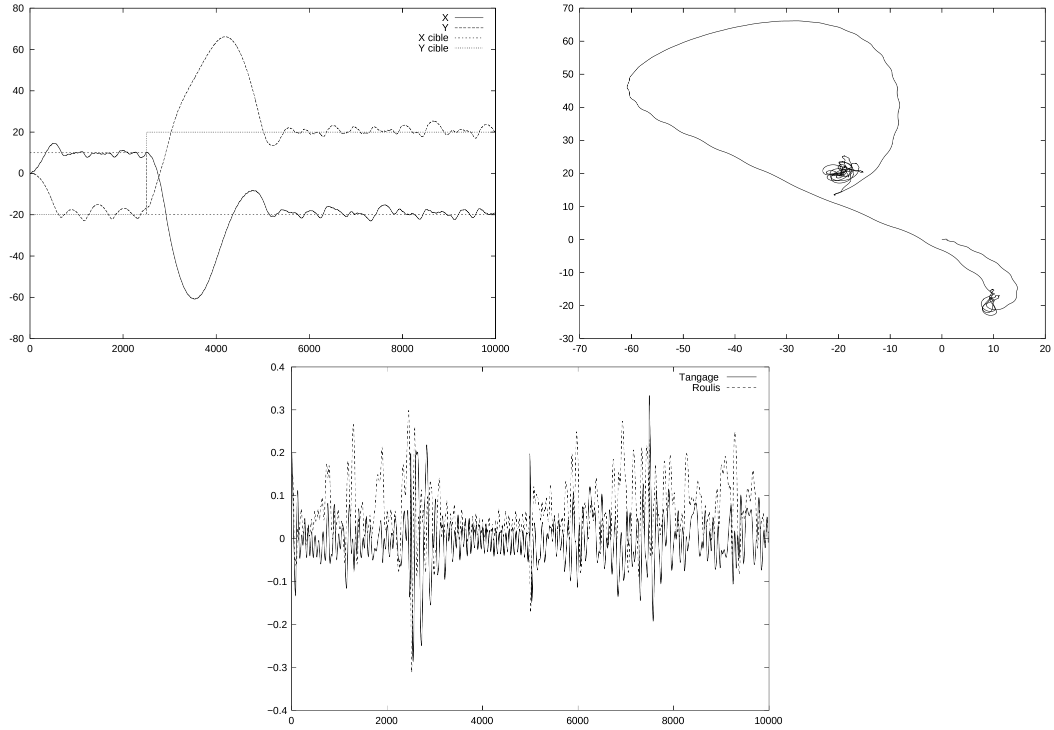


FIG. *8.28: Comportement du dirigeable contrôlé par le réseau 4. Un pas de temps correspond à 25 ms, les positions sont en mètres et les angles en radians. La figure de gauche représente les variations de la position absolue du dirigeable en fonction du temps, la figure de droite représente la trajectoire suivie par le dirigeable (x en abscisse, y en ordonnée) : le dirigeable part de la position (0,0). La première cible est en (10,-20), la deuxième en (-20,20). La figure du bas représente les variations du roulis et du tangage.*

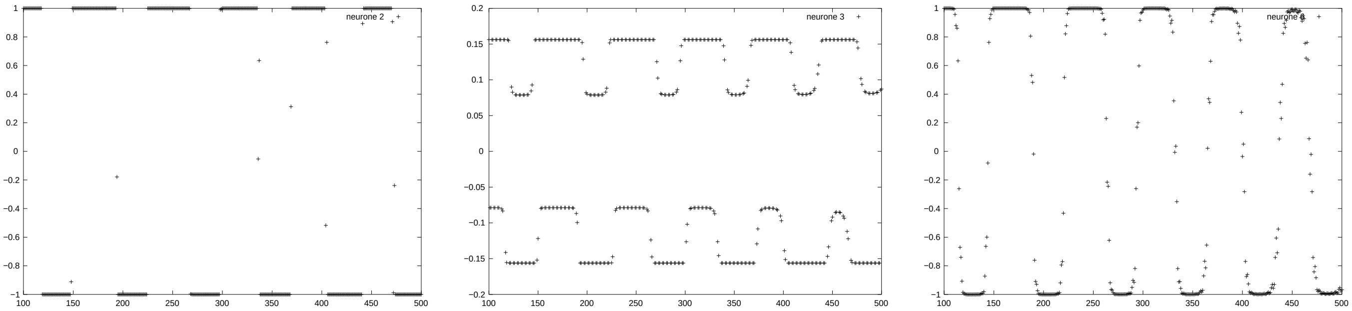


FIG. *8.29: Sorties des neurones 2 et 3 (envoyées aux moteurs 4 et 7) ainsi que du neurone 6 (neurone central) du réseau 4. La sortie du neurone est en abscisse et le temps en ordonnée.*

8.2.2 Maintien complet avec un "patron" de réseau

Dans ces expériences, nous continuons à traiter le problème en un seul bloc : l'évolution doit trouver un contrôleur capable de maintenir tous les degrés de liberté sans qu'un expérimentateur

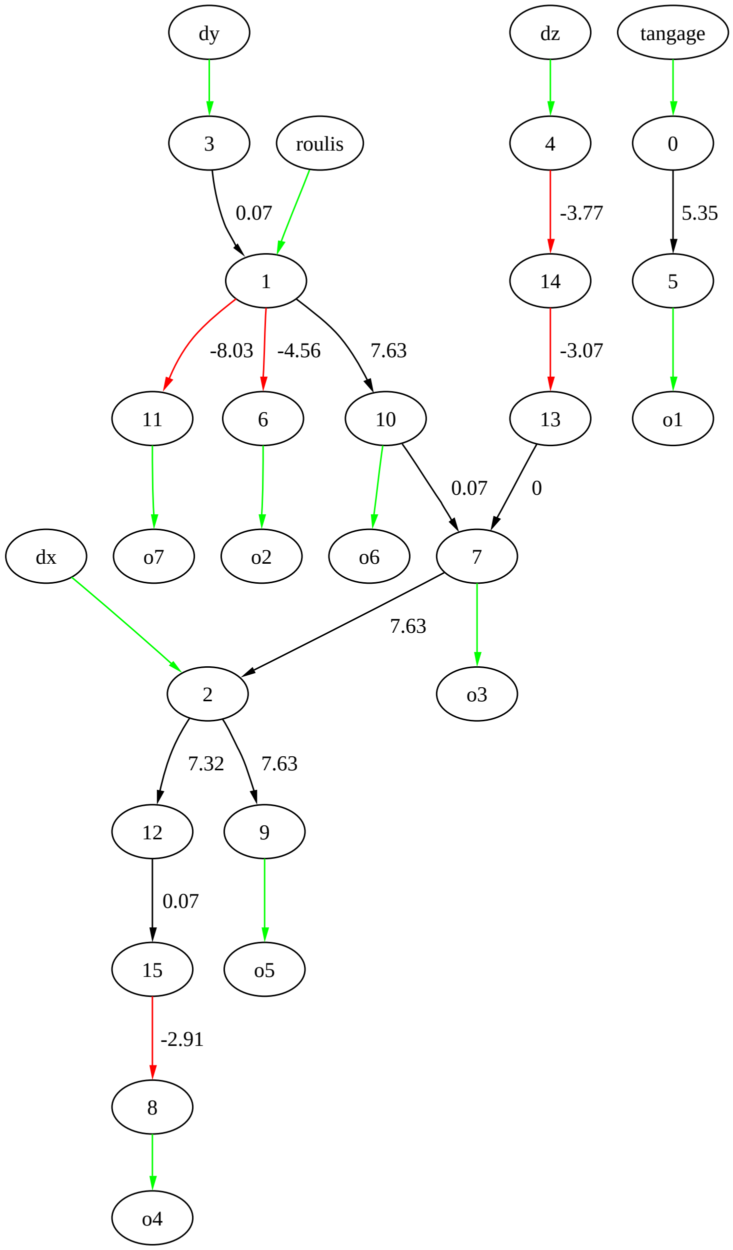


FIG. *8.30: Réseau 5 généré après 1800 générations. Réseau obtenu à partir d'une expérience dans laquelle le réservoir de modules initiaux ne contient qu'un module P.*

n'opère de décomposition fonctionnelle.

Nous avons lancé des expériences en exploitant le patron décrit sur la figure 8.33. Ce patron contient un module à une entrée, une sortie et quatre modules à une entrée et deux sorties. Ce patron a été conçu à partir d'une analyse préliminaire, ainsi que d'observations réalisées sur les expériences précédentes.

Le contrôle de tangage est ainsi réalisé dans le patron par un module reliant le capteur de

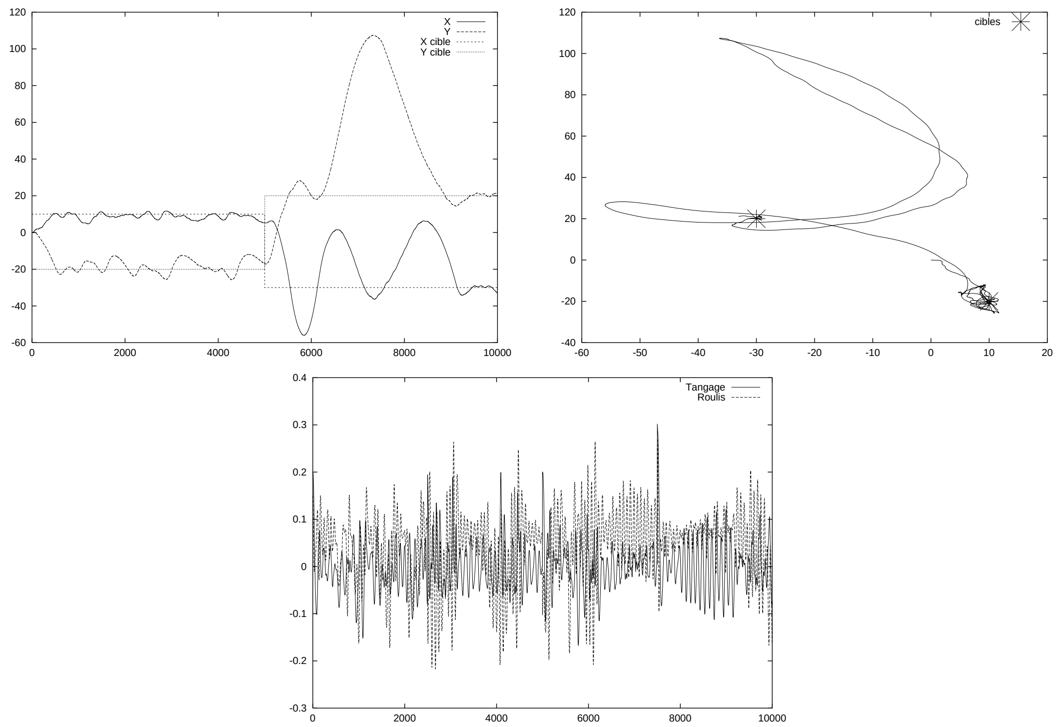


FIG. *8.31: Comportement du dirigeable contrôlé par le réseau 5. Un pas de temps correspond à 25 ms, les positions sont en mètres et les angles en radians. La figure de gauche représente les variations de la position absolue du dirigeable en fonction du temps, la figure de droite représente la trajectoire suivie par le dirigeable (x en abscisse, y en ordonnée). Le roulis et le tangage figurent sur les courbes du bas.*

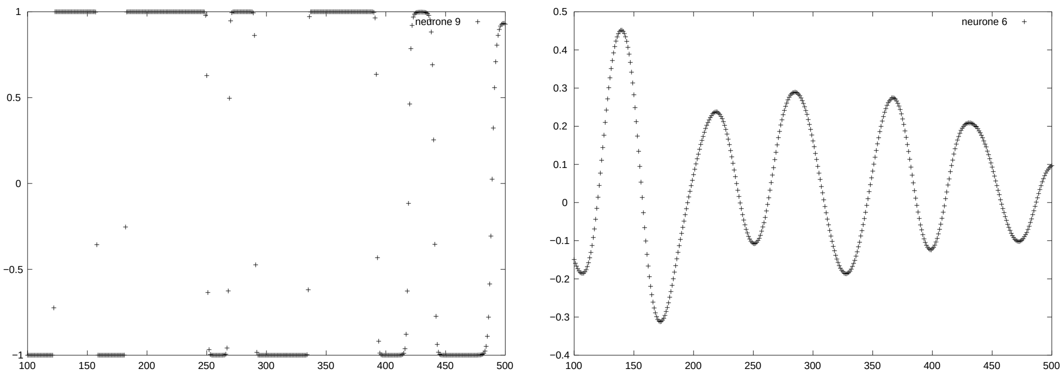


FIG. *8.32: Sorties des neurones 9 et 6 (envoyées aux moteurs 5 et 2) du réseau 5 illustrant les deux stratégies de contrôle présentes dans ce réseau : contrôle "bang-bang", mais avec une fréquence faible et contrôle continu. La sortie des neurones est en abscisse et le temps en ordonnée.*

tangage à la sortie o_1. Cette technique ayant été observée dans toutes les solutions que nous avons étudiées. Le maintien de l'altitude est prévu d'être assuré par un module à une entrée et deux sorties reliant le capteur d'erreur d'altitude aux neurones de sortie o_2 et o_3, cette solution ayant été observée dans certains réseaux capables de maintenir l'altitude et l'assiette dans des expériences de maintien complet (réseaux non représentés ici). Les capteurs de x et de y sont reliés aux neurones de sortie o_4 et o_5, et respectivement o_6 et o_7, ce résultat étant tiré de l'analyse du réseau 5. Le contrôle de roulis, supposé être implémenté grâce au module reliant le capteur de roulis aux neurones o_2 et o_3, a été déterminé par analyse du fonctionnement de l'appareil.

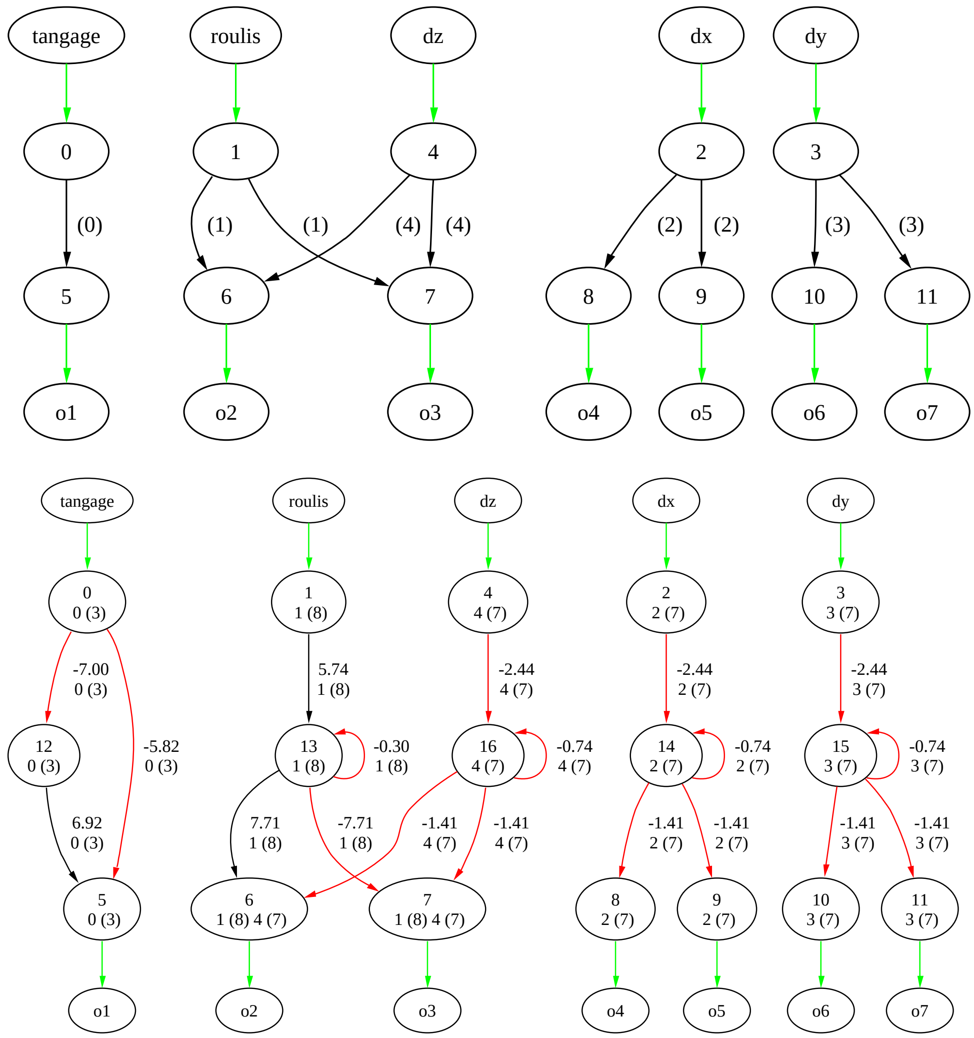


FIG. *8.33: En haut, patron fourni à l'évolution. Cette structure suggère comment les modules pourraient être reliés entre eux et avec les entrées/sorties. Seuls comptent ici les liens entre modules, la structure interne de chaque module étant choisie pendant la génération aléatoire. Les connexions ayant un même numéro appartiennent au même module. En bas, exemple de réseau généré à partir du patron. Ce réseau utilise un module D à une sortie et des modules I à deux sorties (soit égales, soit opposées).*

Nous avons testé trois réservoirs de modules-modèles différents. Le premier contient un module P à une entrée, une sortie et deux autres modules P à une entrée et deux sorties. Dans le premier, les deux connexions sont de même poids ; dans le deuxième, elles sont de poids opposés. Le deuxième réservoir contient, en plus des modules précédemment cités, des modules I et D à une sortie et à deux sorties identiques ou opposées. Le troisième réservoir utilisé contient les mêmes modules que le deuxième, mais avec certains poids imposés (figure 8.34).

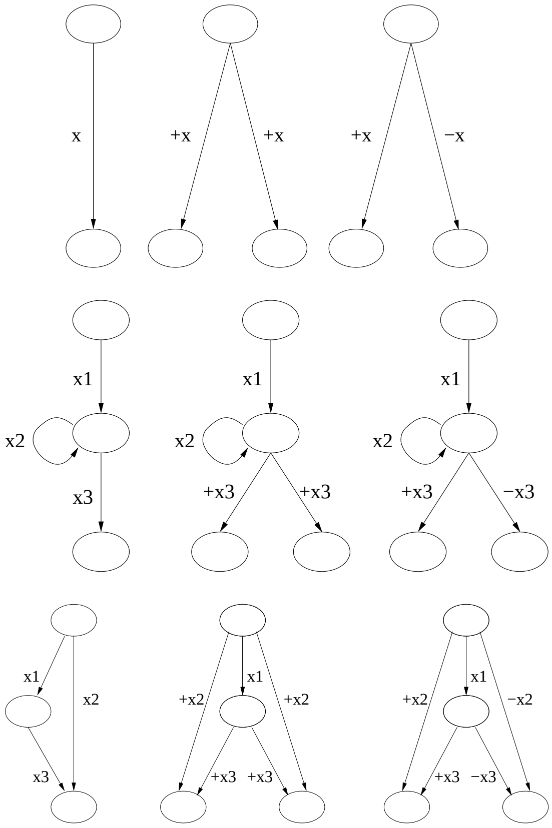


FIG. *8.34: Modules P (ligne du haut), I (ligne du milieu) et D (ligne du bas) à une ou deux sorties (égales ou opposées). Les x et x_i sont des paramètres réels soumis à évolution (ils font partie de l'intervalle $[-10; 10]$, sauf la connexion récurrente des modules I qui appartient à l'intervalle $[-1; 1]$). Dans les expériences avec signes imposés, x_2 est de signe positif dans les modules I, x_1 et x_2 sont positifs et x_3 sont négatifs dans les modules D.*

Les statistiques obtenues sur ces expériences sont nettement supérieures à celles obtenues sans patron, tant au niveau du meilleur individu généré que du plus mauvais ou de la moyenne (figure 8.35). Sur trente expériences qui ont atteint la $2000^{ème}$ génération, toutes sauf deux ont convergé (elles ont dépassé le seuil de $100,8$). Si on compare les séries entre elles, les expériences avec modules P uniquement ont nettement l'avantage pendant les premières générations, mais ce n'est plus le cas à la $500^{ème}$ génération. À la $500^{ème}$ génération, trois expériences de la deuxième série et de la troisième série ont une fitness maximale supérieure à $100,820$, meilleure fitness obtenue par des individus de la première série. Les expériences de la deuxième série ont donc

Génération	Donnée	P	P, I et D	P, I et D avec signes imposés
20	Maximum	100,815	100,741	100,745
	Moyenne	100,701	100,690	100,650
	Minimum	100,626	100,629	100,602
100	Maximum	100,817	100,826	100,82
	Moyenne	100,764	100,73	100,693
	Minimum	100,665	100,645	100,608
500	Maximum	100,820	100,858	100,873
	Moyenne	100,778	100,77	100,759
	Minimum	100,665	100,654	100,648

FIG. *8.35: Statistiques des expériences lancées sur le contrôle de roulis, tangage, altitude et position avec patron de réseau. Valeurs maximum, moyenne et minimum de la fitness sur 10 expériences. Le réservoir de la première série contient des modules à une ou deux sorties et le réservoir de la deuxième série contient des modules P, I ou D à une ou deux sorties.*

eu plus de mal à démarrer, cependant elles se sont révélées plus performantes sur le long terme. Cet écart peut s'expliquer par le fait que les modules I et D ayant plus de paramètres que les modules P, l'évolution a besoin de plus de temps pour trouver des réseaux efficaces. Par contre, une fois que l'évolution est parvenue à exploiter leurs caractéristiques, les performances s'en sont trouvées nettement améliorées.

Le réseau 6 (figure 8.36) a été généré après 2000 générations dans la troisième série d'expériences (avec modules P, I et D à une et deux sorties dont certains signes sont imposés). C'est à noter que sa structure correspond au patron à un détail près : le module D à deux sorties relié au capteur de roulis n'est pas relié aux neurones de sortie o_2 et o_3, mais à o_3 et au neurone 3, lui-même relié aux moteurs o_6 et o_7. Quoiqu'il en soit, ce résultat démontre que le recours à des patrons pour faciliter le démarrage de l'évolution n'impose pas la structure finale des réseaux obtenus.

De même que dans les expériences précédentes, le maintien d'assiette est moins performant que dans les expériences portant uniquement sur l'assiette et l'altitude (figure 8.37). Il est néanmoins efficace pendant les phases transitoires (le tangage moyen est en valeur absolue de $0,07rad$ et le roulis de $0,05rad$). En régime permanent, cependant, il provoque des oscillations entretenues à la fois sur le roulis et le tangage (figure 8.38). Les oscillations de tangage ont une période de $3,5s$ pour une amplitude de $0,18rad$. Les oscillations de roulis ont une période de $40s$ et une amplitude de $0,05rad$. Étant donné la lenteur du maintien de l'altitude, ce régime de fonctionnement ne s'est que rarement présenté pendant les évaluations des individus, aussi la pression sélective sur cette phase du contrôle n'a-t-elle pas été suffisante[69].

La position est cependant efficacement maintenue, de même que l'altitude. Le contrôle d'alti-

[69]De nouvelles expériences avec une évaluation plus longue devraient être relancées pour accroître la pression sélective sur le régime permanent.

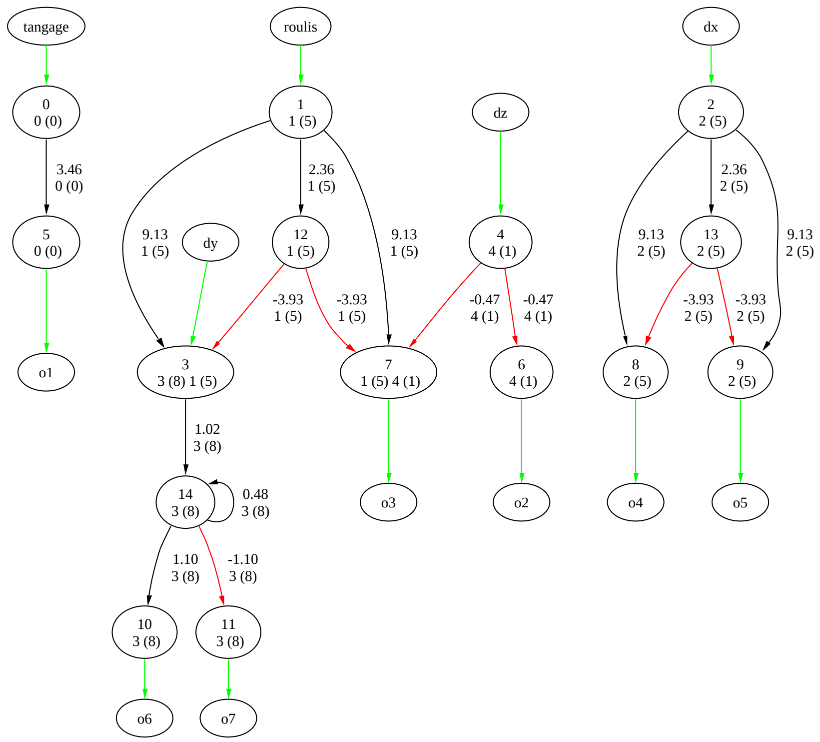


FIG. *8.36: Réseau 6 généré après 2000 générations. Réseau obtenu à partir d'une expérience dans laquelle le réservoir de modules initiaux contient des modules P, I et D à une ou deux sorties (sorties identiques ou de signes opposés avec certains signes imposés).*

tude est précis, mais beaucoup plus lent que sur les expériences portant sur le maintien d'assiette et d'altitude. En effet, les effets aérodynamiques exploités lors de ces précédentes expériences ne peuvent plus être utilisés ici, car le lacet n'est plus libre (il est contrôlé par les moteurs à poussée horizontale). Le dirigeable ne peut donc plus se mettre passivement face au vent et exploiter la portance de l'appareil, à moins d'inhiber ses moteurs horizontaux dans un premier temps, ce qui n'est pas évident à implémenter compte tenu des éléments dont dispose l'évolution[70]. De plus, le réseau n'a pas les capteurs nécessaires pour déterminer la direction du vent et ne peut pas reconstruire aisément cette information ; il ne peut donc pas se mettre face au vent de manière active non plus. La vitesse de montée est de $1,15 m.s^{-1}$, ce qui est plus rapide que les $0,96 m.s^{-1}$ des contrôleurs PID que nous avons générés manuellement. Si le maintien de position est stable après une phase oscillatoire, dans certains cas, le maintien de position longitudinale exhibe une erreur statique (de moins de $4m$ sur la figure 8.38).

[70]Il faudrait ajouter des modules de type sigma-pi, qui pourraient potentiellement inhiber les neurones auxquels ils seraient connectés. Nous avons fait quelques expériences dans ce sens, sans avoir obtenu de résultat efficace.

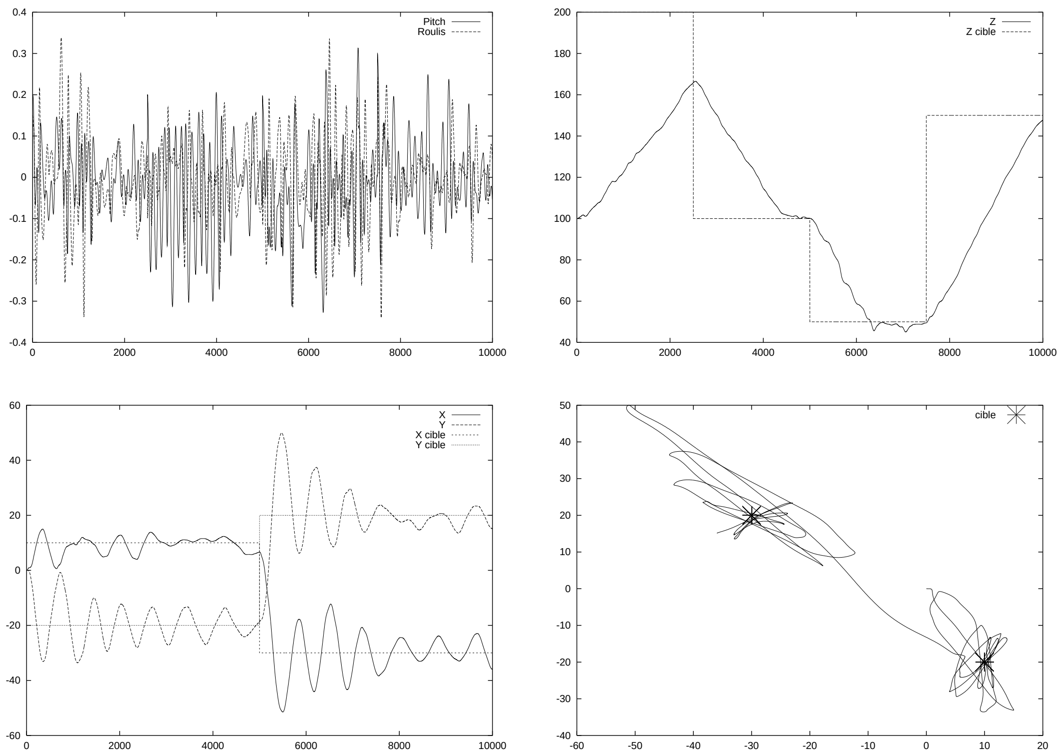


FIG. 8.37: *Comportement du dirigeable contrôlé par le réseau 6. Un pas de temps correspond à 25 ms, les positions sont en mètres. En haut à gauche, variations d'assiette, à droite d'altitude. En bas à gauche, variations de la position absolue du dirigeable en fonction du temps, à droite trajectoire suivie par le dirigeable (x en abscisse, y en ordonnée). Ce réseau parvient à contrôler l'assiette, l'altitude et la position, contrairement aux expériences sans patron qui ne contrôlaient pas soit l'altitude, soit la position.*

8.3 Effet du choix de la fonction de fitness

Dans les résultats présentés précédemment, nous avons utilisé la moyenne pour générer une note unique à partir de l'ensemble de critères indépendants que constituent les erreurs pour chacun des degrés de liberté. Afin de vérifier l'influence de cette fonction d'agrégation, nous avons réalisé des expériences dans lesquelles nous avons remplacé la moyenne arithmétique par une moyenne géométrique. La nouvelle expression de la fitness était donc :

$$f(g) = p(g) + fit_{dec}(g)$$

$$fit_{dec}(g) = \sqrt[n]{\Pi_{i=1}^{n} f_i(g)}$$

avec $f_i(g) = \left(1 - \frac{\sum_t (d_i(g,t)^2)}{T}\right)$.

Hormis la fonction de fitness, les autres paramètres de l'expérience étaient ceux de la meilleure série d'expériences précédemment réalisée, à savoir la série d'expériences avec patron et avec

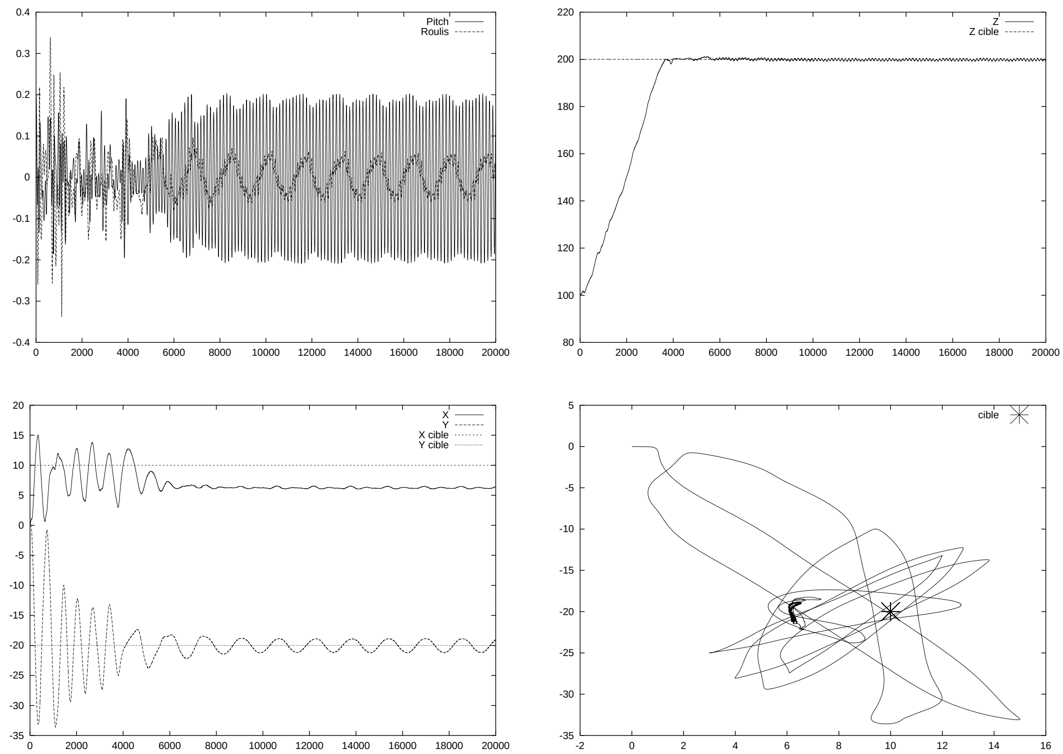


FIG. *8.38: Comportement du dirigeable contrôlé par le réseau 6. Étude de la stabilité du contrôle en régime permanent.*

modules P, I et D avec signes imposés dans le réservoir.

Les statistiques que nous avons réalisées (figure 8.39) révèlent un meilleur comportement moyen sur les expériences avec moyenne géométrique, avec un écart-type réduit d'un critère à un autre : la moins bonne valeur atteinte sur un critère est plus élevée avec la moyenne géométrique, mais la meilleure valeur est également plus faible. Cependant, parmi toutes les expériences réalisées, c'est une expérience utilisant la moyenne arithmétique qui a donné le meilleur contrôleur (fitness de 100,881), lequel s'avère, critère par critère, meilleur que le meilleur contrôleur généré par une expérience avec moyenne géométrique (sa fitness, calculée avec une moyenne arithmétique, pour être comparable à la précédente valeur donnée, est de 100,861).

Utiliser la moyenne géométrique plutôt qu'arithmétique a donc tendance à lisser les performances obtenues sur chacun des critères : les performances les plus faibles sur un critère sont plus élevées, mais les performances les meilleures sont également plus faibles. Au final, les expériences sont, en moyenne, meilleures, mais le meilleur résultat obtenu est plus faible que celui qui a été obtenu avec une moyenne arithmétique.

Génération	**Donnée**	**Moy. arithmétique**	**Moy. géométrique**
20	Maximum	0,979	0,964
	Moy. arithm.	0,652	0,660
	Moy. géom.	0,583	0,610
	Minimum	0,285	0,335
	Écart-type	0,305	0,282
100	Maximum	0,980	0,962
	Moy. arithm.	0,712	0,685
	Moy. géom.	0,661	0,644
	Minimum	0,376	0,387
	Écart-type	0,262	0,255
500	Maximum	0,979	0,973
	Moy. arithm.	0,760	0,787
	Moy. géom.	0,726	0,764
	Minimum	0,457	0,513
	Écart-type	0,215	0,188

FIG. *8.39: Statistiques des expériences lancées avec moyenne arithmétique ou géométrique pour agréger les différents critères. Les expériences portent sur le contrôle de roulis, tangage, altitude et position avec patron de réseau. Les différentes grandeurs présentées dans le tableau sont mesurées par rapport à l'ensemble des notes obtenues pour chacun des critères des meilleurs individus générés, ces critères étant l'erreur pour chacun des degrés de liberté. Les valeurs présentées sont les moyennes (arithmétiques) réalisées sur 10 expériences.*

8.4 Discussion

Le dirigeable ALPhA est dotée de nombreux capteurs et effecteurs et engendre une dynamique impliquant des couplages non linéaires qui en font une plateforme de test très intéressante. Les expériences réalisées sur le maintien de l'assiette et de l'altitude ont démontré que l'évolution était capable d'exploiter les caractéristiques physiques de l'appareil pour générer une solution efficace. Une approche incrémentale directe avec un découpage trop rapide du problème nous aurait conduit à maintenir séparément chacun des différents degrés de liberté. Une solution de ce type est au mieux trois fois plus lente que l'approche qui a été trouvée par l'évolution.

Les expériences visant à maintenir les cinq degrés de liberté : roulis, tangage, altitude et position (x et y) sans utiliser de patron ne sont pas parvenues à une convergence parfaite. Elles ont réussi à garder l'assiette à l'intérieur des limites imposées par le domaine de vol sans réussir à la stabiliser complètement. Ensuite, elles se sont divisées en deux catégories : certaines expériences ont réussi à maintenir la position, d'autres l'altitude. Parmi toutes les expériences que nous avons lancées, seules les expériences dont la population initiale a été préparée avec un "patron" ont généré des solutions capables de contrôler les cinq degrés de liberté. Parmi ces expériences, celles qui disposaient de modules P, I et D ont convergé plus lentement que celles qui ne disposaient que de modules P mais elles ont, cependant, convergé vers des solutions plus efficaces.

Chapitre 9

Contrôle d'un hélicoptère

L'hélicoptère est un aéronef très maniable, susceptible de se maintenir en position stationnaire au-dessus d'un objectif. Il est capable de décoller et d'atterrir verticalement avec précision.

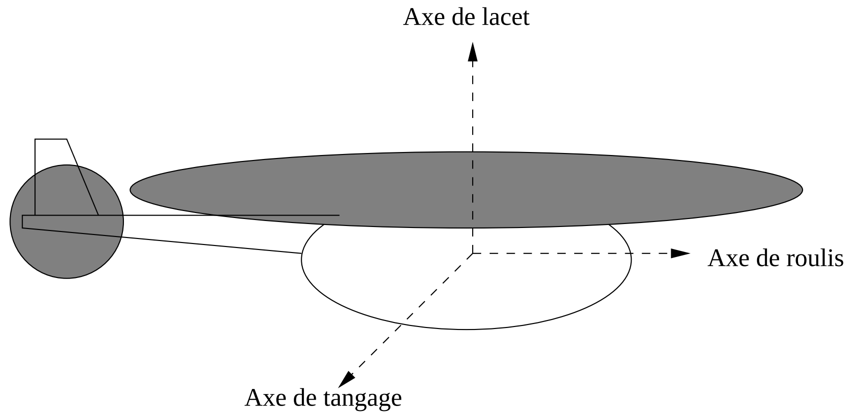


Fig. *9.1: Axes de l'hélicoptère.*

9.1 Principe de fonctionnement

Une compréhension approfondie de l'hélicoptère n'est pas indispensable pour trouver une architecture de contrôle avec des algorithmes évolutionnistes, excepté lors de la programmation de la simulation. La connaissance et la compréhension du fonctionnement des commandes et de leurs actions sur la dynamique de l'hélicoptère n'est pas indispensable non plus. Nous allons cependant les rappeler ici pour pouvoir expliquer ensuite le fonctionnement des contrôleurs obtenus par évolution.

Les principes fondamentaux du fonctionnement d'un hélicoptère sont les mêmes que ceux d'un avion car les pales de l'hélicoptère se comportent comme des ailes. Il s'ensuit que l'air n'a pas la même distance à parcourir pour passer au-dessus et en dessous d'une aile et que le chemin par en dessous (intrados) est beaucoup plus court que le chemin par en dessus (extrados). La vitesse de l'air étant plus importante sur l'extrados que sur l'intrados, il se crée une pression sur l'intrados et une dépression sur l'extrados. Ces pressions-dépressions créent une portance qui tire l'aile vers le haut. La portance augmente avec la vitesse relative de l'air et, dans une certaine mesure, avec l'angle d'incidence (angle entre l'aile et la vitesse de l'air)[71]. Pour un hélicoptère, la somme des portances des différentes pales du rotor forme la portance globale de l'hélicoptère, une force perpendiculaire au plan de rotation. C'est cette force qui maintient l'hélicoptère en l'air. Le pilote peut la faire varier en modifiant la partie constante pendant une révolution de l'angle d'incidence des pales. Cette commande est appelée le "collectif principal".

La rotation des pales provoque la rotation de l'hélicoptère en sens inverse. Ce phénomène de réaction est contré par le rotor placé sur la queue de l'appareil. Le pilote peut faire varier la commande de ce rotor, commande appelée "collectif de queue", afin de faire tourner l'hélicoptère autour de son axe de lacet.

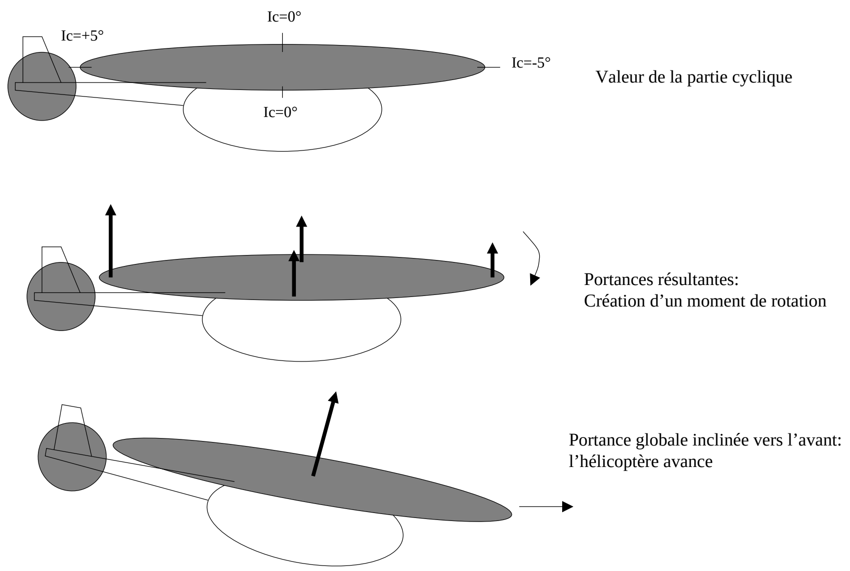


FIG. *9.2: Principes de la commande cyclique. Ic est la partie cyclique de l'angle d'incidence.*

Nous venons de voir comment faire monter ou descendre l'hélicoptère et comment le faire tourner autour de son axe de lacet. Il nous reste à étudier les commandes permettant à l'hélicoptère d'avancer ou d'aller sur les côtés. Ces deux manœuvres exploitent le même principe : l'utilisation d'une variation cyclique de l'angle d'incidence pour incliner le plan de rotation des pales (figure 9.2). Considérons en effet une révolution de pale et supposons que l'angle d'incidence soit au maximum lorsque la pale passe au-dessus de l'arrière de l'hélicoptère et au minimum lorsque la pale passe au-dessus de l'avant. La portance est donc maximale lorsque la pale passe au-dessus de l'arrière de l'hélicoptère et minimale lorsqu'elle passe au-dessus de l'avant[72]. Cette différence entre les portances crée un moment de rotation autour de l'axe transversal du plan de

[71] La portance augmente avec l'angle d'incidence jusqu'à ce que ce dernier atteigne une valeur limite au delà de laquelle la portance diminue brutalement. C'est le décrochage.

[72] En fait, il y a un décalage de 90° entre la position où la portance est maximale et la position où l'angle d'incidence est maximal.

rotation. Ce moment de rotation provoque l'inclinaison du plan de rotation vers l'avant ; la portance globale, qui est perpendiculaire à ce plan, s'incline donc aussi vers l'avant ; la composante de la portance sur l'axe longitudinal de l'hélicoptère est positive : l'hélicoptère avance. Ainsi, en faisant varier la position angulaire du maximum de l'angle d'incidence (le minimum étant toujours à 180° du maximum), on peut provoquer un mouvement de l'hélicoptère vers l'avant ou sur un côté. Cette commande est appelée la “commande cyclique”. La valeur de l'angle d'incidence est donc la somme du collectif (constant au cours d'une révolution) et du cyclique (variable au cours d'une révolution).

Nous connaissons donc à présent les commandes disponibles pour piloter un hélicoptère :

- Collectif principal : manche à balai 1 axe
- Collectif de queue : palonnier
- Cyclique “avant-arrière” et “gauche-droite” : manche à balai 2 axes

Ces commandes sont celles qui ont été mises à disposition des contrôleurs pendant l'évolution.

Pour l'évaluation des solutions proposées par l'évolution, nous avons utilisé le simulateur open source lié au projet autopilot[73] que nous avons interfacé avec notre plateforme de développement SFERES (voir section 6.1).

9.2 Expériences

9.2.1 Maintien d'assiette seulement

Le simulateur du projet autopilot permet de ne laisser libres que certains degrés de liberté pendant que les autres sont figés. Nous avons utilisé cette possibilité pour réaliser nos premières expériences sur l'hélicoptère avec pour objectif de maintenir l'assiette de l'appareil. La position et l'altitude sont figées, seuls le roulis, le tangage et le lacet peuvent varier.

Le réseau de neurones utilisé pour accomplir cette tâche dispose en entrée des angles de roulis et de tangage. En sortie, il agit sur les deux commandes cycliques qui permettent de contrôler ces degrés de liberté.

Conditions expérimentales

La fitness que nous avons utilisée est similaire à celle employée sur le pendule inversé et le dirigeable : on mesure le temps $p(g)$ pendant lequel l'hélicoptère est maintenu dans son domaine de vol (en pourcentage du temps total de l'évaluation), temps auquel on ajoute une partie décimale $fit_{dec}(g)$ permettant de prendre en compte l'écart moyen à la consigne souhaitée pour chaque degré de liberté à maintenir.

[73]http ://autopilot.sourceforge.net/

$$
\begin{aligned}
f(g) &= p(g) + fit_{dec}(g) \\
fit_{dec}(g) &= \frac{\sum_{i=1}^{N} f_i(g)}{N} \\
f_i(g) &= \left(1 - \frac{\sum_t (d_i(g,t)^2)}{T}\right)
\end{aligned}
$$

L'évaluation porte sur quatre expériences durant au total 100s simulées. Au début de chaque expérience, le roulis et le tangage de l'hélicoptère sont positionnés à des valeurs comprises entre $-0,2$ et $0,2$ radians.

Le domaine de vol utilisé est le même que pour le dirigeable. Si le roulis ou le tangage dépassent une valeur limite de $0,35$ radians, l'expérience est immédiatement arrêtée et l'individu s'en trouve pénalisé car la principale partie de sa note (le pourcentage du temps de maintien dans le domaine de vol) prendra cet échec en compte. Cette valeur de $0,35$ radians est utilisée pour normaliser les valeurs des angles apparaissant dans la fitness.

La population utilisée est de 100 individus, avec une stratégie de sélection élitiste dans laquelle on garde 60% des meilleurs individus. Compte tenu de la lenteur de la simulation utilisée, nous ne générons initialement que 100 individus, et non pas 10000 comme lors des expériences portant sur le dirigeable.

Résultats

Toutes les expériences lancées (une dizaine environ) ont été capables très rapidement (en moins de 50 générations) de maintenir l'hélicoptère dans son domaine de vol et de ramener son roulis et son tangage aux valeurs de repos (à savoir 0 pour le roulis et le tangage).

La figure 9.3 montre le comportement d'un hélicoptère contrôlé par un réseau obtenu après 100 générations (figure 9.4). Les deux degrés de libertés sont ramenés à leurs valeurs de repos après des oscillations rapidement amorties.

Le réseau généré comporte deux sous-réseaux distincts : un pour le contrôle du roulis, un pour le contrôle du tangage[74]. Les deux contiennent des récurrences. Le contrôle est continu, il n'utilise pas de technique de type "bang-bang".

Nous avons comparé le fonctionnement de ce réseau à celui généré par le contrôleur d'assiette fourni avec autopilot (contrôleur de type PD), voir figure 9.5. Le contrôle de tangage est un peu plus efficace avec le PD : le dépassement est beaucoup moins important (il varie entre 0,5 et 0,25 pour le PD alors qu'il vaut 0,8 pour le réseau), et le temps de réponse est légèrement inférieur (2,4s contre 3s pour le réseau). Cependant, dans un cas de figure (le troisième), le contrôle est beaucoup plus lent avec le PD ; le contrôle du réseau semble donc plus robuste.

La connexion allant du neurone 2 au neurone 0 intervenant dans le contrôle du tangage a quasiment la valeur maximale autorisée (à savoir -1). Si l'on autorise cette connexion à prendre des valeurs un peu plus importantes, on peut améliorer les performances en baissant le dépassement

[74] Cela n'a pas été imposé à l'évolution.

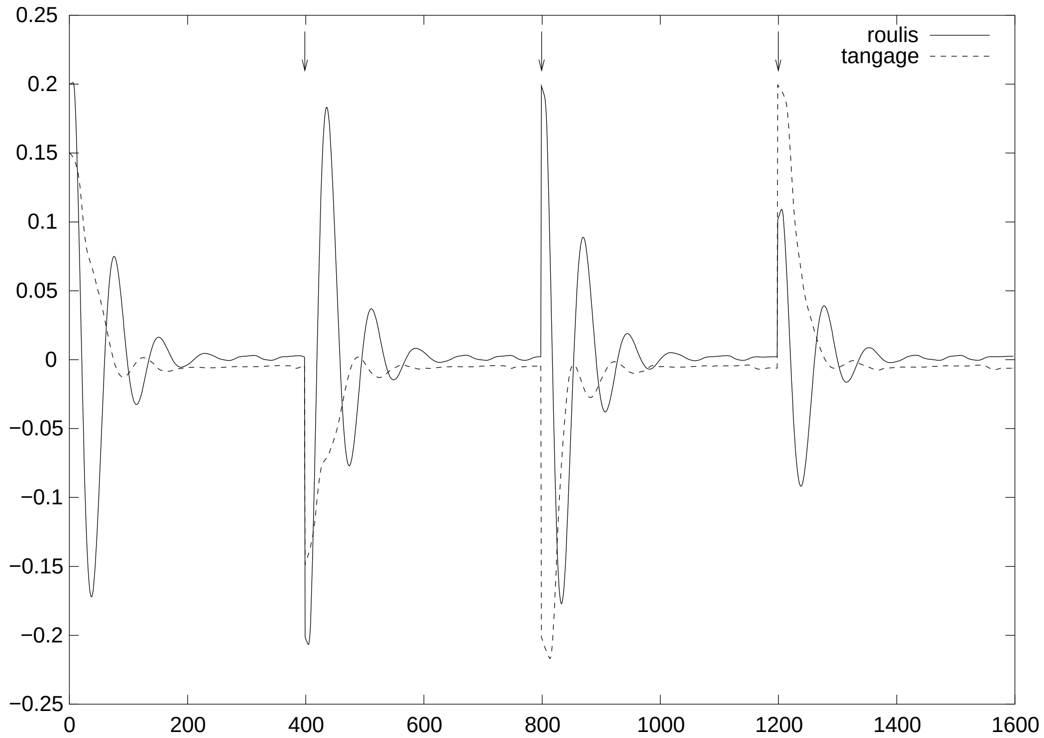


FIG. *9.3: Comportement d'un hélicoptère contrôlé par un réseau de neurones obtenu après 100 générations (voir réseau figure 9.4). Une perturbation est introduite sur le roulis ainsi que sur le tangage tous les 400 pas de temps (10s), aux instants pointés par des flèches, pour vérifier la robustesse du contrôle. Le temps en abscisse est en pas de temps de 0,025s, les angles en ordonnée sont en radians.*

et en diminuant le nombre d'oscillations (figure 9.6), ce qui indique que l'intervalle de valeurs autorisées pour les poids de connexions est trop faible pour optimiser cette partie du contrôleur. Cependant, même avec une valeur plus importante, le contrôle trouvé reste tout de même de moins bonne qualité qu'avec un PD.

Par contre, le contrôle de roulis est nettement plus efficace avec le réseau de neurone. Il n'y a de dépassement avec aucun des deux contrôleurs ; en revanche, le temps de réponse du réseau de neurones se révèle toujours plus faible que celui du PD, l'écart variant entre $0,5s$ et presque $2s$ (le temps de réponse du réseau étant de l'ordre de $1,5s$).

Le contrôle de roulis est donc plus efficace dans sa version réseau de neurones alors que le contrôle de tangage l'est légèrement plus dans sa version PD. La différence est en faveur du réseau de neurones, mais elle demeure relativement faible. Il est ainsi difficile de trancher en faveur d'une approche plutôt que d'une autre. Ce qu'il est intéressant de noter, c'est la différence de point de vue entre les deux approches.

Dans le cas du PD, on se place d'un point de vue d'automaticien en donnant initialement l'erreur ainsi que sa dérivée, tout en indiquant quelles entrées sont reliées à quelles sorties. Le système est alors décrit par un jeu de quatre paramètres seulement (six, si on prend en compte le

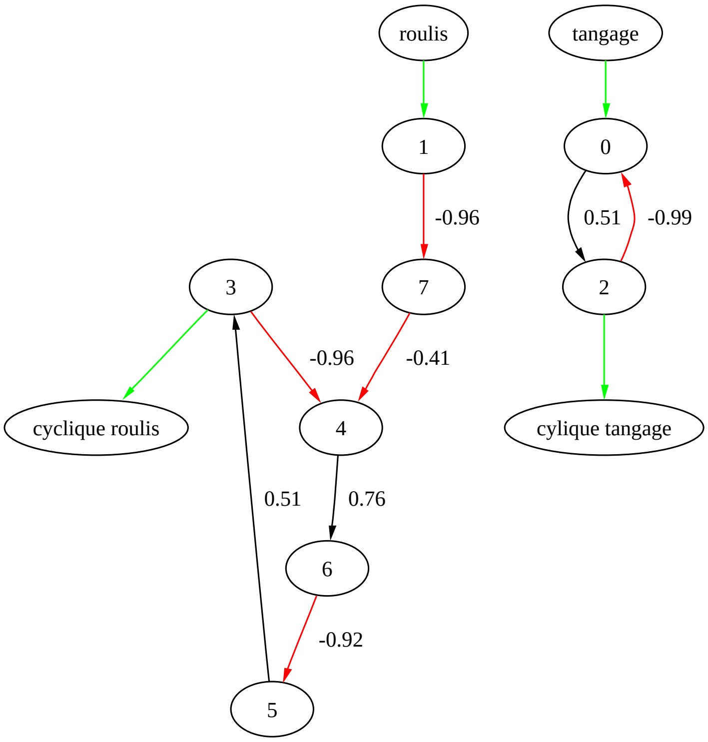


FIG. *9.4: Réseau de neurones obtenu après 100 générations.*

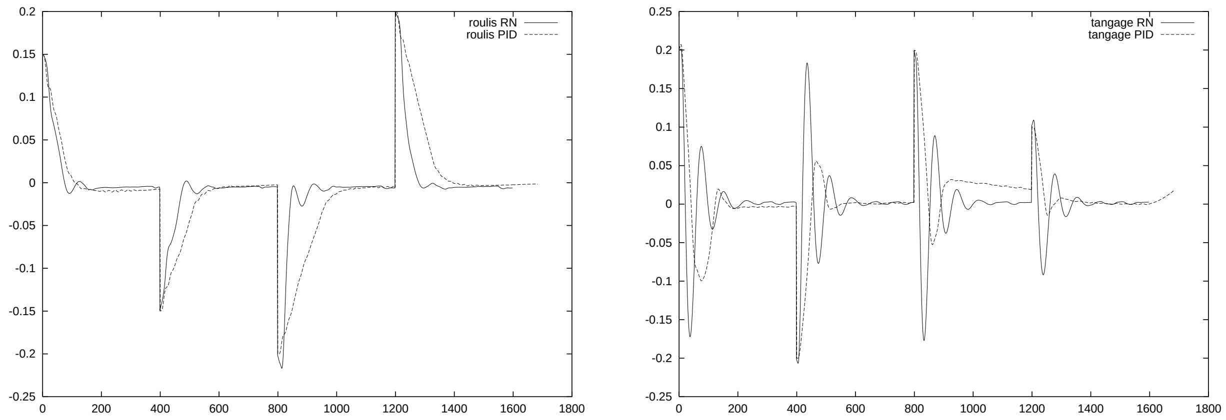


FIG. *9.5: Comparaison entre le comportement généré par le réseau de neurones obtenu par évolution et le contrôleur PID utilisé dans le projet autopilot. Les conditions d'évaluation sont les mêmes que précédemment. Le temps en abscisse est en pas de temps de 0,025s, les angles en ordonnée sont en radians.*

coefficient intégral, nul dans ce cas) et il dispose de quatre entrées (deux erreurs et leurs dérivées, ce nombre monte à six si on prend en compte l'intégrale).

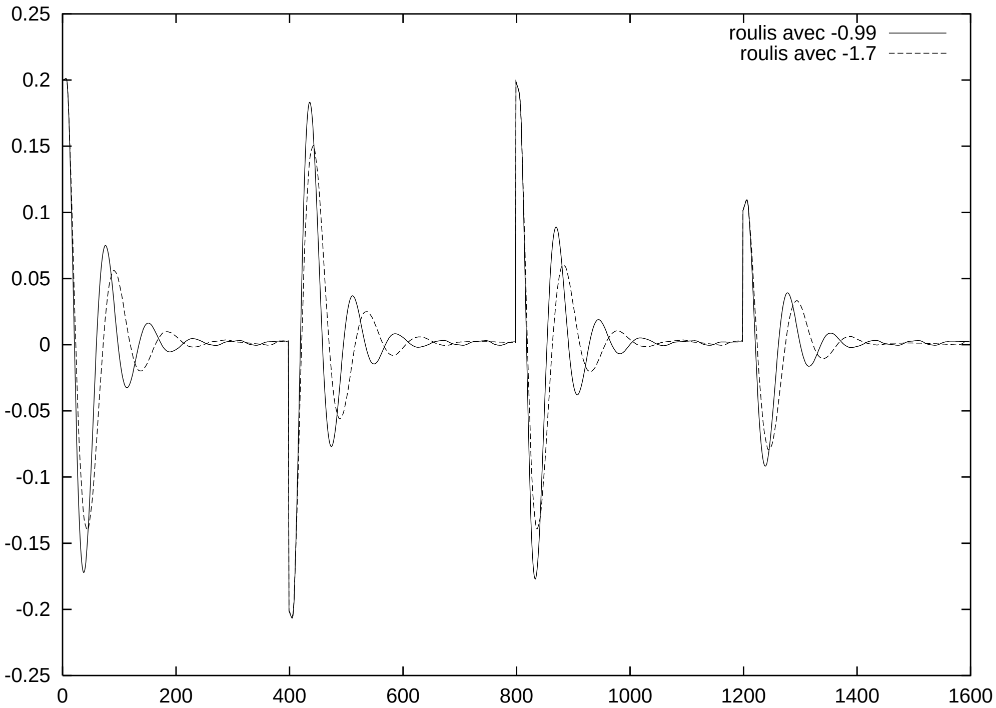


FIG. 9.6: *Comportement de l'hélicoptère en roulis (le comportement en tangage est inchangé) avec plusieurs valeurs de la connexion allant du neurone 2 vers le neurone 0.*

Lorsque l'on traite le problème par évolution de réseaux de neurones, on ne dispose plus que de deux entrées dont on ne connait pas a priori les liens avec les sorties. Ces liens doivent être découverts par le système. De plus, ils ne se limitent pas forcément à une connexion directe, mais peuvent être plus complexes afin d'exploiter plus finement la dynamique du système (c'est ce qui s'est produit dans le cas du contrôle de roulis).

9.2.2 Maintien d'assiette et de position

Dans la deuxième série d'expériences que nous avons lancée, l'objectif était de maintenir à la fois l'assiette et la position de l'hélicoptère. Nous avons fait ce choix, car ces quatre degrés

Génération	20	100	500	1000	2000
Moyenne	100,693	100,708	100,71	100,713	100,731
Médiane	100,701	100,715	100,716	100,717	100,72
Maximum	100,752	100,763	100,763	100,763	100,769
Minimum	100,61	100,641	100,65	100,65	100,716

FIG. 9.7: *Statistiques réalisées sur quatre expériences avec modules P (le poids de la connexion appartient à l'intervalle [-1 ;1]). Maintien de l'assiette et de la position.*

Génération	20	100	500	1000	2000
Moyenne	100,7188	100,7294	100,7402	100,753	100,772
Médiane	100,745	100,757	100,76	100,767	100,772
Maximum	100,766	100,77	100,772	100,775	100,784
Minimum	100,662	100,671	100,688	100,688	100,76

Fig. 9.8: *Statistiques réalisées sur quatre expériences avec modules P, I et D. Plusieurs versions de modules P sont disponibles : le poids de la connexion peut appartenir à l'intervalle [-1 ;1], [-10 ;10] ou encore [-100 ;100]. Maintien de l'assiette et de la position.*

de liberté font, a priori, intervenir les mêmes commandes, à savoir les commandes cycliques. Les commandes de collectif principal et de collectif de queue sont assurées par les PID fournis avec autopilot.

Les entrées sont les angles de roulis et de tangage, ainsi que la position relative de la cible (selon l'axe longitudinal et selon l'axe latéral).

Nous avons lancé deux séries d'expériences. La première série n'utilise qu'un seul module P dont le poids de l'unique connexion appartient à l'intervalle [-1 ;1]. La deuxième série contient des modules P, I et D. Au module P précédemment cité, nous avons ajouté deux autres modules P dont le poids de la connexion appartient à l'intervalle [-10 ;10] et, respectivement, [-100 ;100]. Le premier dispose d'une probabilité de choix dix fois inférieure au premier module P (dont le poids de l'unique connexion appartient à [-1 ;1]) et le second dispose d'une probabilité 100 fois inférieure. Ces connexions de poids potentiellement plus élevé peuvent permettre à neurone d'avoir une influence très importante sur un autre. Par exemple, cela peut permettre à un capteur d'avoir une influence prépondérante sur un neurone moteur. Nous avons cependant doté ces modules avec des connexions élevées d'une probabilité de choix faible de façon à éviter, autant que possible, que le réseau fonctionne uniquement en mode saturé et pour qu'il se maintienne dans la zone linéaire de fonctionnement des neurones, plus compatible avec les moteurs et servomoteurs utilisés sur un système réel (même si les calculs présentés ici n'ont été réalisés qu'en simulation).

Les modules D et I ont une probabilité de choix dix fois inférieure à celle du premier module P.

Les statistiques, présentées sur les figures 9.7 et 9.8, sont en faveur des expériences avec modules P, I et D. Cependant, si la différence est sensible, elle est insuffisante pour résoudre le problème complet. En effet, parmi toutes les expériences réalisées dans ces deux séries, aucune n'est parvenue à maintenir la position latérale.

9.2.3 Maintien complet

Malgré les difficultés précédemment rencontrées à contrôler les quatre degrés de liberté, nous avons tout de même essayé de contrôler l'hélicoptère complet - c'est-à-dire l'assiette, la position et l'altitude. Les entrées susceptibles d'être utilisées sont le roulis, le tangage, la position relative

Génération	20	100	500	1000	2000
Moyenne	100,630	100,654	100,676	100,677	100,688
Médiane	100,624	100,644	100,7	100,7	100,712
Max	100,681	100,702	100,723	100,723	100,725
Min	100,583	100,592	100,601	100,603	100,603

FIG. *9.9: Statistiques réalisées sur quatre expériences avec modules P : le poids de la connexion appartient à l'intervalle [-1 ;1]. Maintien complet.*

Génération	20	100	500	1000	2000
Moyenne	100,636	100,662	100,686	100,7	100,735
Médiane	100,638	100,65	100,685	100,701	100,728
Max	100,658	100,714	100,716	100,718	100,765
Min	100,605	100,637	100,66	100,679	100,712

FIG. *9.10: Statistiques réalisées sur quatre expériences avec modules P, I et D. Plusieurs versions de modules P sont disponibles : le poids de la connexion peut appartenir à l'intervalle [-1 ;1], [-10 ;10], [-100 ;100]. Maintien complet.*

de la cible (sur l'axe longitudinal et latéral), l'erreur sur l'altitude par rapport à une valeur de consigne ainsi que le lacet. Les sorties sont les quatre commandes de l'hélicoptère : les deux commandes cycliques ainsi que les deux commandes de collectif. La valeur maximale autorisée pour le roulis et le tangage est passée de $0,35rad$ à $0,7rad$, pour pénaliser moins brutalement les réseaux proposés par l'évolution. Nous avons imposé le calage du neutre des commandes de collectif (il est de 0,15 pour le collectif principal et de 0,09 pour le collectif de queue). Ce calage permet à l'hélicoptère d'être par défaut dans une configuration proche du vol stationnaire : le collectif principal compense approximativement le poids de l'appareil et le collectif de queue compense approximativement le couple créé par le rotor principal.

De même que pour les expériences précédentes, nous avons lancé deux séries de calculs. Les deux séries diffèrent uniquement par le réservoir de modules. Les réservoirs de modules sont identiques aux séries d'expériences de la section précédente.

Les statistiques sont favorables à la deuxième série d'expérience (figures 9.9 et 9.10) dont les performances, tant minimales, moyennes que maximales, sont nettement supérieures aux résultats de la première expérience.

Parmi toutes les expériences lancées, peu parviennent à contrôler tous les degrés de liberté (la moitié des expériences lancées) et la plupart de ces dernières génèrent un contrôle très peu efficace : maintien très oscillant de l'altitude ou contrôle très approximatif de la position.

Le comportement du meilleur réseau obtenu dans ces séries d'expériences est représenté sur la figure 9.11 (réseau généré par une expérience de la première série). Le maintien d'altitude est correct, mais avec une erreur statique de $1m20$. Le maintien de position est encore approximatif, mais il est stable : l'hélicoptère se met à tourner autour de la cible après une phase transitoire (figure 9.12). Le réseau correspondant ne dispose d'aucune récurrence (figure 9.13). La commande de lacet est quasiment inhibée, le poids reliant le capteur de lacet à la commande de collectif de queue étant très faible $(0,01)$. Les capteurs ont été reliés par évolution aux effecteurs correspondants au travers d'un ou deux modules P[75], la seule surprise étant la connexion reliant le neurone 10 au neurone 7, qui a donc tendance à faire incliner l'appareil sur le côté lorsque la cible est devant l'hélicoptère. L'influence de cette connexion sur la commande cyclique de roulis est cependant faible : les poids des deux connexions traversées avant d'atteindre le neurone 7 est strictement inférieur à 1 ($-0,35$ et $0,19$, le signal issus de dx est donc multiplié par $-0,06$ avant d'atteindre le neurone 7[76]).

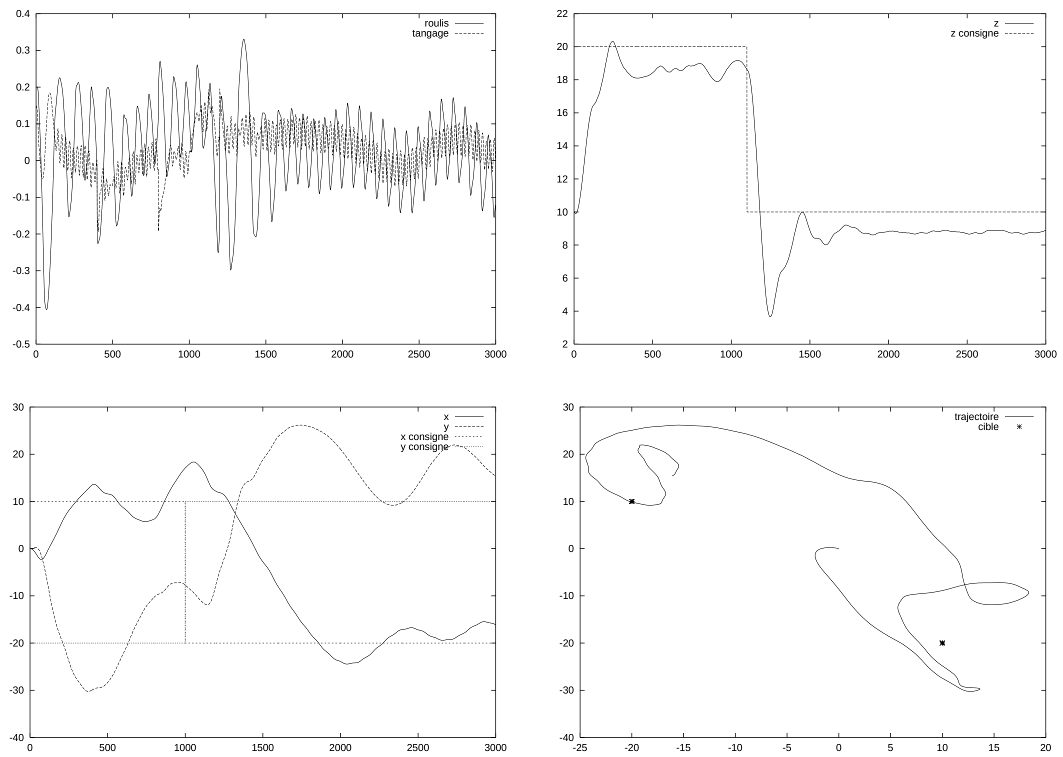


FIG. *9.11: Comportement d'un hélicoptère contrôlé par un réseau de neurones obtenu après 1100 générations (voir réseau figure 9.13). Les positions sont en mètres, les angles en radians.*

[75]le roulis et l'erreur de position latérale sont reliés à la commande cyclique de roulis, le tangage et l'erreur longitudinale de position sont reliés à la commande cyclique de tangage, le lacet est relié à la commande de collectif de queue et l'altitude est reliée à la commande de collectif principal.

[76]sans prendre en compte les non linéarités des fonctions d'activation

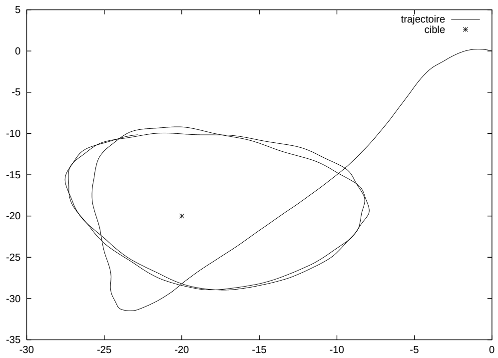


FIG. *9.12: Trajectoire de l'hélicoptère avec une cible fixe.*

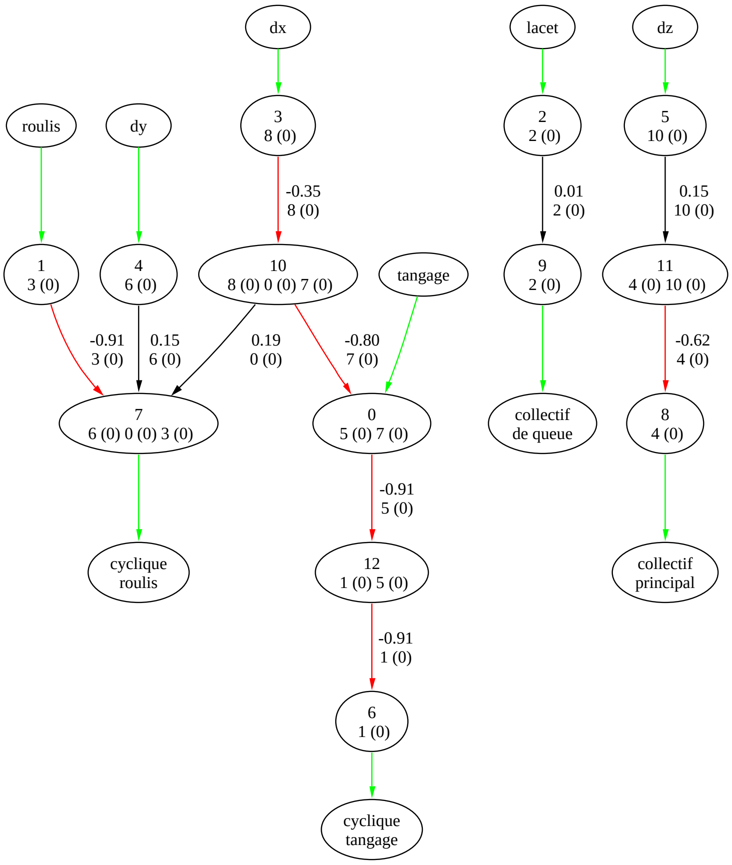


FIG. *9.13: Réseau de neurones obtenu après 1100 générations.*

9.2.4 Maintien complet avec "patron" de réseau

Les expériences précédentes ayant eu de grandes difficultés à générer des solutions pour le maintien de tous les degrés de liberté, nous avons préparé des expériences, dans lesquelles les réseaux étaient initialisés à partir d'un patron de réseau (figure 9.14), dont la structure a été inspirée des résultats des expériences précédentes.

Le patron correspond ainsi au réseau de la figure 9.13, auquel on a supprimé la connexion reliant la partie roulis de la commande cyclique à la partie tangage, qui ne ne correspondait pas à un contrôle intuitivement intéressant. Le contrôle de position (x et y) passe par un neurone intermédiaire de façon à donner la priorité au contrôle de roulis et de tangage (si un module P, dont la connexion est de poids inférieur à un, est choisie pour réaliser).

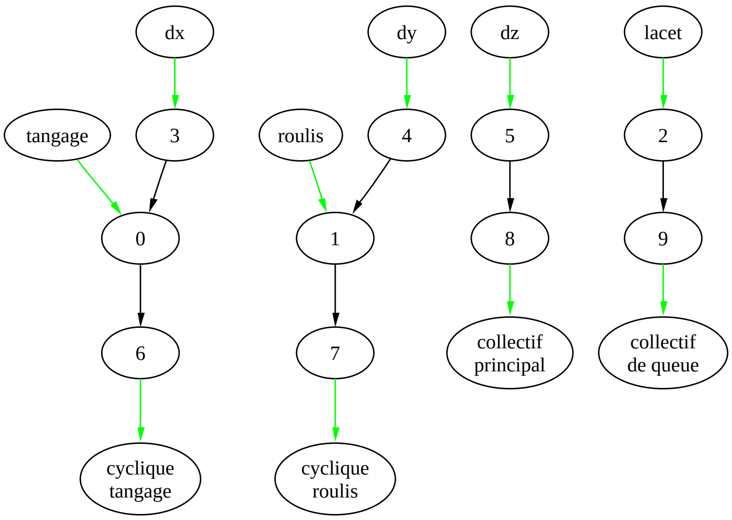


Fig. *9.14: Patron de réseau fourni à l'évolution. Les réseaux de la génération initiale sont obtenus en remplaçant chacune des connexions par un module P, I ou D. Dans les générations suivantes, la structure peut être modifiée par mutation.*

Les performances de ces expériences sont nettement supérieures à celles qui ont été réalisées sans patron (figures 9.15 et 9.16) : dès la vingtième génération, presque aucune expérience n'est en dessous de 100,7 alors que, sans le patron, très peu ont dépassé ce seuil avant la centième génération.

Les résultats sont similaires aux résultats portant sur le dirigeable complet avec patron : les expériences avec modules P uniquement sont plus performantes sur les premières générations, mais les expériences de la deuxième série, avec modules P, I et D, rattrapent ensuite leur retard.

Le comportement des meilleurs individus générés consiste à tourner autour de la cible (figure 9.17), l'altitude étant maintenue correctement avec, cependant, toujours une erreur statique de 50 cm. Le réseau correspondant (figure 9.18) a été directement déduit du patron, aucun module n'a été ajouté ou enlevé par mutation. Presque tous les modules utilisés sont des modules P avec une connexion comprise entre -1 et 1 ; un autre module P a, cependant, une connexion comprise entre -100 et 100. Un dernier module est tiré d'un module I mais, une des connexions étant nulle, ce module est inutile. On retrouve ainsi le même résultat que dans les expériences sans patron : le contrôle direct de lacet n'est pas utilisé[77], ce qui permet de faire tourner l'hélicoptère,

[77]La commande de collectif de queue reste donc à sa valeur de repos, calculée pour compenser approximativement

Génération	20	100
Moyenne	100,720	100,743
Max	100,726	100,76
Min	100,713	100,73

FIG. *9.15: Statistiques réalisées sur quatre expériences avec patron et modules P uniquement. Les résultats sont nettement supérieurs aux expériences sans patron. Les résultats sont arrêtés à 100 car les calculs étant longs (d'autant plus longs que, dans ces expériences, très peu d'individus ne parviennent pas à maintenir l'hélicoptère dans son domaine de vol, autrement dit peu d'expériences sont écourtées), nous disposons de peu d'expériences ayant dépassé ce seuil.*

Génération	20	100
Moyenne	100,701	100,746
Max	100,724	100,756
Min	100,671	100,734

FIG. *9.16: Statistiques réalisées sur dix expériences avec patron et modules P, I et D. Les résultats sont nettement supérieurs aux expériences sans patrons.*

comportement exploité pour le maintenir à une distance constante de la cible.

9.3 Effet du choix de la fonction de fitness

De même que pour le dirigeable, nous avons réalisé des expériences en utilisant une moyenne géométrique pour agréger les contributions des différents critères observés. L'expérience de base qui a servi à la comparaison est celle qui utilise un patron et un réservoir avec modules P, I et D dont certains signes sont imposés.

Les statistiques réalisées (figure 9.19) ne montrent pas un aplanissement des performances obtenues pour chacun des critères, comme c'était le cas pour le dirigeable, sauf à partir de la 500^{me} génération. Les performances moyennes ne sont pas non plus meilleures, contrairement au dirigeable, les différences restent très faibles.

9.4 Discussion

Les résultats obtenus sur l'hélicoptère sont similaires à ceux que nous avons trouvés sur le dirigeable lenticulaire. Lorsque nous restons limité à un faible nombre de degrés de liberté, l'évolution parvient sans mal à résoudre le problème. Le contrôle d'assiette de l'hélicoptère n'a ainsi posé aucun problème, toutes les expériences réalisées ont convergé très rapidement. Dès

le couple du rotor principal lorsqu'il se trouve en position "neutre", c'est-à-dire lorsque l'hélicoptère est en vol stationnaire.

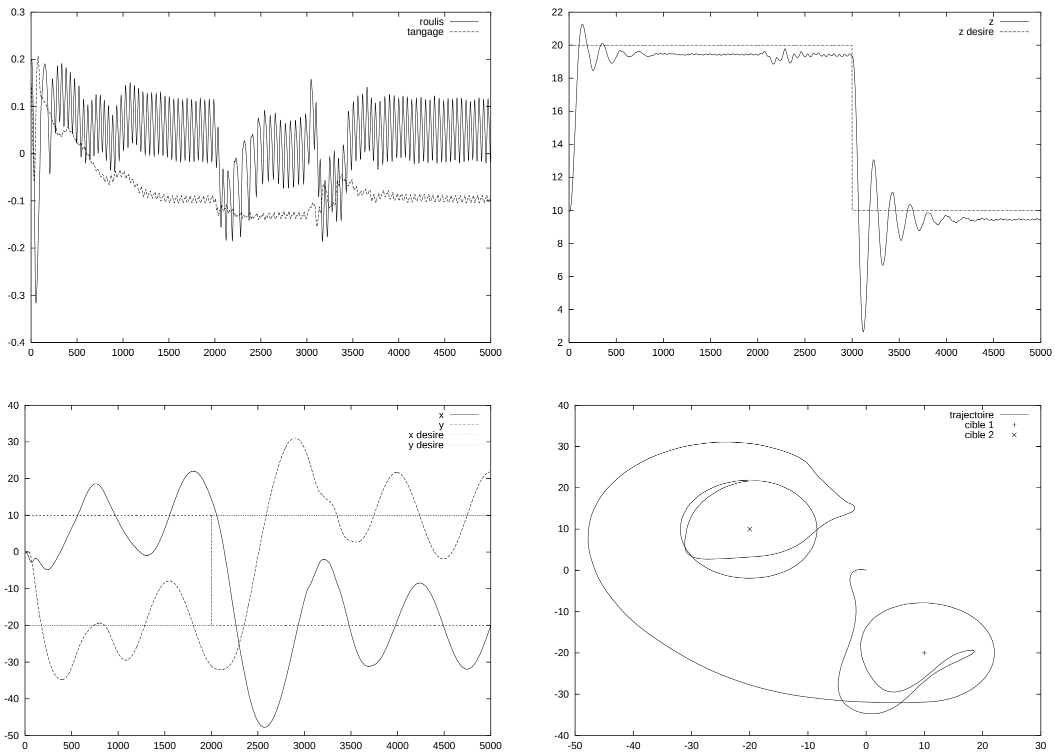


FIG. *9.17: Comportement d'un hélicoptère contrôlé par un réseau de neurones obtenu après 400 générations (voir réseau figure 9.18).*

que l'on augmente la complexité en cherchant à contrôler un plus grand nombre de variables, l'évolution reste bloquée sur des solutions intermédiaires, qui peinent à stabiliser convenablement tous les degrés de liberté. L'ajout de modules améliore les performances, mais pas suffisamment pour permettre à l'évolution de converger vers des solutions efficaces sur chacun des degrés de liberté.

Le problème intermédiaire, qui semblait une première étape vers le contrôle complet de l'hélicoptère, s'est révélé plus complexe à résoudre que le contrôle complet. En effet, aucun individu des expériences de maintien de position et d'assiette n'est parvenu à maintenir correctement la position alors que plusieurs individus des expériences de maintien d'assiette, altitude et position ont réussi à maintenir la position de l'appareil dans un voisinage proche de la cible. La stratégie utilisée par ces derniers requiert de ne pas avoir de contrôle précis de lacet, afin de laisser l'hélicoptère tourner sur lui-même. Associé à un contrôle adapté des autres degrés de liberté, cela a permis de faire tourner l'hélicoptère autour de sa cible à une distance réduite. Ce contrôle n'est pas optimal, mais il correspond à la meilleure stratégie que l'évolution nous a fournie.

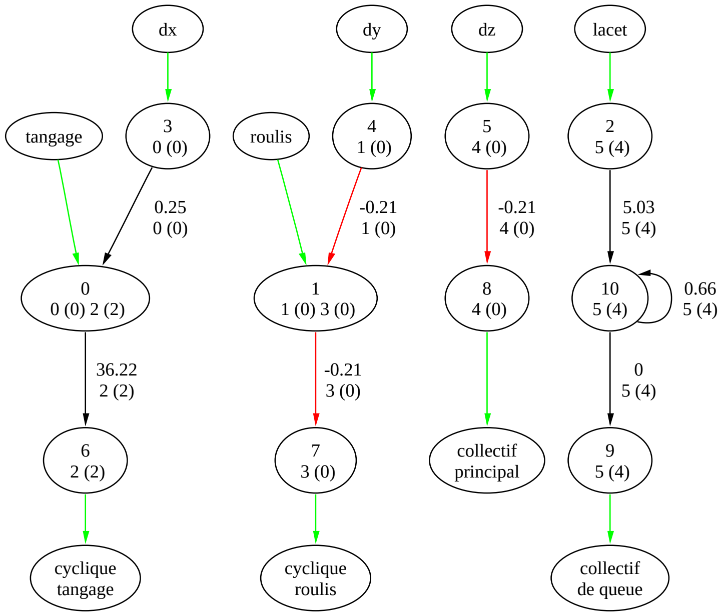


FIG. *9.18: Réseau de neurones obtenu après 400 générations.*

Génération	Donnée	Moy. arithmétique	Moy. géométrique
20	Maximum	0,993	0,990
	Moy. arithm.	0,699	0,681
	Moy. géom.	0,652	0,627
	Minimum	0,388	0,364
	Écart-type	0,279	0,288
100	Maximum	0,989	0,985
	Moy. arithm.	0,737	0,724
	Moy. géom.	0,706	0,691
	Minimum	0,486	0,484
	Écart-type	0,233	0,240
500	Maximum	0,991	0,979
	Moy. arithm.	0,752	0,750
	Moy. géom.	0,727	0,726
	Minimum	0,527	0,527
	Écart-type	0,218	0,209

FIG. *9.19: Statistiques des expériences lancées avec moyenne arithmétique ou géométrique pour agréger les différents critères. Les expériences portent sur le contrôle de roulis, tangage, altitude et position avec patron de réseau. Les différentes grandeurs présentées dans le tableau sont mesurées par rapport à l'ensemble des critères des meilleurs individus générés, ces critères étant l'erreur pour chacun des degrés de liberté. Les valeurs présentées sont les moyennes (arithmétiques) réalisées sur 10 expériences.*

Chapitre 10

Discussion

Dans cette thèse, nous nous sommes intéressé à la conception et à l'optimisation de contrôleurs neuronaux en utilisant des Algorithmes Évolutionnistes (AE). Les problèmes généralement résolus par ces techniques ne comprennent que peu de degrés de liberté, les réseaux n'ont que peu d'entrées et de sorties, et le nombre de neurones cachés reste faible. Notre objectif était d'étudier les performances de neurocontrôleurs conçus par AE pour le maintien de systèmes instables dont la complexité dépasse ce qui a été réalisé jusqu'à présent. À cette fin, nous avons développé ModNet, un codage modulaire que nous avons appliqué à des problèmes de difficulté croissante. Après le pendule inversé, qui est un banc de test couramment utilisé, nous nous sommes intéressé au contrôle d'un dirigeable lenticulaire, puis à celui d'un hélicoptère.

Le contrôle du pendule inversé a démontré que notre codage pouvait générer des réseaux de neurones susceptibles de contrôler et stabiliser un système intrinsèquement instable. L'évolution a été, en particulier, capable de découvrir des structures calculant une dérivée, caractéristique indispensable à la stabilisation du pendule. L'aspect modulaire de ModNet a permis de trouver facilement cette fonctionnalité, alors que, dans des expériences témoin sans modules[78], aucun des réseaux générés n'est parvenu à l'instancier.

Le contrôle du dirigeable ou de l'hélicoptère s'est avéré très efficace tant qu'il est resté limité à deux ou trois degrés de liberté. L'évolution a été capable de générer des réseaux de neurones plus efficaces que des contrôleurs conçus manuellement (maintien de l'assiette de l'hélicoptère) ou des réseaux capables d'exploiter des caractéristiques particulières à un type de plateforme (portance de l'enveloppe du dirigeable lenticulaire).

En revanche, l'évolution a eu énormement de difficultés à générer un réseau de neurones permettant de contrôler tous les degrés de liberté de l'hélicoptère dans les expériences où la génération aléatoire n'était pas guidée. Sur ce problème, l'aspect modulaire de ModNet n'a pas permis d'améliorer sensiblement les performances. Très peu d'expériences ont convergé, mais nous avons tout de même obtenu des solutions acceptables. Au contraire, sur le dirigeable, parmi toutes les expériences que nous avons réalisées sans préparation des réseaux, aucune n'est parvenue à

[78]En fait, ces expériences contenaient des modules "connexions", qui relient directement l'entrée à la sortie du module.

maintenir ses cinq degrés de liberté[79]. Au mieux, quatre degrés de liberté ont été contrôlés : le roulis, le tangage et la position horizontale.

Il est clair dès lors que notre travail souligne la difficulté à contrôler des systèmes présentant un nombre important de capteurs et d'effecteurs, parce que l'algorithme évolutionniste doit alors explorer un espace de recherche de taille prohibitive. Nous avons néanmoins réussi à résoudre ce problème, à condition de ne pas partir d'individus générés entièrement au hasard et d'utiliser des patrons de structure indiquant, a priori, quels capteurs devaient être reliés à quels effecteurs. Ces patrons ont été conçus manuellement, à partir de l'étude a priori du problème et de l'observation de solutions partielles générées au préalable par l'évolution. Ils ont orienté la génération aléatoire vers une zone de l'espace de recherche qui nous semblait pertinente. C'est grâce à ces informations que l'évolution a pu générer des contrôleurs efficaces sur le maintien complet du dirigeable comme de l'hélicoptère[80].

Dans la suite de ce chapitre, nous allons tout d'abord discuter les avantages et les limitations des réseaux de neurones. Nous nous intéresserons ensuite aux algorithmes évolutionnistes, avant de revenir sur l'utilisation des AE dans le but de concevoir des réseaux de neurones. Nous discuterons enfin l'intérêt de se consacrer à des applications plus complexes que celles qui sont actuellement étudiées en robotique évolutionniste.

10.1 Contrôleurs : réseaux de neurones

Le formalisme que nous avons utilisé pour les contrôleurs que nous avons générés est celui des réseaux de neurones. Leur avantage par rapport à d'autres types de contrôleurs est leur grande souplesse. En effet, tous les contrôleurs que nous avons présentés à la section 2.1 génèrent un comportement uniquement réactif et nécessitent des adaptations pour prendre en compte des aspects temporels (systèmes à base de règles, contrôleurs PID, programmation fonctionnelle). De plus, tous ne sont pas adaptés à la génération de sorties continues nécessaires au contrôle précis d'un robot[81] (systèmes de classeurs, automates à états finis). Enfin, certains de ces contrôleurs sont peu adaptés aux systèmes ayant plusieurs sorties (systèmes à base de règles floues et PID) parce qu'ils ne peuvent pas réaliser directement de commandes couplées entre plusieurs effecteurs et qu'ils doivent alors être inclus dans une architecture plus complète, conçue manuellement, pour regrouper les contrôleurs associés aux différents effecteurs.

Les réseaux de neurones n'ont pas ces inconvénients. Ils peuvent générer des comportements réactifs ou à dynamique complexe (prise en compte du passé, comportement chaotique). De plus, ils fonctionnent sur un nombre quelconque d'entrées et de sorties et peuvent générer des sorties binaires ou continues. Ils paraissent donc constituer un outil de choix en vue de contrôler des ro-

[79] Le roulis, le tangage et la position selon les trois axes

[80] Nous n'avons toutefois pas obtenu de contrôleur permettant de stabiliser l'hélicoptère au-dessus d'un objectif. Les meilleures solutions tournent autour de la cible à une distance fixe. Le maintien de roulis, de tangage et d'altitude est, par contre, efficace.

[81] Des stratégies de type "bang-bang" utilisant des commandes binaires peuvent être utilisées (commandes en tout ou rien), cependant, elles sont rarement compatibles avec les systèmes réels.

bots autonomes. Ainsi, en plus d'être dotés d'importantes capacités théoriques[82], leur utilisation avec des algorithmes supervisés a permis de résoudre des problèmes d'apprentissage complexes et variés - comme le contrôle d'un véhicule tout-terrain (Rivals, 1995) ou celui du NavLab, le véhicule sans pilote de l'université Carnegie Mellon (Pomerleau, 1991) - démontrant ainsi leurs capacités pratiques. Leur usage, lorsque l'on ne dispose pas d'exemples précis du comportement qu'ils doivent générer, est beaucoup plus délicat. Si l'apprentissage par renforcement a permis de générer des contrôleurs efficaces pour des problèmes complexes - comme le contrôle de la nage pour des systèmes ayant de nombreux degrés de liberté (Coulom, 2002) - il reste cependant généralement limité à des structures fixes, de type perceptron, dont la taille doit être déterminée empiriquement. En revanche, les algorithmes évolutionnistes peuvent être utilisés pour découvrir la structure ainsi que les paramètres d'un réseau de structure quelconque. Ils peuvent ainsi générer des stratégies de contrôle complexes avec des réseaux de structure simple, en exploitant des boucles de récurrences, comme nous l'avons vu à plusieurs reprises sur le contrôle du dirigeable ou de l'hélicoptère. De tels réseaux demandent peu de puissance de calcul et peuvent donc facilement être implémentés dans une électronique embarquée.

Quelles que soient les techniques de conception utilisées, les réseaux de neurones demeurent cependant restreints à ce jour à des applications dans lesquelles ils n'implémentent qu'un petit nombre de comportements différents :

> "A conclusion from numerous application studies, within a control systems environment, is that artificial neural networks can rarely be used in a black box, standalone mode, in that they are not generally powerful or versatile enough to deal with more than two or three tasks, often being restricted in operation to one specific role"
>
> (chapitre 1 de (Zalzala, 1996))

Ce problème constitue une limitation importante des réseaux de neurones. Afin de dépasser cette limitation, le réseau peut-être décomposé en modules fonctionnels regroupés dans une structure hiérarchique, par exemple, qui sélectionnera à chaque instant le sous-réseau à utiliser (Duro *et al.*, 2002). Cette solution pose néanmoins la question de la décomposition du problème, qui est délicate à réaliser dans certains cas. Ainsi, dans les expériences que nous avons effectuées sur le maintien de l'assiette et de l'altitude du dirigeable, un découpage trop rapide nous aurait conduit à générer des contrôleurs beaucoup moins efficaces que ceux que nous avons obtenus. De plus, ce type d'approche pose le problème de l'agrégation des contrôleurs élémentaires qui peuvent générer des comportements antagonistes. Des mécanismes de sélection de l'action appropriés doivent alors être mis en place (Girard *et al.*, 2002; Sigaud et Flacher, 2002).

La génération automatique de contrôleurs, qui tendrait à éviter ces problèmes liés au découpage fonctionnel et à l'agrégation des modules élémentaires, est un problème difficile à résoudre. L'évolution directe de contrôleurs ne parvient pas à converger vers des structures implémentant plusieurs comportements distincts. Ainsi, les architectures que nous avons obtenues pour

[82]Ce sont des approximateurs universels (Lapedes et Farber, 1987; Cybenko, 1988; Cybenko, 1989; Hornik, 1993)

le contrôle complet du dirigeable réalisaient un contrôle *simultané* de tous les degrés de liberté, alors qu'un contrôle *séquentiel* aurait pu générer des solutions plus efficaces[83]. Floréano et Urzelai ont observé les mêmes limitations lorsqu'ils ont fait évoluer des réseaux de neurones pour forcer un robot à réaliser deux tâches successives : allumer un interrupteur puis se diriger vers la lumière (Floreano et Urzelai, 2001). Les contrôleurs obtenus doivent réaliser successivement ces deux comportements antagonistes (l'interrupteur étant repéré par une marque noire). L'évolution de réseaux de neurones simples n'est pas parvenue à générer des réseaux réalisant les deux comportements. Les meilleurs réseaux obtenus faisaient tourner le robot en rond. Les caractéristiques de cette trajectoire étant finement ajustées par l'évolution, le robot finissait par atteindre l'interrupteur, puis, en continuant sa trajectoire sans qu'elle ait été en quoi que ce soit influencée ou modifiée par cet événement, le robot finissait par atteindre la lumière.

L'utilisation de modules tels que nous les avons définis n'a pas permis de dépasser ces limites, mais ce n'était pas l'objectif premier des travaux présentés ici et leur intérêt éventuel reste à étudier. En effet, un réseau contrôlant séquentiellement l'altitude, puis la position, ne dispose pas forcément d'une fitness nettement plus importante avec la fonction d'adaptation que nous avons utilisée car, si le maintien d'altitude ainsi que la consommation énergétique sont meilleurs, le maintien de position sera moins efficace, puisqu'il sera inhibé dans un premier temps si la stratégie de contrôle est séquentielle. La différence de fitness entre un maintien simultané et séquentiel n'est ainsi peut-être pas suffisante pour clairement favoriser les solutions séquentielles. De plus, l'utilisation de neurones et de connexions classiques[84], ainsi que nous l'avons fait ici, ne permet pas facilement de générer les inhibitions indispensables, a priori, à la réalisation d'un contrôleur séquentiel. Il nous faudra ajouter de nouveaux types de neurones ou de connexions afin de favoriser l'apparition de tels comportements[85]. Floreano et Urzelai (2001) ont ainsi résolu leur problème en enrichissant le formalisme qu'ils ont utilisé. Au lieu de considérer des connexions simples, ils ont utilisé des connexions adaptatives dont les poids étaient susceptible de varier en suivant des lois de type Hebbiennes[86]. Cette solution génère cependant des contrôleurs instables, dont les poids varient constamment, et elle ne nous a pas parue adaptée aux problèmes que nous avons traités. Elle suggère cependant qu'en ajoutant des mécanismes supplémentaires au formalisme classique, le problème de la génération de plusieurs comportements avec un seul réseau peut être résolu. Le couplage de ces capacités avec des algorithmes évolutionnistes, en vue de générer des réseaux stables et robustes, reste cependant à étudier.

[83]Contrôle de l'altitude tout d'abord - en exploitant la portance de l'appareil - puis, lorsque l'altitude est correcte, contrôle de la position.

[84]Neurones à fonction de transfert de type sigmoïde et connexions à poids constants pendant une évaluation. L'activation des neurones étant la somme pondérée des entrées.

[85]Des connexions sigma-pi, par exemple (Mel, 1990).

[86]Une connexion est modifiée en fonction de l'activité pré- et post-synaptique (Hebb, 1949; Stent, 1973; Kelso *et al.*, 1986; Singer, 1987; Stanton et Sejnowski, 1989).

10.2 Les Algorithmes Évolutionnistes

10.2.1 Les avantages des AE

Les AE sont des algorithmes non supervisés. Ils ne nécessitent donc pas de connaître une solution idéale : il suffit de disposer d'un retour ponctuel permettant de noter un comportement dans son ensemble. Ils n'ont pas besoin de modèles dont ils imiteraient le comportement et peuvent ainsi générer des solutions originales par rapport à ce qu'un expert humain aurait pu faire.

C'est ce que nous avons constaté sur les expériences de contrôle d'assiette et d'altitude du dirigeable, ainsi que sur le contrôle complet de l'hélicoptère. Ces expériences ont en effet exploité des couplages non linéaires, là où une approche directe et rapide nous aurait conduit à réaliser un contrôle indépendant des différents degrés de liberté. Cette caractéristique est le principal avantage des systèmes dotés de telles capacités d'adaptation : ils sont capables de trouver par eux-mêmes des stratégies efficaces compte tenu du contexte dans lequel ils se trouvent. Ils ne font pas d'hypothèses simplificatrices ; ils prennent leur environnement comme il est, sans chercher à le simplifier. Cette caractéristique permet aux AE de découvrir des solutions simples et efficaces. Ainsi, tous les réseaux de neurones que nous avons obtenus contiennent peu de neurones cachés et pourront donc facilement être implémentés dans l'électronique embarquée lors des essais sur systèmes réels.

De plus, l'évolution peut aider à l'étude de systèmes dont la dynamique est mal connue. Si l'on peut disposer d'un banc de test réel, l'évolution peut permettre d'aider à la compréhension du système par analyse a posteriori des résultats générés. Les solutions générées peuvent également indiquer des stratégies de contrôle efficaces, ce qui est intéressant lorsque l'on ne connait pas d'autres solutions. L'évolution a déjà été utilisée pour générer des contrôleurs pour robots à ailes battantes (Augustsson *et al.*, 2002) que l'on commence à savoir modéliser, mais dont on ne sait pas encore générer de contrôleurs. Elle pourrait, de même, être utilisée pour étudier le contrôle d'appareils volants peu encore étudiés, comme les “lifters”[87] (Naudin, 2002), pour lesquels on ne dispose pas de modèles physiques[88].

10.2.2 Limitations des AE en tant qu'algorithme d'optimisation de contrôleurs

Les AE convergent vers des solutions maximisant la fonction d'adaptation, ce qui ne conduit pas toujours au comportement souhaité, spécialement lorsque l'espace de recherche est vaste. L'opportunisme des AE les conduit en effet à exploiter tout ce qui leur permet d'obtenir une bonne fitness. Si cette caractéristique permet souvent de générer des comportements efficaces et originaux, elle peut aussi correspondre à l'exploitation de failles ou de caractéristiques fragiles du système. Les bugs de simulation peuvent être exploités, par exemple, mais les systèmes réels

[87] Appareils volants fonctionnant sur des principes électrocinétiques

[88] Une plateforme réelle devrait alors être utilisée pour évaluer les solutions générées par évolution.

ne sont sont pas exempts non plus de ce type d'effet de bord. Le contrôleur neuronal obtenu par Floreano et Urzelai (2001) était adapté à la taille de l'arène pour générer une trajectoire résolvant en apparence le problème mais, en pratique, il se contentait de suivre en aveugle une trajectoire apprise par le réseau de neurones. En d'autres termes, le contrôleur correspondant était "adapté" à l'arène utilisée, mais pas adaptable à une arène différente. L'apparition de tels comportements pathologiques peut être réduite en utilisant un certain nombre de principes simples (Jakobi, 1997). Floreano et Urzelai ont obtenu des contrôleurs efficaces n'exhibant pas ce type de comportement en utilisant des synapses adaptatives. En effet, les poids du réseau étant initialisés aléatoirement[89], le comportement consistant à tourner en rond en suivant une trajectoire exacte n'était alors plus possible, car il nécessite un ajustement précis des poids de connexions que l'initialisation aléatoire rendait impossible.

Cependant, ces techniques ne suffisent pas toujours. La fonction de fitness doit également être conçue avec soin. Elle constitue le seul retour utilisé par l'évolution et doit donc favoriser toutes les solutions proches du comportement souhaité en pénalisant toutes celles qui en sont éloignées.

De même, les conditions dans lesquelles sera évaluée la fitness jouent un rôle primordial. Elles doivent contenir un échantillon représentatif du domaine de fonctionnement dans lequel le système doit fonctionner. Si elles sont réalisées dans trop peu de conditions différentes, l'évolution va générer des solutions spécialisées, qui seront très performantes dans les conditions de test, mais dont les capacités de généralisation seront très réduites.

La définition de l'évaluation d'un individu - que ce soit le choix de la fonction de fitness ou celui des conditions d'évaluation - constitue le talon d'achille des méthodes évolutionnistes. Sur cette étape reposeront entièrement les performances des contrôleurs générés. Quelques principes généraux simples permettent d'éviter les cas pathologiques. Dans les expériences que nous avons réalisées, par exemple, la notation d'un individu reposait sur l'évaluation de ses performances dans quatre conditions différentes, ce qui nous a permis d'obtenir des réseaux dont le fonctionnement restait toujours aussi efficace dans les différentes conditions dans lesquelles nous les avons ensuite testés. De plus, nous avons évalué nos solutions pendant une durée suffisante, estimée empiriquement, pour que les contrôleurs générés soient stables.

10.3 Combinaison Algorithmes Évolutionnistes / Réseaux de Neurones

Les AE offrent l'avantage d'être des méthodes d'apprentissage non supervisé fonctionnant sur tous types de réseaux de neurones. En effet, du fait de leur généricité, ils peuvent tout aussi bien optimiser des réseaux à couches que des réseaux récurrents.

De plus, déterminer la structure d'un réseau de neurones est un problème ouvert que peu

[89] Ces poids étaient ensuite modifiés en permanence en utilisant les lois choisies par évolution pour chacune des connexions.

d'algorithmes abordent, même en apprentissage supervisé. Certains algorithmes existent cependant dans ce contexte pour déterminer le nombre de neurones à utiliser dans la couche cachée d'un perceptron(Fahlman et Lebiere, 1991; Rivals et Personnaz, 2000). Les AE peuvent traiter cette problématique et générer des structures quelconques tout en optimisant leurs poids simultanément.

10.3.1 Le codage

L'espace de recherche est l'ensemble des réseaux susceptibles d'être générés par évolution. Tous les codages ne permettent pas de générer n'importe quel type de structure. Certains se concentrent sur la génération de réseaux à couches, alors que d'autres sont susceptibles de générer des structures plus générales. Se limiter à des réseaux à couches permet de réduire considérablement l'espace de recherche et de nombreux problèmes peuvent être résolus à l'aide de tels réseaux. Cependant, ces structures ne peuvent générer qu'un comportement réactif, une réponse toujours identique à une même stimulation. Des réseaux récurrents peuvent prendre en compte les perceptions précédentes ou générer des comportements dynamiques indépendants des entrées, comme des CPG[90], par exemple.

À plusieurs reprises, les systèmes que nous avons contrôlés ne devaient pas nécessiter, a priori, de récurrences (contrôle de l'assiette de l'hélicoptère, contrôle de l'assiette et de l'altitude du dirigeable) puisque nous disposions de contrôleurs PID permettant de générer des solutions efficaces. Des réseaux à couches auraient donc du suffire[91]. Cependant l'évolution s'est servie à plusieurs occasions de sa capacité à générer des structures récurrentes pour concevoir des contrôleurs qui se sont révélés plus efficaces que les contrôleurs classiques dont nous disposions (contrôle d'assiette de l'hélicoptère ou contrôle d'assiette et d'altitude du dirigeable) et qui ont utilisé des stratégies de contrôle différentes de celles auxquelles on aurait pu penser (comme la stratégie de contrôle oscillante de type "bang-bang" pour la stabilisation du dirigeable). Ainsi, même pour des problèmes dont on connaît la forme de la solution, il est intéressant de ne pas contraindre l'évolution si ce n'est pas nécessaire, car des solutions originales peuvent être générées.

L'espace de recherche est défini par l'ensemble des structures susceptibles d'être engendrées par un codage donné, ainsi que par les opérateurs génétiques qui vont permettre de l'explorer. En ajoutant à cet espace la fonction d'adaptation qui va associer une note à chaque individu généré, on obtient le "paysage" de fitness dont dépendent pour beaucoup les performances de l'AE. Ce paysage doit avoir un certain nombre de propriétés pour que l'AE puisse converger, comme nous l'avons vu dans le chapitre 3 : il ne doit pas être de taille trop importante et doit surtout être aussi lisse que possible. En effet, si les solutions, qui représentent des extrema du paysage de fitness, sont sur des "pics", toute solution, même très proche, aura une mauvaise fitness et sera donc rejetée par l'AE - qui cessera ainsi ses recherches dans cette zone pourtant intéressante. Les solutions qui seront découvertes par l'évolution ne seront donc pas les extrema

[90]Central Pattern Generator : générateur de rythme utilisé pour la marche ou la nage (Gallistel, 1980).

[91]Avec éventuellement une connexion récurrente d'un neurone sur lui-même pour implémenter un calcul d'intégrale.

globaux du paysage de fitness, mais plutôt les extrema situés sur les pics les plus lisses, qui sont les plus faciles à découvrir. C'est le phénomène que nous avons observé sur les expériences portant sur le pendule inversé : compte tenu de notre codage et de notre fonction de fitness, les solutions les plus faciles à découvrir dans les expériences témoin[92] étaient des solutions avec une boucle de récurrence contenant plusieurs neurones. Alors que ces solutions sont nettement moins performantes que celles qui ont été ensuite découvertes, c'est vers ce type de solution que s'est orientée l'évolution. La raison en est que ces structures peuvent être découvertes progressivement à partir de structures ayant déjà un avantage sélectif par rapport aux autres. Les mutations que nous avons définies peuvent insérer des connexions, aussi ces boucles ont-elles pu être construites par adjonctions progressives. Les solutions découvertes dans les séries d'expériences suivantes sont complètement différentes. Elles ne sont pas plus complexes et auraient tout à fait pu être générées par une expérience de la première série. Cependant, ces structures étaient moins susceptibles d'être découvertes progressivement dans ces expériences témoin et étaient donc plus difficiles à obtenir. Dans les expériences des séries suivantes, le paysage de fitness avait changé, bien que la fonction de fitness reste la même : le fait de donner la possibilité d'utiliser des modules a modifié la structure du paysage et l'a en quelque sorte lissé au voisinage des solutions optimales.

Ainsi, même sans modifier la fonction d'adaptation, nous avons changé la structure du paysage de fitness, simplement en modifiant le codage génétique manipulé par l'évolution. Ces expériences ont montré l'intérêt de disposer d'un croisement efficace, échangeant des blocs fonctionnels entre individus, et de protéger des mutations structurelles des sous-parties du réseau.

10.3.2 Problèmes multi-critères ou multi-tâches

La conception de réseaux de neurones pour accomplir des tâches multiples, ou pour satisfaire simultanément plusieurs critères, est un problème ouvert qui n'est pas lié à l'utilisation des AE, comme nous l'avons vu précédemment. Dans le contexte des AE, de même que pour les autres approches, faute de mieux, les techniques couramment utilisées sont celles de la décomposition fonctionnelle et de la résolution incrémentale du problème (Urzelai *et al.*, 1998). Ces techniques s'adaptent bien aux AE qui peuvent optimiser séparément des contrôleurs avant de concevoir l'interface les regroupant (Lara *et al.*, 2001). Ils peuvent également optimiser un contrôleur de complexité croissante intégrant progressivement toutes les fonctionnalités souhaitées (Kodjabachian et Meyer, 1997; Pasemann *et al.*, 2000).

Ces techniques reposent néanmoins sur la qualité du découpage du problème, laquelle nécessite une analyse approfondie. Tous les problèmes ne se prêtent pas aisément à ce type de stratégie. Ainsi dans les expériences que nous avons présentées dans cette thèse, plusieurs ont montré les limites de ce type d'approche. Le contrôle d'altitude et d'assiette du dirigeable aurait pu être obtenu en traitant les deux problèmes de manière indépendante. Cependant, dans ce cas, la solution trouvée, contrôlant séparément chacun des degrés de liberté, aurait été beaucoup moins efficace, comme nous l'avons montré au chapitre 8. Le contrôle d'hélicoptère partiel s'est

[92]Expériences n'exploitant pas les aspects modulaires de ModNet

également révélé être un problème plus délicat que le contrôle complet, la stratégie proposée à l'évolution pour maintenir la position de l'appareil (exploitation des commandes cycliques) se révélant plus complexe à mettre en œuvre du point de vue de l'évolution qu'un maintien complet dans lequel le contrôleur a exploité le lacet pour permettre à l'appareil de tourner autour de la cible.

La fonction de fitness utilisée permet de comparer les individus et de les classer afin d'effectuer la sélection qui constitue la base d'un AE. Les algorithmes de sélection les plus utilisés, dont font partie ceux que nous avons utilisés dans cette thèse, nécessitent une note unique pour chaque individu, ce qui implique de ramener un ensemble de critères à une seule valeur. La fonction d'agrégation que nous avons utilisée dans cet objectif est la moyenne arithmétique. Ce choix n'est cependant pas sans conséquences sur les performances des solutions obtenues. Nous avons vu qu'en utilisant une moyenne géométrique pour le dirigeable nous obtenions des résultats plus réguliers, avec un écart-type inter-critères plus faible, même si les meilleurs contrôleurs n'étaient pas au final plus performants. Même si la différence sur l'hélicoptère était cependant moins nette, le choix de cette fonction dispose d'une influence non négligeable sur les résultats obtenus.

10.3.3 Nombre important de capteurs et d'effecteurs

Lorsque le nombre de capteurs et d'effecteurs à utiliser est important, l'évolution de réseaux de neurones a, jusqu'à présent, vite montré ses limites. La solution généralement employée consiste à diviser le contrôleur en réseaux de neurones plus simples. Les contrôleurs disposant de symétries évidentes peuvent ainsi être découpés facilement en contrôleurs élémentaires, le problème étant, là encore, de parvenir à assembler convenablement les différents éléments.

Si le contrôleur ne dispose pas de symétries évidentes, il peut éventuellement se découper en contrôleurs élémentaires indépendants utilisant chacun des capteurs et effecteurs différents, ce qui nous ramène aux techniques évoquées dans le paragraphe précédent.

10.3.4 Solutions proposées dans cette thèse

Les solutions que nous avons proposées dans cette thèse ont pour objectif d'éviter la décomposition fonctionnelle de la tâche à résoudre, ainsi que le découpage du réseau de neurones.

Apport

La première technique consiste à donner à l'évolution des éléments structurels plus riches que le simple neurone ou la simple connexion. L'évolution ne devra alors plus assembler des unités computationnelles élémentaires, mais pourra manipuler des structures plus complexes, implémentant des comportements ou des fonctionnalités élémentaires pertinentes. Ces structures seront protégées au sein de divers modules et seront donc moins susceptibles de disparaître par mutation structurelle. De plus, si elles se révèlent efficaces, elles pourront être échangées rapidement avec les autres individus de la population grâce au croisement.

Les modules permettent aussi de capitaliser sur des connaissances acquises, soit lors de précédentes expériences, soit au moyen d'études préliminaires et de connaissances théoriques sur le système. Cette approche a clairement montré ses avantages sur le problème du pendule inversé.

La deuxième méthode proposée consiste à fournir un patron de réseau à l'évolution afin de limiter la combinatoire liée à un grand nombre d'entrées et de sorties. Le patron indique une structure reliant les capteurs et les effecteurs au travers de modules que l'AE devra ensuite choisir et paramétrer. Cette méthode a permis d'obtenir des correcteurs capables de contrôler entièrement un dirigeable ou un hélicoptère sans avoir, ni à réaliser de décomposition fonctionnelle du problème, ni à découper le réseau en plusieurs morceaux indépendants. Le patron n'étant utilisé que lors de la génération aléatoire, des modifications de la structure suggérée peuvent se produire par mutation ainsi que cela s'est produit pour le contrôle du dirigeable.

Limites

Sur le contrôle du dirigeable et de l'hélicoptère sans utiliser de patron, le gain qu'a apporté l'utilisation de modules est moins marqué que sur les expériences portant sur le pendule inversé. Cette approche a amélioré les performances moyennes des différentes expériences, mais n'a pas eu d'effet significatif sur le meilleur individu généré dans chaque série. L'effet des modules était, par contre, beaucoup plus net sur les expériences avec patron où, après un démarrage difficile, les expériences utilisant des modules P, I et D parviennent à de meilleurs résultats.

L'utilisation de modules contenant un plus grand nombre de paramètres (modules I et D) pourrait être un handicap face aux modules les plus simples ne contenant qu'un seul paramètre (modules P). Ce phénomène serait d'autant plus marqué que l'on aurait affaire à des problèmes mettant en jeu un grand nombre d'entrées et de sorties. Les expériences portant sur le pendule inversé, qui ont le mieux illustré l'intérêt des modules, utilisent un réseau n'ayant que deux entrées et une sortie. Les expériences portant sur le maintien de l'assiette et de l'altitude du dirigeable, ainsi que sur le maintien complet de l'hélicoptère avec patron, semblent également conforter cette hypothèse : alors que les expériences avec modules P, I et D avec signes libres ne semblent pas avantagées par rapport aux expériences témoin n'utilisant qu'un module P, les expériences avec modules P, I et D dont certains signes sont imposés donnent de bien meilleurs résultats. De même, dans les expériences avec patron, celles qui utilisent des modules P, I et D finissent par avoir de meilleures fitness après un démarrage difficile.

Trouver le bon patron utilisable est un point délicat de l'approche que nous avons utilisée dans cette thèse. Le résultat d'expériences précédentes peut être exploité à cette fin : le réseau obtenu pour le maintien complet de l'hélicoptère, dont la structure était en fait très proche du patron que nous avons utilisé, aurait ainsi pu nous servir de modèle. De manière générale, lors du passage d'une expérience simple à une expérience plus complexe, nous aurions pu utiliser les solutions issues de l'expérience simple comme des patrons pour la résolution du problème plus complexe ainsi que le suggèrent Pasemann et al. (2000). Cependant, sur le problème du contrôle du dirigeable, cette technique n'aurait pas donné de résultats intéressants, car la stratégie utilisée pour le maintien de l'altitude et de la position ne peut fonctionner que si le dirigeable peut se mettre face au vent, ce qui n'est possible que si les moteurs horizontaux sont éteints. Or les contrôleurs n'avaient pas les moyens d'inhiber le fonctionnement de ces moteurs. Cette partie nécessite donc toujours une analyse du fonctionnement du système. Elle peut être guidée par les résultats de précédentes expériences, mais ne peut se faire de manière autonome.

10.4 Le contrôle de robots volants : le nouveau défi des systèmes adaptatifs

Les robots volants constituent des plateformes dont le fonctionnement est beaucoup plus riche que celui des robots à roues. Ils disposent de nombreux degrés de liberté dont certains peuvent être couplés entre eux, comme nous l'avons constaté lors de nos expériences. Dans ce contexte, ces robots génèrent de nouvelles problématiques permettant de tester les capacités de passage à l'échelle des algorithmes adaptatifs.

Le type de contrôle requis est similaire à celui des robots à roues, mais il nécessite une plus grande robustesse afin de contrer l'effet des perturbations atmosphériques ou de prendre en compte l'inertie de ces appareils. De plus, ces plateformes n'étant pas en équilibre statique (sauf en ce qui concerne le dirigeable), elles nécessitent un contrôle constant efficace et rapide.

Leur principal défaut est la difficulté de mise en œuvre de plateformes réelles, ce qui implique une logistique plus lourde que pour des robots à roues. Les risques de destruction sont également beaucoup plus importants que pour des robots à roues, ce qui rend les algorithmes fonctionnant directement sur le système réel difficilement utilisables, sauf s'ils ont été au préalable entraînés en simulation.

Ces robots permettent donc de tester les capacités de passage à l'échelle des algorithmes de contrôle. En effet, si le pendule inversé permet d'évaluer les performances au niveau de la qualité du contrôle, il n'implique qu'un espace de recherche réduit. Les réseaux générés n'ont que deux à quatre entrées, et surtout qu'une seule sortie. Même s'il est intéressant de tester des algorithmes sur le pendule inversé, nous avons pu constater que les caractéristiques observées lors du contrôle de ce système ne se transposent pas forcément lors de la résolution d'un problème de difficulté supérieure. C'est pourquoi, il nous semble indispensable désormais de ne plus se limiter à des problèmes de contrôle relativement simples, mais d'étudier des systèmes de plus grande complexité afin de s'assurer des capacités de passage à l'échelle des algorithmes testés. En effet, le contrôle du pendule, par exemple, ne constitue pas une fin en soi et les recherches devraient se concentrer sur le véritable problème encore non résolu de l'utilisation d'AE pour la conception de réseaux de neurones : celui du passage à l'échelle. Dans ce contexte, les robots volants fournissent de très bonnes plateformes de test et constituent les nouveaux défis que les algorithmes adaptatifs doivent relever. Les résultats présentés dans cette thèse montrent que l'évolution peut contrôler ce type d'appareils sans avoir besoin de décomposer ni le problème, ni l'architecture de contrôle. Elle a exploité pour ce faire des patrons neuronaux que nous lui avons fournis et qu'elle a ensuite automatiquement inclus dans des contrôleurs neuronaux appropriés. Il reste à déterminer dans quelle mesure de tels patrons pourraient, eux aussi, être générés de façon automatique.

Chapitre 11

Perspectives

11.1 Étude des modules

11.1.1 Modules et comportements

Les modules que nous avons conçus et utilisés dans le cadre de ce travail correspondent à des structures fonctionnelles élémentaires au sein d'un réseau de neurones. Ils réalisent des calculs intermédiaires et ne sont pas forcément liés à un comportement extérieur particulier. Un seul et même module peut donc être utilisé dans l'expression de plusieurs comportements et un comportement particulier peut mettre en œuvre plusieurs modules.

Il se trouve que la résolution de problèmes multi-tâches peut nécessiter l'implémentation de plusieurs comportements différents observables d'un point de vue extérieur : prendre un objet, le poser, suivre la lumière... Or l'approche couramment utilisée dans ce type de problème consiste à le décomposer en extrayant des comportements élémentaires que l'on cherche ensuite à résoudre séparément. Dès que les différents comportements ne sont plus indépendants, il faut choisir quels comportements activer en fonction des conditions, ce qui n'est pas un problème simple à résoudre (Tyrrell, 1993).

L'utilisation de ModNet sur de tels problèmes peut nous permettre de mieux comprendre les mécanismes à l'œuvre lors de la génération automatique de modules et nous fournir des pistes pour concevoir des systèmes dotés de capacités accrues d'auto-organisation sur des problèmes complexes. En effet, en utilisant ModNet, la décomposition et l'agrégation des contrôleurs élémentaires seraient réalisées simultanément, ce qui garantirait la compatibilité des comportements obtenus les uns avec les autres et éviterait l'étape délicate de la conception d'un système de sélection.

L'objectif serait de disposer d'un système capable de générer automatiquement des modules résolvant une tâche élémentaire dans le cadre du problème global à résoudre. Ces modules pourraient soit être utiles pendant tout le fonctionnement du robot (un module CPG pour un robot marcheur ou un module D pour un robot volant), soit n'être utiles que dans certaines conditions (module de marche arrière ou de saisie d'un objet pour un robot terrestre, module de décollage ou d'atterissage pour un robot volant). Le but serait le même que celui qui est adopté lors de la résolution incrémentale d'un problème : rendre les comportements plus robustes, plus faciles à analyser, permettre l'inclusion de connaissances.

11.1.2 Composition des modules

Les systèmes modulaires implémentés manuellement reposent généralement sur la capacité à activer un module au détriment des autres. C'est le cas, par exemple, du mécanisme de type "winner-takes-all" : seul le module dont la "motivation" ou l'activation est la plus importante est actif, tous les autres sont inhibés (Atkinson et Birch, 1970), mais c'est aussi le cas de mécanismes plus réalistes du point de vue biomimétique (Girard *et al.*, 2002).

Cette technique n'est pas aisément implémentable avec le formalisme classique des réseaux de neurones. C'est pourquoi nous pouvons envisager dans le futur l'ajout de nouveaux types de neurones ou de connexions - que ce soient des connexions sigma-pi (Mel, 1990) ou des synapses adaptatives (Willshaw et Dayan, 1990) - qui pourront inhiber temporairement le fonctionnement d'un ou plusieurs neurones de façon à invalider un module.

Le fait qu'un module puisse en inhiber un autre, en inhibant les neurones et/ou connexions d'un autre module, devrait faciliter l'émergence de modules correspondant à des comportements susceptibles d'être déclenchés ou interrompus à tout moment. En effet, les modules que nous avons utilisés dans ce travail sont constamment actifs et ont donc naturellement tendance à correspondre à des fonctionnalités nécessaires pendant toute la durée de vie d'un individu. Il serait intéressant d'étudier et de comparer les modules générés avec et sans mécanisme d'inhibition pour la résolution d'une tâche requérant l'implémentation de plusieurs comportements actifs séquentiellement.

11.1.3 Génération automatique de modules

L'objectif de notre approche est de parvenir à générer automatiquement des modules fonctionnellement significatifs, autrement dit des modules dont le fonctionnement et l'importance sont clairement identifiables[93] et qui sont, autant que possible, indépendants les uns des autres. En effet, pour que l'échange de fonctionnalités par croisement soit possible entre réseaux, chaque fonctionnalité doit être implémentée dans un module indépendant. Si une fonctionnalité est obtenue par assemblage de plusieurs modules, elle ne pourra pas être transmise par croisement.

De plus, si un module implémente une fonctionnalité particulière, il pourra facilement être réutilisé dans d'autres expériences de difficultés croissantes. C'est ce que nous avons observé avec la génération automatique de modules 'D' dans les expériences sur le pendule inversé, modules qui ont pu être réutilisés pour d'autres problèmes plus complexes, comme le contrôle de dirigeable ou d'hélicoptère.

[93]Identifiables du point de vue du robot, comme le sont les modules 'D' découverts pendant les expériences sur le pendule. Ces modules ne correspondent pas forcément à un comportement apparent du point de vue d'un observateur extérieur.

Ce type de mécanisme doit être un processus émergent de l'interaction entre le codage modulaire et l'utilisation des opérateurs génétiques de mutation et de croisement. L'émergence des modules et leur propagation au sein de la population devront donc être étudiés de manière plus approfondie de façon à mieux comprendre les mécanismes qui les sous-tendent.

De nouveaux modules pourraient être créés par fusion de modules existants en un seul nouveau module, de taille plus importante. Si cette possibilité est, a priori, intéressante, elle pose de nombreux problèmes de combinatoire : quand décider de regrouper plusieurs modules ? Quels modules regrouper ? Ces questions pourraient être traitées, par exemple, par des tests statistiques sur les individus les plus performants. Une autre solution, plus simple et plus immédiate, serait d'ajouter un nouveau type de mutation qui fusionnerait ou diviserait des modules. Les problèmes que nous avons traité ici ne nécessitent pas d'assemblage complexe de modules particuliers, aussi ces approches n'apporteraient-elles probablement rien pour les problèmes que nous avons traités. Par contre, pour des problèmes plus complexes et nécessitant la répétition de structures complexes imbriquées les unes dans les autres, cette propriété potentielle de ModNet pourrait être étudiée.

11.2 Apprentissage et évolution

L'évaluation d'un individu est l'étape la plus longue dans le déroulement d'un algorithme évolutionniste. Or, dans les approches présentées jusqu'à présent, aucune exploration de l'espace de recherche n'est effectuée pendant cette étape qui sert uniquement à générer une note. Ajouter de l'apprentissage pendant la durée de vie des individus permettrait de mettre à profit cette période incompressible.

Plusieurs méthodes existent pour faire le lien entre évolution et apprentissage.

11.2.1 Lamarckisme

La première consiste à rétropropager les modifications effectuées dans le génome pendant la durée de vie, de façon à ce que les descendants profitent des caractères acquis par leurs parents. Si cette possibilité d'une transmission des caractères acquis a été envisagée par Jean-Baptiste Lamarck au début du $19^{ème}$, elle a été depuis largement abandonnée par les biologistes. En effet, il semble difficile d'imaginer un mécanisme capable de transcrire les modifications anatomiques ou comportementales survenues durant la vie d'un individu dans l'ADN des cellules germinales de sa progéniture. Cependant, cela ne signifie pas qu'un tel mécanisme soit dépourvu d'intérêt en informatique ou en ingénierie. Il est très facile à implémenter sur des codages directs, plus difficile sur des codages indirects ; mais si, intuitivement, il semble pouvoir accélérer la convergence de l'évolution, les effets correspondants sont, à ce jour, loin d'être systématiques (Gruau et Whitley, 1993; Whitley *et al.*, 1994; Mitchell et Belew, 1996; Sasaki et Tokoro, 1997).

11.2.2 Effet Baldwin

Une autre explication a été avancée pour prendre en compte ces liens apparents entre évolution et apprentissage. Si on imagine qu'il y a effectivement apprentissage ou adaptation pendant la durée de vie des individus et qu'un caractère particulier offre un avantage sélectif important, tous les individus ayant acquis ce caractère vont disposer d'un avantage sélectif sur les autres. Leurs descendants seront, a priori, également susceptibles d'acquérir le même caractère, car ils auront globalement les mêmes prédispositions que leurs parents. De plus, parmi ces individus, ceux qui seront susceptibles d'acquérir ce caractère le plus rapidement seront avantagés par rapports aux autres - qui devront passer du temps et consommer de l'énergie avant de l'acquérir - et ceux qui possèdent le caractère directement dans leur génome seront d'autant plus favorisés. Ils seront donc plus facilement sélectionnés et prendront progressivement le dessus dans la population, jusqu'à ce que la majorité des individus dispose du caractère considéré de façon innée. Ce phénomène a été nommé *sélection organique* ou *effet Baldwin* du nom du biologiste qui l'a proposé (Baldwin, 1896).

L'effet Baldwin ne nécessite pas d'implémenter de mécanisme particulier autre que celui de favoriser la reproduction des individus les plus aptes à acquérir par apprentissage le caractère en question. Il résulte d'un simple effet de bord de l'interaction entre évolution et apprentissage et permet d'obtenir des résultats similaires à une approche Lamarckiste (Gruau et Whitley, 1993; Mayley, 1997; Watson et Wiles, 2002).

11.2.3 Bénéfices et coûts liés à l'apprentissage

Le premier avantage est que l'apprentissage assure une partie de l'exploration de l'espace de recherche. Si un individu est proche d'une solution, mais pas suffisamment pour voir sa fitness augmenter, l'apprentissage peut franchir le pas séparant le phénotype tel que décrit par son génotype et le phénotype-solution. Il peut donc augmenter la fitness de l'individu concerné, ce qui le dote d'un avantage sélectif qui assure sa survie lors des générations suivantes. De plus, si cet individu survit, l'algorithme évolutionniste va pouvoir explorer de façon plus approfondie l'espace de recherche avoisinant cette solution. Ainsi, si un génome a une mauvaise note, mais est suffisamment proche d'une solution intéressante, l'apprentissage va pouvoir trouver cette solution et augmenter la note du génome. La recherche va alors pouvoir continuer dans cette zone alors que, sans apprentissage, elle aurait été plus facilement abandonnée. Hinton et Nowlan (Hinton et Nowlan, 1987) ont étudié ce phénomène sur un cas simple et ont montré que l'apprentissage pouvait lisser l'espace de recherche et donc le rendre plus facile à explorer pour l'algorithme évolutionniste. Ils ont utilisé un Algorithme Génétique pour optimiser une fonction. Leur apprentissage consistait à modifier aléatoirement quelques bits du génome en regardant à chaque fois comment évoluait la note correspondante. La note globale prenait en compte la note initiale ainsi que les autres notes obtenues par "apprentissage" avec une pondération inversement proportionnelle au nombre de bits modifiés. Ils ont observé que le paysage des notations était plus lisse avec cet apprentissage et les solutions d'autant plus faciles à trouver.

De plus, même sans Lamarckisme, les traits acquis par apprentissage peuvent se retrouver progressivement intégrés au génotype grâce aux effets de bords liant évolution et apprentissage, autrement dit grâce à l'effet Baldwin. Le Lamarckisme n'est ainsi pas nécessaire, ce qui simplifie l'implémentation de l'apprentissage et évite de mettre en place des mécanismes de modification du génome - ce qui ne serait pas toujours faisable selon le codage employé.

Le troisième, et sans doute le plus important des gains, réside dans l'ajout de capacités adaptatives liées à l'apprentissage. En effet, l'évolution assure une adaptation à long terme aux modifications lentes de l'environnement, tandis que l'apprentissage adapte l'individu aux petites modifications rapides. De plus, l'apprentissage rend le comportement trouvé beaucoup plus robuste car la présence de capacités adaptatives limite les risques d'exploiter des caractéristiques trop fragiles de l'environnement. Floreano et Urzelai (2001), par exemple, obtiennent par évolution des réseaux de neurones spécialisés qui exploitent des caractéristiques fragiles de leur environnement (taille de l'arène dans laquelle évolue le robot). En remplaçant des connexions de poids fixe par des connexions adaptatives dont le poids varie selon des lois hebbiennes, ils ont obtenu des comportements beaucoup plus robustes qui se transposaient facilement à des environnements de caractéristiques très différentes.

Un dernier point joue en faveur de l'utilisation de l'apprentissage, celui de l'acquisition de caractères complexes. Le passage à des réseaux de plus en plus importants va nécessiter de trouver des moyens pour limiter l'explosion combinatoire de la recherche. Ainsi, les codages développés ne pourront progressivement plus se permettre de disposer d'un paramètre par connexion dans le chromosome. Le codage utilisé devra donc compresser l'information représentant le réseau de façon à conserver des génomes de taille raisonnable sur lesquels l'évolution pourra continuer à fonctionner. Cela signifie que, s'il sera peut-être encore possible de représenter toutes les structures possible, il ne sera sûrement plus possible de représenter toutes les combinaisons de poids existantes. Dans ces conditions, l'emploi d'un algorithme d'apprentissage pourra permettre d'ajuster les paramètres que l'évolution ne pourra pas positionner précisément. Cependant, bien que l'association évolution-apprentissage présente des avantages indéniables, il ne faut pas oublier que l'utilisation d'un apprentissage crée des coûts supplémentaires parfois non négligeables (Mayley, 1996).

Le temps passé à rechercher une solution efficace peut faire baisser la performance moyenne d'un individu. Ainsi un individu ayant besoin d'apprendre un caractère sera défavorisé par rapport à un individu disposant de ce trait de façon innée[94]. De plus, l'apprentissage nécessite un temps minimal pour fonctionner et augmentera donc d'autant la durée de l'évaluation. Plus la recherche effectuée par l'algorithme d'apprentissage sera importante, plus on aura besoin de temps pour évaluer ses performances. L'évaluation devra alors durer suffisamment longtemps pour que l'apprentissage puisse prendre place et pour tester sa stabilité.

Enfin, il ne faut pas perdre de vue que, pendant la phase de recherche liée à l'apprentissage, des comportements dangereux peuvent être mis en œuvre. C'est pourquoi, cette étape doit être

[94] C'est le mécanisme provoquant l'effet Baldwin.

strictement surveillée. Si cela ne semble pas prêter à conséquence lorsqu'il s'agit de faire évoluer un robot Khépéra dans une arène constituée de polystyrène, cela devient plus critique dans le cas de l'évolution de contrôleurs d'hélicoptère, où la moindre erreur de pilotage peut endommager l'appareil ou même blesser d'éventuels spectateurs.

11.2.4 Couplage Modules - Apprentissage

Jusqu'à présent, dans les approches regroupant apprentissage et évolution, l'apprentissage était réalisé de manière uniforme et concernait l'ensemble du réseau de neurones (Gruau et Whitley, 1993; Floreano et Urzelai, 2001). Or ModNet nous permet de faire cohabiter au sein d'un même réseau des modules dont les poids seront constant au cours d'une évaluation avec des modules dont les poids sont susceptibles de changer. Il suffit pour cela de fournir dans le réservoir des modules de paramètres fixes et d'autres modules susceptibles d'être soumis à apprentissage, d'une façon prédéfinie ou elle aussi évoluée. L'évolution pourrait ainsi déterminer les parties du réseaux qui nécessitent d'être adaptatives et celles qui doivent rester fixes. L'apprentissage ne serait alors réalisé que sur une petite partie du réseau, ce qui pourrait offrir de nombreux avantages.

En particulier, le réseau obtenu ne varierait que très peu. Seules les connexions les plus pertinentes seraient susceptibles d'être modifiées, ce qui rendrait le réseau beaucoup plus stable et limiterait les risques de dérive de l'apprentissage. De plus, le nombre de paramètres soumis à l'apprentissage étant fortement réduit, celui-ci aurait plus de chances de converger et agirait beaucoup plus rapidement. En donnant la possibilité à l'évolution de définir des critères différents pour l'apprentissage des différents modules, il serait également possible d'apprendre plusieurs caractéristiques en même temps, en ne modifiant à chaque fois que les connexions liées à la caractéristique concernée.

11.3 Autres applications

11.3.1 Robots marcheurs

Si ce travail a clairement illustré l'intérêt de la conception automatique de modules dans le cas du pendule inversé, les avantages de cette approche se sont avérés beaucoup moins nets dans le cas du contrôle du dirigeable et de l'hélicoptère. L'étude de plateformes différentes - telles que des robots hexapodes, par exemple - devrait permettre d'évaluer les capacités de notre algorithme sur un autre problème complexe impliquant de nombreux degrés de liberté, le problème du contrôle de la marche. En effet ce problème est difficile à résoudre si l'on ne donne pas la symétrie a priori. Il serait, en particulier, intéressant d'étudier les solutions que pourrait générer ModNet sans patron aux solutions obtenues à partir de modules CPG. ModNet étant capable de générer des symétries, il serait notamment possible de vérifier ses capacités sur un problème de ce type, sans imposer la symétrie a priori.

11.3.2 Robots volants

Les essais en vol du dirigeable lenticulaire doivent reprendre en septembre 2003. Nous pourrons à cette occasion vérifier et valider notre simulation ainsi que les contrôleurs que nous avons générés. Il se pourrait, en particulier, que nous constations que l'utilisation de la portance réalisée par les contrôleurs neuronaux exploite une faille de la simulation. En effet, lors d'expériences réelles, l'enveloppe devrait se retrouver rapidement entraînée par la masse d'air, ce qui devrait annuler la portance après une courte période transitoire. Or lorsque le dirigeable simulé a une incidence non nulle et que les moteurs horizontaux sont utilisés, la traînée crée une composante importante sur l'axe longitudinal du dirigeable, ce qui lui permet de contrer le vent. Cet effet ne sera probablement pas présent sur le dirigeable réel. Néanmoins, le contrôleur que nous avons obtenu dans le présent travail serait tout de même utilisable dans cette hypothèse, parce que le pilote pourrait s'opposer au vent en utilisant les moteurs à poussée horizontale[95]. Le dirigeable réel ne serait donc pas, lui non plus, entraîné par la masse d'air pendant les essais en vol.

Il se trouve, par ailleurs, que le dirigeable lenticulaire est susceptible de générer d'autres problèmes que ceux que nous avons étudiés dans cette thèse. L'utilisation des gouvernes, par exemple - lesquelles sont redondantes par rapport aux moteurs, mais plus économiques en énergie - pose des problèmes d'optimisation de la consommation énergétique. De plus, le fait de devoir changer de comportement en fonction du contexte, que ce soit pour économiser de l'énergie ou pour un contrôle plus efficace, devrait nous permettre d'aborder sur cette plateforme le problème des relations entre modularité fonctionnelle - découverte automatiquement par le système - et modularité comportementale - observée par un spectateur extérieur.

Il se trouve, enfin, que nous collaborons actuellement avec l'ENSICA sur un projet de microdrone en vue de participer au concours organisé par l'ONERA en 2005[96]. Ce projet va nous donner l'occasion de tester notre algorithme sur un autre type d'aéronef : les microdrones de type avion. Ce type de plateforme étant très instable et ayant un temps de réponse très rapide, cela nécessitera de disposer de contrôleurs simples pour que les calculs soient rapides et robustes pour stabiliser l'appareil. De plus, plusieurs comportements doivent être implémentés - comme l'exploration rapide d'une zone ou l'observation précise d'un objectif particulier. De tels comportements nécessitent des stratégies de contrôle différentes, pour lesquelles il sera particulièrement intéressant d'étudier les solutions générées par ModNet.

---

[95] En effet, ce contrôleur ne maintient que l'assiette et l'altitude. La présence d'un pilote pour éviter la dérive du dirigeable sera de toute façon indispensable.

[96] `http: //concours-drones.onera.fr/`

Chapitre 12

Conclusion

Dans cette thèse, nous avons présenté ModNet, un codage modulaire de réseaux de neurones qui a permis de résoudre efficacement le problème du contrôle pendule inversé en utilisant seulement les erreurs de positions et non leurs dérivées[97]. Les réseaux générés ont découvert et exploité des modules permettant d'acquérir ces informations. Nous avons ensuite appliqué cet algorithme au contrôle d'un dirigeable lenticulaire puis au contrôle d'un hélicoptère. Sur ces deux plateformes, l'évolution s'est révélée capable de générer des solutions plus efficaces que des architectures définies manuellement, du moins tant que le contrôle restait restreint à un faible nombre de degrés de liberté (allant de deux - roulis et tangage pour l'hélicoptère - à trois - roulis, tangage et altitude - voire à quatre degrés de liberté - roulis, tangage et position pour le dirigeable). De plus, les architectures générées sont très simples et peuvent donc aisément être implémentées dans l'électronique embarquée. Dans ces expériences, l'utilisation de modules a permis d'améliorer les performances moyennes des solutions générées, mais de manière beaucoup moins nette que sur le problème du pendule inversé.

La résolution directe du contrôle complet du dirigeable et de l'hélicoptère (contrôle du roulis, tangage, altitude et position) n'a donné que des résultats médiocres. Cependant, en extrayant des informations de solutions partielles observées ainsi que d'études préliminaires, nous avons défini des patrons indiquant à l'évolution comment relier les modules entre eux pendant la génération aléatoire d'individus de la première génération. L'évolution a alors été guidée dans le choix des structures reliant les capteurs aux effecteurs, tout en devant choisir quels modules utiliser à cette fin et tout en pouvant, par la suite, remettre en cause ces structures par mutations. Cette technique nous a permis d'obtenir des contrôleurs stabilisant à la fois tous les degrés de liberté, tant sur le dirigeable que sur l'hélicoptère.

Il apparait ainsi que deux problèmes sont liés au passage à l'échelle lors de l'évolution de réseaux de neurones : le nombre d'entrées et de sorties, ainsi que la nécessité de découvrir des structures particulières indispensables à la résolution du problème posé. La probabilité de découvrir des solutions dans ces conditions diminue rapidement avec l'augmentation de la complexité du réseau à découvrir. De plus, le temps requis pour évaluer une solution empêche d'utiliser des

[97] qui sont indispensables à la résolution du problème.

populations de taille trop importantes qui permettraient de résoudre ce problème. Une façon de limiter ces problèmes consiste à inclure des connaissances dans la définition de la fonction de fitness ou à décomposer la tâche à résoudre pour se ramener à des problèmes plus simples.

De manière générale l'évolution n'a pas tendance à trouver la solution la plus efficace à un problème, mais plutôt celle qui est la plus facile à découvrir, compte-tenu de l'espace de recherche sur lequel elle fonctionne et des opérateurs de recherche mis à sa disposition. C'est la source du problème principal que nous avons rencontré dans ce travail : les réseaux naturellement obtenus sont les solutions les plus efficaces que l'on peut trouver par mutations et croisements à partir d'un réseau généré aléatoirement. Cependant, pour permettre à l'évolution de converger vers des solutions efficaces, il faut s'assurer que les opérateurs génétiques et la génération aléatoire vont permettre d'explorer des solutions pertinentes. Pour ce faire, nous nous sommes concentré sur deux techniques.

D'une part, l'utilisation de modules nous a permis de protèger des structures de l'effet destructeur des mutations structurelles. De plus, les mutations peuvent provoquer l'ajout de structures complexes et efficaces en une seule opération insécable, tandis que le croisement a permis d'échanger des modules, donc des informations fonctionnelles, entre réseaux. Sur le cas simple du pendule inversé, cette technique s'est révélée très efficace et a permis à l'évolution de converger plus rapidement et vers des solutions beaucoup plus performantes que lorsque les modules ne sont pas disponibles. Dans les cas plus complexes de contrôle de robots volants, l'utilisation de modules a certes amélioré les performances, mais de manière beaucoup moins nette. Ainsi, les modules ont permis d'améliorer les performances moyennes et/ou les performances des meilleurs individus, selon les expériences. Cependant, la différence, si elle est généralement notable, n'est pas systématique. Nous avons également pu remarquer que, dans des expériences impliquant de nombreux degrés de liberté, les expériences avec modules évoluent plus lentement que les expériences sans, avant de rattraper ces dernières et de les dépasser. L'utilisation seule de modules n'a cependant pas permis de résoudre directement le problème du contrôle complet du dirigeable ou de l'hélicoptère, qui n'a été résolu que grâce à l'utilisation de "patrons".

D'autre part, la génération aléatoire exploitant des "patrons" de réseaux a permis de réduire fortement la combinatoire en orientant les recherches vers les solutions les plus pertinentes. Il a été ainsi possible de résoudre le problème du maintien du roulis, du tangage, de l'altitude et de la position de l'hélicoptère et du dirigeable, sans avoir à décomposer le problème.

Au sortir de ce travail, il semble que l'utilisation d'algorithmes évolutionnistes pour générer des contrôleurs neuronaux a peu de chances de s'avérer efficace sur des problèmes à nombreux degrés de liberté si on ne dispose d'aucune procédure pour limiter l'espace de recherche correspondant.

Pour ce faire, nous avons capitalisé ici sur l'utilisation de patrons d'initialisation qui nous ont été suggérés par nos connaissances d'ingénieur ou par des résultats d'expériences préliminaires. Il se pourrait que de telles procédures, ou d'autres, puissent être automatisées à l'avenir et nous avons évoqué dans les perspectives de ce travail quelques directions de recherche à explorer dans ce but.

Annexes

Annexe A

Simulation d'un dirigeable lenticulaire

A.1 Description du dirigeable

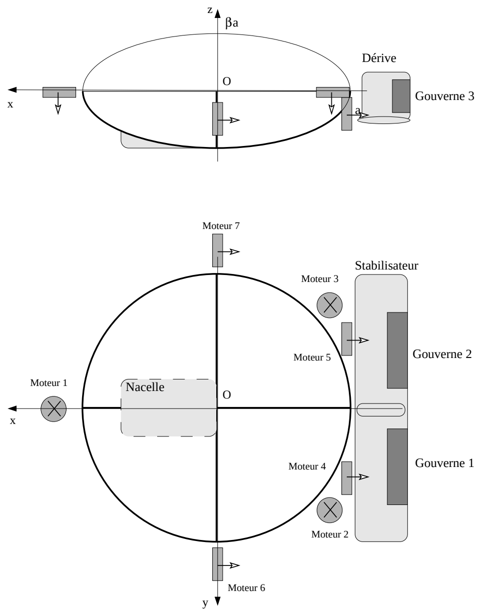


FIG. *A.1: Caractéristiques du dirigeable*

Le dirigeable est composé d'une enveloppe, de la nacelle, des empennages arrières et de différents blocs moteurs.

L'enveloppe du dirigeable est de forme paraboloïdale :

$$y = \frac{r}{4}\left(1 - \left(\frac{x}{r}\right)^2\right) + \frac{r}{12}\sqrt{1 - \left(\frac{x}{r}\right)^2}$$

$a = r$ est le grand rayon du dirigeable, $b = \beta \times a = \frac{r}{4} + \frac{1}{12}$ est le petit.

La nacelle contient la charge utile, principalement l'électronique embarquée pour le contrôle (capteurs, calculateur). Sa position est déterminée de façon à équilibrer le dirigeable.

Les empennages arrières comprennent un stabilisateur horizontal ainsi qu'une dérive. Le stabilisateur comprend deux gouvernes et la dérive une. Ces gouvernes doivent assurer la stabilisation du dirigeable pour des vitesses non nulles (vitesse relative à la masse d'air).

Les blocs moteurs sont répartis sur la périphérie du dirigeable en dessous du plan de symétrie (cf figure A.1). Ils sont de deux types différents. Trois d'entre eux (numérotés 1 à 3 sur la figure A.1) assurent la stabilisation à vitesse faible (lorsque la gouverne est inefficace) et sont de puissance inférieure aux autres. Ils sont disposés dans le plan de symétrie horizontal et dirigés verticalement pour assurer une poussée perpendiculaire au plan horizontal. Les autres moteurs, au nombre de quatre (numérotés 4 à 7), assurent la propulsion de l'aéronef et sont disposés horizontalement dans le plan horizontal contenant le centre de gravité pour éviter tout couple de rotation lors des phases de propulsion. Parmi ceux-ci, les deux moteurs diamétralement opposés (numérotés 6 et 7) pourront assurer un couple de lacet.

A.2 Repères

$OXYZ$ est le repère absolu lié à la terre et $Oxyz$ le repère relatif (figure A.1).

$Gxyz$ est le repère relatif centré sur le centre de gravité lié au dirigeable. Gx est l'axe longitudinal du dirigeable, avec $x > 0$ vers l'arrière. Gy est l'axe transversal, avec $y > 0$ vers la gauche lorsque l'on regarde dans le sens des x positifs. Gz est perpendiculaire au plan Gxy et dirigé vers le haut du dirigeable.

Les forces ou moments sont indicés par le repère dans lequel ils sont exprimés. Indicé par des majuscules : $OXYZ$, indicé par des minuscules : $Gxyz$.

Les angles utilisés pour passer du repère fixe au repère mobile sont l'azimuth ψ, l'assiette longitudinale θ_l et le roulis φ_1. Le passage du repère mobile au repère fixe se fait donc de la façon suivante :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = Tr. \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

avec

$$Tr = \begin{bmatrix} \cos\psi.\cos\theta_l & \cos\psi.\sin\theta_l.\sin\varphi_1 - \sin\psi.\cos\varphi_1 & \cos\psi.\sin\theta_l.\cos\varphi_1 + \sin\varphi_1.\sin\psi \\ \sin\psi.\cos\theta_l & \sin\psi.\sin\theta_l.\sin\varphi_1 + \cos\psi.\cos\varphi_1 & \sin\psi.\sin\theta_l.\cos\varphi_1 - \sin\varphi_1.\cos\psi \\ -\sin\theta_l & \cos\theta_l.\sin\varphi_1 & \cos\theta_l.\cos\varphi_1 \end{bmatrix} \tag{A.1}$$

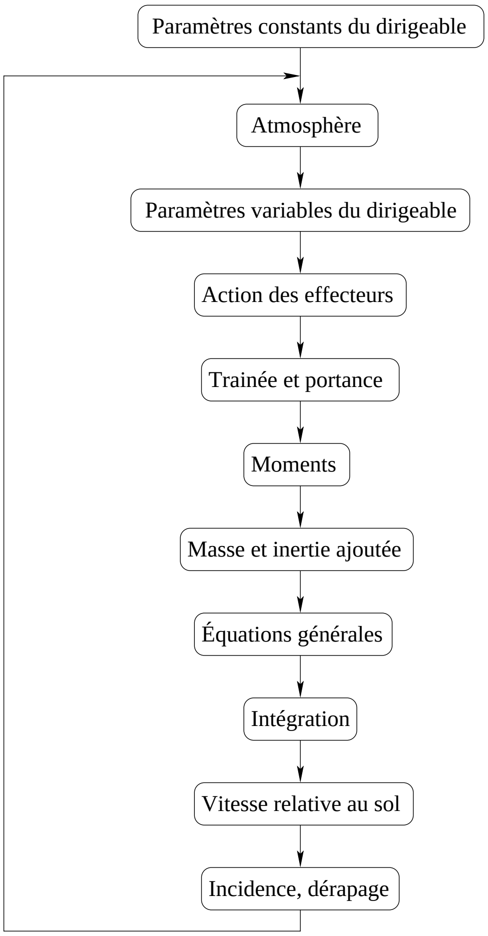


FIG. *A.2: Algorithme complet*

A.3 Algorithme complet

A.4 Équations du système

A.4.1 Équations générales

L'équation générale du système dans le repère fixe est la suivante :

$$M_t\vec{\Gamma} = M_t\vec{g} + \vec{P} + \vec{A}_{OXYZ} + \vec{F}_{OXYZ}$$

$\vec{A}$ est la force résultante aérodynamique : $\vec{A}_{OXYZ}$ si elle est donnée dans le repère $OXYZ$ ou $\vec{A}_{Gxyz}$ dans le repère $Gxyz$.

$\vec{F}$ est la résultante des forces de commandes.

$\vec{P}$ est la poussée d'Archimède.

NB : la signification des noms de variables est donnée à la fin de cette annexe.

Ce qui donne :

$$
\begin{aligned}
X'' &= \frac{1}{M_x}.(A_X + F_X) && \text{(A.2)}\\
Y'' &= \frac{1}{M_y}.(A_Y + F_Y) && \text{(A.3)}\\
Z'' &= \frac{1}{M_z}.(-M_t.g(Z) + \rho(Z).g(Z).V_o.Rp + A_Z + F_Z) && \text{(A.4)}\\
\psi' &= \frac{(q.\sin\varphi_1 + r.\cos\varphi_1)}{\cos\theta_l} && \text{(A.5)}\\
\theta_l' &= q.\cos\varphi_1 - r.\sin\varphi_1 && \text{(A.6)}\\
\varphi_1' &= p + \psi'.\sin\theta_l && \text{(A.7)}\\
p' &= \frac{1}{I_{x1}}.(L - (I_z - I_y).r.q) && \text{(A.8)}\\
q' &= \frac{1}{I_{y1}}.(M - (I_x - I_z).p.r) && \text{(A.9)}\\
r' &= \frac{1}{I_z}.(N - (I_y - I_x).p.r) && \text{(A.10)}
\end{aligned}
$$

$$
\begin{aligned}
A_{OXYZ} &= Tr.A_{Gxyz}\\
F_{OXYZ} &= Tr.F_{Gxyz}
\end{aligned}
$$

$$
A_{Gxyz} = \begin{bmatrix} T_0 \cos I.\cos J + P_0.\sin I.\cos J \\ T_0.\cos I.\sin J + P_0.\sin I.\sin J \\ -T_0.\sin I + P_0.\cos I \end{bmatrix} \tag{A.11}
$$

$$
\Omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix}
$$

est le vecteur rotation autour du centre de gravité dans un repère fixe.

$$
I = \begin{bmatrix} Ix & 0 & 0 \\ 0 & Iy & 0 \\ 0 & 0 & Iz \end{bmatrix}
$$

est la matrice des moments d'inertie.

M_x, M_y et M_z représentent la masse totale du dirigeable avec prise en compte de la masse d'air entrainée.

$$
\begin{aligned}
M_x &= M_t + |\tanh(V_{rx})| 0.1\rho V_0 && \text{(A.12)}\\
M_y &= M_t + |\tanh(V_{ry})| 0.1\rho V_0 && \text{(A.13)}\\
M_z &= M_t + |\tanh(V_{rz})| 0.5\rho V_0 && \text{(A.14)}
\end{aligned}
$$

I_{x1} et I_{y1} sont les coefficients d'inertie prenant en compte la masse d'air entrainée lors de rotations autour d'un axe équatorial : $I_1 = 1.8I$.

A.4.2 Intégration

L'intégration est effectuée avec la méthode de Runge-Kutta d'ordre 4.

On calcule successivement $K1_p$, $K2_p$, $K3_p$ et $K4_p$ pour p appartenant à l'ensemble (X', Y', Z', X, Y, Z, ψ, θ_l, φ_1, P, Q, R).

Le calcul des Ki_p est effectué de la façon suivante :

On commence par sauvegarder dans le vecteur $X(t)$ toutes les coordonnées à intégrer. On calcule ensuite les Ki_p :

$$\begin{aligned}
K_1 &= Step * f(X(t)) \\
K_2 &= Step * f(X(t) + \frac{K_1}{2}) \\
K_3 &= Step * f(X(t) + \frac{K_2}{2}) \\
K_4 &= Step * f(X(t) + K_3)
\end{aligned}$$

Et on en déduit la nouvele valeur du vecteur X

$$X(t+1) = X(t) + \frac{K_1}{6} + \frac{K_2}{3} + \frac{K_3}{3} + \frac{K_4}{6}$$

A.5 Calculs atmosphériques

Tirés de l'ouvrage de Rebuffet, z étant toujours inférieur à 11km.

$$T = 288 - 6,5.\frac{z}{1000} \tag{A.15}$$

$$\rho = 1,2255.(\frac{T}{288})^{4,255} \tag{A.16}$$

$$P = 101325.(\frac{T}{288})^{5.255} \tag{A.17}$$

$$g = 9,80665.(1 - \frac{z}{3183000}) \tag{A.18}$$

A.6 Calcul des paramètres du dirigeable

A.6.1 Assiette vraie et précession

$$\theta = \arccos(\cos\theta_l.\cos\varphi_1) \tag{A.19}$$

$$\gamma = \psi - \arcsin\left(\frac{\sin\varphi_1}{\sin\theta}\right) \tag{A.20}$$

A.6.2 Trainée et portance

La résultante des forces exercées sur l'engin par le vent relatif (appelée force aérodynamique) a pour expression dans le repère lié à la vitesse relative à la masse d'air :

$$F_{xv} = T_0 \tag{A.21}$$

$$F_{yv} = 0 \tag{A.22}$$

$$F_{zv} = P_0 \tag{A.23}$$

$$T_0 = -\frac{1}{2}.\rho.S.V_{rel}^2.C_x \tag{A.24}$$

$$P_0 = \frac{1}{2}.\rho.S.V_{rel}^2.C_{z_i}.I \tag{A.25}$$

T_0 est la trainée, de signe contraire à la vitesse relative à la masse d'air

A.6.3 Masses

$$M_t = M_{solide} + M_{Helium} + \rho.(V_0 - V_{Helium}(\rho)) \tag{A.26}$$

Dans les équations (A.2) et (A.3), nous avons utilisé $M_{t1} = M_t + 0,1.\rho.V_0$ et dans (A.4), nous avons utilisé $M_{t2} = M_t + 0,5.\rho.V_0$ pour prendre en compte la masse d'air déplacée.

A.6.4 Position du centre de gravité

La position du centre de gravité est supposée constante au cours du vol, l'effet du déplacement des masses d'air et d'hélium étant réduit par l'existence de cloisons à l'intérieur de l'enveloppe.

$$\overrightarrow{OG}_{Mot} = \frac{\sum_{i=0}^{NbMot} M_i \times \overrightarrow{OG}_i}{\sum_{i=0}^{NbMot} M_i}$$

$$\overrightarrow{OG}_{Env} = \frac{(\mu_{ex}-\mu_{in})}{(\mu_{ex}+\mu_{in})}.\frac{4}{3}\frac{b}{\pi}$$

$$\overrightarrow{OG} = \frac{M_{Mot}\times\overrightarrow{OG}_{Mot}+M_{Nac}\times\overrightarrow{OG}_{Nac}+M_{Gouv}\times\overrightarrow{OG}_{Gouv}+M_{Env}\times\overrightarrow{OG}_{Env}}{M_{Mot}+M_{Nac}+M_{Gouv}+M_{Env}}$$

A.6.5 Moments

Moments d'inertie

Les moments d'inertie sont calculés à partir de la formule :

$$I = \int r^2 dm$$

Le moment d'inertie total est la somme des contributions de chacun des éléments du dirigeable.

Moment d'inertie de l'enveloppe, de la nacelle, de la gouverne et des moteurs [98]

avec $\mu = (\mu_{ex} + \mu_{in})$.

$$I_{env} = \begin{bmatrix} \frac{\pi}{4}\mu.\left(1+2\beta^2\right).a^4 & 0 & 0 \\ 0 & \frac{\pi}{4}\mu.\left(1+2\beta^2\right).a^4 & 0 \\ 0 & 0 & \frac{\pi}{2}\mu.a^4 \end{bmatrix} \tag{A.27}$$

$$I_{nac} = \begin{bmatrix} \left(y_{nac}^2+z_{nac}^2\right).M_{nac} & 0 & 0 \\ 0 & \left(x_{nac}^2+z_{nac}^2\right).M_{nac} & 0 \\ 0 & 0 & \left(x_{nac}^2+y_{nac}^2\right).M_{nac} \end{bmatrix} \tag{A.28}$$

$$I_{gouv} = \begin{bmatrix} \left(y_{gouv}^2+z_{gouv}^2\right).M_{gouv} & 0 & 0 \\ 0 & \left(x_{gouv}^2+z_{gouv}^2\right).M_{gouv} & 0 \\ 0 & 0 & \left(x_{gouv}^2+y_{gouv}^2\right).M_{gouv} \end{bmatrix} \tag{A.29}$$

$$I_{mot} = \begin{bmatrix} \left(y_{mot}^2+z_{mot}^2\right).M_{mot} & 0 & 0 \\ 0 & \left(x_{mot}^2+z_{mot}^2\right).M_{mot} & 0 \\ 0 & 0 & \left(x_{mot}^2+y_{mot}^2\right).M_{mot} \end{bmatrix} \tag{A.30}$$

Moments des forces aérodynamiques

Ce moment est mesuré en O, on peut trouver sa valeur en G avec le théorème de Huyghens :

$$\vec{M}_G = \vec{M}_O + \overrightarrow{GO} \wedge \vec{F}$$

Soit avec x_O,y_O,z_O les coordonnées de O dans le repère lié à l'engin :

$$M_G = \begin{bmatrix} y_O(P_0.\cos i - T_0 \sin i) - z_O(T_0 \cos i \sin j + P_0 \sin i \sin j) + \frac{1}{2}\rho S l V_{rel}^2 C_m \sin j \\ z_O(T_0 \cos i \cos j + P_0 \sin i \cos j) - x_O(P_0 \cos i - T_0 \sin i) - \frac{1}{2}\rho S l V_{rel}^2 C_m cos j \\ x_O(T_0 \cos i \sin j + P_0 \sin i \sin j) - y_O(T_0 \cos i \cos j + P_0 \sin i \cos j) \end{bmatrix}$$

l est la longueur de référence : $l = \sqrt{S}$, S étant la surface de référence

Moments de la poussée d'Archimède

$$\begin{aligned} \vec{M}_{arch} &= \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix} \wedge Tr^{-1} \begin{bmatrix} 0 \\ 0 \\ \rho g V_0 R p \end{bmatrix} \\ &= \rho g V_0 R p \begin{bmatrix} \cos\theta_l (y_O \cos\varphi_1 - z_O \sin\varphi_1) \\ -z_O \sin\theta_l - x_O \cos\theta_l cos\varphi_1 \\ x_O cos\theta_l \sin\varphi_1 + y_O sin\theta_l \end{bmatrix} \end{aligned}$$

[98] Voir Annexes pour le détail éventuel des calculs.

Moments des forces de commande

La résultante des forces de commande a pour moment (L_c, M_c, N_c). Voir la section A.8 sur l'action des moteurs pour sa valeur.

À cela s'ajoute le moment créé par les gouvernes. La portance d'une gouverne vaut :

$$F_{Gouv} = \frac{1}{2}\rho S_{Gouv} V_{rel}^2 Cz_{gouv} \alpha_{Gouv}$$

avec S_{Gouv} surface de la gouverne, Cz_{gouv} coefficient aérodynamique de la gouverne, α_{Gouv} angle d'incidence de la gouverne ($\alpha_{Gouv} = i_{Gouv} + i_{Cmd}$, i_{Cmd} étant l'angle entre le plan de l'empennage et la gouverne).

La répartition des gouvernes et la combinaison des moments permet de créer 3 moments, un dans chaque axe.

Le moment créé par la gouverne 1 vaut (voir figure A.1 pour la numérotation des gouvernes) :

$$\vec{M}_{Gouv1} = \begin{bmatrix} x_{Gouv1} \\ y_{Gouv1} \\ z_{Gouv1} \end{bmatrix} \wedge \begin{bmatrix} 0 \\ 0 \\ F_{Gouv1} \end{bmatrix}$$

avec

$$F_{Gouv1} = \frac{1}{2}\rho S_{Gouv1} V_{rel}^2 Cz_{GouvH} \alpha_{Gouv1}$$

$\alpha_{Gouv1} = i + i_{Cmd1}$.

Le moment créé par la gouverne 2 vaut :

$$\vec{M}_{Gouv2} = \begin{bmatrix} x_{Gouv2} \\ y_{Gouv2} \\ z_{Gouv2} \end{bmatrix} \wedge \begin{bmatrix} 0 \\ 0 \\ F_{Gouv2} \end{bmatrix}$$

avec

$$F_{Gouv2} = \frac{1}{2}\rho S_{Gouv2} V_{rel}^2 Cz_{GouvH} \alpha_{Gouv2}$$

$\alpha_{Gouv2} = i + i_{Cmd2}$.

Le moment créé par la gouverne 3 vaut :

$$\vec{M}_{Gouv3} = \begin{bmatrix} x_{Gouv3} \\ y_{Gouv3} \\ z_{Gouv3} \end{bmatrix} \wedge \begin{bmatrix} 0 \\ F_{Gouv3} \\ 0 \end{bmatrix}$$

avec

$$F_{Gouv3} = \frac{1}{2}\rho S_{Gouv3} V_{rel}^2 Cz_{GouvH} \alpha_{Gouv3}$$

$\alpha_{Gouv3} = j + i_{Cmd3}$.

Au final :

$$\vec{M}_{Gouv} = \vec{M}_{Gouv1} + \vec{M}_{Gouv2} + \vec{M}_{Gouv3}$$

On obtient donc par sommation :

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} L_c \\ M_c \\ N_c \end{bmatrix} + \vec{M}_{arch} + \vec{M}_G + \vec{M}_{Gouv}$$

A.6.6 Coefficients aérodynamiques

C_x,C_{z_i} et C_{m_i} sont donnés dans le fichier de configuration.
C_z et C_m s'en déduisent par les formules :

$$C_x = C_{x_i} + (1 - C_{x_i}) \sin i^2 \quad \text{(A.31)}$$

$$C_z = \frac{C_{z_i}}{2}. \sin 2i \quad \text{(A.32)}$$

$$C_m = C_{mi} \sin i \cos^3 i \quad \text{(A.33)}$$

auxquelles s'ajoute l'effet de sol :

$$\begin{aligned} \Delta_{C_x} &= \frac{4.10^{-4}}{(\frac{Z}{D})^2} \\ \Delta_{C_z} &= -\frac{1,6.10^{-3}}{(\frac{Z}{D})^2} \\ \Delta_{C_m} &= \frac{6,5.10^{-3}}{(\frac{Z}{D})^2} i + \frac{5,5.10^{-4}}{(\frac{Z}{D})^2} \end{aligned}$$

Z étant l'altitude et D le diamètre.

A.6.7 Incidence et dérapage

L'incidence et le dérapage sont calculés à partir de la vitesse relative à la masse d'air.

$$\vec{V}_{rel} \triangleq \begin{bmatrix} V_{r_x} \\ V_{r_y} \\ V_{r_z} \end{bmatrix} = \begin{bmatrix} X' - V_{ent} \cos Av \cos Jv \\ Y' - V_{ent} \cos Av \sin Jv \\ Z' + V_{ent} \sin Av \end{bmatrix} \quad \text{(A.34)}$$

voir la section A.7 et la figure A.3 pour la définition du vent et la signification de Av et Jv.

On en déduit :

$$i = \arcsin\left(-\frac{Vr_z}{V_{rel}}\right) \quad \text{(A.35)}$$

$$j = \arcsin\left(\frac{1}{\cos i}\frac{Vr_y}{V_{rel}}\right) \quad \text{(A.36)}$$

i étant l'incidence et j le dérapage.

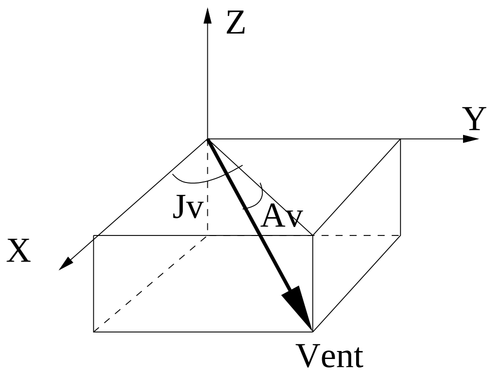


FIG. *A.3: Vent*

A.7 Effet du vent

Le module de la vitesse du vent est généré à partir du spectre de Dryden.

$$V_e = V_m + \sigma s$$

avec V_m moyenne de la vitesse du vent et σ son écart-type. s est la sortie d'un filtre de fonction de transfert :

$$H(p) = \sqrt{T_c}\frac{1 + T_c\sqrt{3}p}{(1 + T_c p)^2}$$

avec un bruit blanc gaussien de moyenne 0 et d'écart type 1 en entrée. Voir section A.9.1 pour le détail des calculs implémentés.

A.8 Action des moteurs

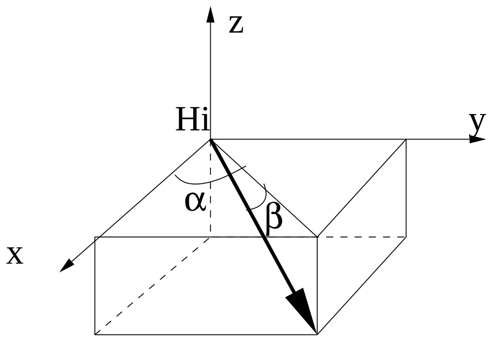


FIG. *A.4: Paramètres de position d'un moteur : α=azimuth, β=site. Ils sont donnés dans le repère lié au dirigeable.*

La force de traction exercée par un moteur dans le repère lié au dirigeable est donnée par la formule :

$$\vec{T}_{raction} = \begin{bmatrix} f \times \cos\alpha \times \cos\beta \\ f \times \sin\alpha \times \cos\beta \\ f \times \sin\beta \end{bmatrix} \tag{A.37}$$

α et β étant l'azimuth et le site donnés dans le repère lié au dirigeable (et non dans un repère lié au moteur qui serait obtenu par rotation à partir du repère lié au dirigeable).

et

$$\begin{aligned} f &= \eta \times P_{max} \times Cmd \times V_{rel} \\ \eta &= \eta_{hel} \times \eta_{meca} \end{aligned}$$

η est le rendement total, η_{hel} étant celui de l'hélice et η_{mec} étant le rendement mécanique du moteur (donnés dans le fichier de configuration).

La force totale s'obtient par sommation des forces de tractions sur l'ensemble des moteurs :

$$\vec{F}_{mot} = \sum_{i=0}^{Nb_{mot}} \vec{T}_{raction,i} \tag{A.38}$$

Le moment résultant s'obtient par la formule :

$$\vec{M}_{mot} = \overrightarrow{GC} \wedge \vec{F}_{mot} + \sum_{i=0}^{Nb_{mot}} \left(\overrightarrow{CH_i} \wedge \vec{T}_{raction,i} \right) \tag{A.39}$$

H_i étant le centre du moteur.

A.9 Calcul du moments d'inertie de l'enveloppe

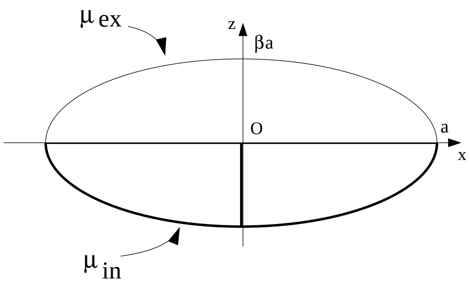


FIG. *A.5: Enveloppe du dirigeable*

Par rapport à Oz

$$I_{zenv} = \int_0^a x^2 dm$$

$$dm = 2\pi x(\mu_{ex} + \mu_{in})dx$$

$$\begin{aligned} I_{zenv} &= \int_0^a 2\pi x(\mu_{ex} + \mu_{in})x^3 dx \\ &= 2\pi(\mu_{ex} + \mu_{in})\frac{a^4}{4} \end{aligned}$$

Par rapport à Ox ou Oy

On commence par calculer le moment d'inertie d'une section d'élipsoïde (qui est donc une élipse) d'épaisseur δ_{ep}, puis on intègre sur Ox (ou Oy).

Moment d'inertie d'une élipse d'épaisseur δ_{ep}

Équation de l'élipse :

$$x^2 + \frac{y^2}{\beta^2} = a^2$$

$$\begin{aligned} I &= \int r^2 dm \\ &= \int_0^{\beta a} \int_0^a (x^2 + y^2)dm \\ &= \int_0^{\beta a} \int_0^a 2(\mu_{ex} + \mu_{in})(x^2 + y^2)\delta ep dx \\ &= 2(\mu_{ex} + \mu_{in})\delta_{ep} \int_0^a ((\beta a)^2 + (1 - \beta^2)x^2)dx \\ &= 2(\mu_{ex} + \mu_{in})\delta_{ep} \left[\beta^2 a^3 + \frac{1-\beta^2}{3}a^3\right] \end{aligned}$$

Intégration

Calcul de I_{xenv}. On remplace δ_{ep} par dx, on remplace a par la valeur du 'rayon' en un point : $\sqrt{a^2 - x^2}$ et on intègre le long de Ox.

$$\begin{aligned} I_{xenv} &= 2\int_0^a 2(\mu_{ex} + \mu_{in}) \left[\beta^2 \sqrt{a^2 - x^2}^3 + \frac{1-\beta^2}{3}\sqrt{a^2 - x^2}^3\right] dx \\ &= \frac{4}{3}(\mu_{ex} + \mu_{in}) \left(1 + 2\beta^2\right) \int_0^a \left(\sqrt{a^2 - x^2}\right)^3 dx \\ &= \frac{4}{3}(\mu_{ex} + \mu_{in}) \left(1 + 2\beta^2\right) a^3 \int_0^{\frac{\pi}{2}} \cos^3 \varphi a \cos \varphi d\varphi \\ &= \frac{4}{3}(\mu_{ex} + \mu_{in}) \left(1 + 2\beta^2\right) a^4 \int_0^{\frac{\pi}{2}} \cos^4 \varphi d\varphi \\ &= \frac{1}{4}\pi(\mu_{ex} + \mu_{in}) \left(1 + 2\beta^2\right) a^4 \end{aligned}$$

avec

$$\begin{aligned}
I &= \int_0^{\frac{\pi}{2}} \cos^4 \varphi d\varphi \\
&= \frac{1}{4}\int_0^{\frac{\pi}{2}} (\cos 2\varphi + 1)^2 \, d\varphi \\
&= \frac{1}{4}\int_0^{\frac{\pi}{2}} \left(\cos^2 2\varphi + 2\cos 2\varphi + 1\right) d\varphi \\
&= \frac{1}{4}\int_0^{\frac{\pi}{2}} \left(\frac{1}{2}\cos 4\varphi + 2\cos 2\varphi + \frac{3}{2}\right) d\varphi \\
&= \frac{1}{4}\left[0 + 0 + \frac{3}{2}\pi\right] \\
&= \frac{3}{16}\pi
\end{aligned}$$

A.9.1 Vent aléatoire

Pour obtenir la valeur du vent aléatoire à partir des échantillons précédents, on calcul la transformée en Z de la fonction de transfert en remplaçant p par $\frac{1-Z^{-1}}{T_e}$ (T_e étant la période d'échantillonage, prise égale à la constante de temps d'intégration). $H(p)$ devient :

$$H(Z) = \sqrt{(T_c)}T_e \frac{T_e + \sqrt{3}T_c - \sqrt{3}T_c Z^{-1}}{(T_e + T_c)^2 - 2T_c(T_e + T_c)Z^{-1} + T_c^2 Z^{-2}}$$

ce qui donne :

$$Y(Z)\left((T_e + T_c)^2 - 2T_c(T_e + T_c)Z^{-1} + T_c^2 Z^{-2}\right) = X(Z)\sqrt{(T_c)}T_e(T_e + \sqrt{3}T_c - \sqrt{3}T_c Z^{-1})$$

ce qui nous permet d'exprimer y_n en fonction des échantillons aux pas de temps précédents :

$$y_n = \frac{1}{(T_e + T_c)^2}\left[2T_c(T_e + T_c)y_{n-1} - T_c^2 y_{n-2} + \sqrt{T_c}T_e(T_e + \sqrt{3}T_c)x_n - \sqrt{3}T_c^{\frac{3}{2}}T_e x_{n-1}\right]$$

A.10 Index des variables utilisées

a Grand axe du dirigeable. Donné dans le fichier de configuration

b Petit axe du dirigeable. Donné dans le fichier de configuration

β Coefficient d'aplatissement : $\beta = \frac{b}{a}$

X Première coordonnée du **centre de gravité** dans le **repère absolu**. Calculé par double intégration de (A.2)

Y Deuxième coordonnée du **centre de gravité** dans le **repère absolu**. Calculé par double intégration de (A.3)

Z Troisième coordonnée du **centre de gravité** dans le **repère absolu**. Calculé par double intégration de (A.4)

$\mathbf{X}'$ Première coordonnée de la **vitesse du centre de gravité** dans le **repère absolu**. Calculé par intégration de (A.2)

$\mathbf{Y}'$ Deuxième coordonnée de la **vitesse du centre de gravité** dans le **repère absolu**. Calculé par intégration de (A.3)

$\mathbf{Z}'$ Troisième coordonnée de la **vitesse du centre de gravité** dans le **repère absolu**. Calculé par intégration de (A.4)

$\mathbf{X}''$ Première coordonnée de l'**accélération du centre de gravité** dans le **repère absolu**. Calculé dans (A.2)

$\mathbf{Y}''$ Deuxième coordonnée de l'**accélération du centre de gravité** dans le **repère absolu**. Calculé dans (A.3)

$\mathbf{Z}''$ Troisième coordonnée de l'**accélération du centre de gravité** dans le **repère absolu**. Calculé dans (A.4)

ρ **Masse volumique de l'air**. Calculé en fonction de l'altitude dans (A.16)

θ **Assiette vraie** ou **nutation**. Angle des plans Gxy et OXY, qui est aussi l'angle entre Gz et OZ, ou encore l'angle entre OXY et la ligne de plus grande pente de Gxy. Calculé dans (A.19)

γ **Précession**. Angle des plans verticaux contenant l'un OX, l'autre la ligne de plus grande pente, c'est aussi l'angle que fait OY avec la ligne des noeuds. Calculé dand (A.20)

Tr Matrice de transition entre le repère relatif et absolu : $OM = Tr.Gm$

φ_1 **Roulis**. Angle de rotation autour de Gx. Calculé par intégration

θ_l **Assiettte longitudinale**. Angle entre Gx et le plan horizontal OXY lié à la terre.

ψ **Azimuth**. Angle que fait la projection de Gx sur le plan XOY avec OX. Lorsque OX est dirigé vers le nord, l'azimuth est appelé **cap**.

p Rotation (vitesse angulaire) autour de GX (repère fixe).

q Rotation (vitesse angulaire) autour de GY (repère fixe).

r Rotation (vitesse angulaire) autour de GZ (repère fixe).

Rp Coefficient de remplissage en hélium. Donné dans le fichier de configuration (supposé constant).

$\mathbf{M_t}$ Masse totale. Calculé dans (A.26).

$\mathbf{M_x}$ Masse totale avec prise en compte de l'air entrainé. Calculé dans (A.13).

$\mathbf{M_y}$ Masse totale avec prise en compte de l'air entrainé. Calculé dans (A.14).

$\mathbf{M_z}$ Masse totale avec prise en compte de l'air entrainé. Calculé dans (A.14).

L 1^{ere} Composante du moment total par rapport à G.

M 2^{eme} Composante du moment total par rapport à G.

N 3^{eme} Composante du moment total par rapport à G.

$\mathbf{L_c}$ 1^{ere} Composante du moment des forces de commande par rapport à G.

$\mathbf{M_c}$ 2^{eme} Composante du moment des forces de commande par rapport à G.

$\mathbf{N_c}$ 3^{eme} Composante du moment des forces de commande par rapport à G.

$\mu_{\mathbf{ex}}$ Densité surfacique de l'extrados. Donné dans le fichier de configuration.

$\mu_{\mathbf{in}}$ Densité surfacique de l'intrados (structure incluse). Donné dans le fichier de configuration.

i Incidence du dirigeable. Calculé dans (A.35).

j Dérapage du dirigeable. Calculé dans (A.36).

Annexe B

Exemple de fichier de configuration de SFERES

Fichier de configuration des expériences portant sur le maintien de l'assiette et de l'altitude du dirigeable. Expérience témoin avec contrôle à base de PID.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE config PUBLIC "SFERES DTD"
"http://miriad.lip6.fr/SFERES/sferes4.dtd">
  <sferes  host="manta.lip6.fr" pid="12345">
    <evolution type="evolution" begin="0" end="10000"
contextfrequency="10" statfrequency="1" flush="10"/>
   <visualisation type="visualisation" />
   <generation index="0">
   <context date="06/02/2002" counter="0">
      <simulator type="dirigeable">
<initagents>
  <agent type="alpha">

          <!--Capteur theta1 -->
     <sensor type="inclino">
             <target type="arch" index="0" input="0" />
             <parameter name="coord" type="uint">0</parameter>
             <parameter name="noise" type="float">0.0001</parameter>
          </sensor>

          <!--Capteur phi1 -->
     <sensor type="inclino">
             <target type="arch" index="1" input="0" />
```

```
            <parameter name="coord" type="uint">1</parameter>
            <parameter name="noise" type="float">0.0001</parameter>
         </sensor>

         <!--Capteur x cible  -->
   <sensor type="targetPosition">
            <target type="arch" index="2" input="0" />
            <parameter name="coord" type="uint">2</parameter>
            <parameter name="noise" type="float">0.0001</parameter>
         </sensor>

         <!--Capteur y cible  -->
   <sensor type="targetPosition">
            <target type="arch" index="3" input="0" />
            <parameter name="coord" type="uint">3</parameter>
            <parameter name="noise" type="float">0.0001</parameter>
         </sensor>

         <!--Capteur Altimètre Différentiel  -->
   <sensor type="altimetreDiff">
            <target type="arch" index="4" input="0" />
            <parameter name="noise" type="float">0.0001</parameter>
         </sensor>

         <!--Capteur Compas diff -->
   <sensor type="PsiDiff">
            <target type="arch" index="5" input="0" />
            <parameter name="noise" type="float">0.0</parameter>
         </sensor>

          <!-- Moteur de sustentation 0 -->
  <effector type="moteur">
            <parameter name="alpha" type="float">0.0</parameter>
            <parameter name="beta" type="float">1.57</parameter>
            <parameter name="m" type="float">0.9</parameter>
            <parameter name="powermax" type="float">500.0</parameter>
            <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
            <parameter name="propellerefficiency" type="float">
0.75</parameter>
```

```
              <parameter name="coord" type="vfloat">
5.0 0.0 0.0</parameter>
    </effector>

            <!-- Moteur de sustentation 1 -->
    <effector type="moteur">
              <parameter name="alpha" type="float">0.0</parameter>
              <parameter name="beta" type="float">1.57</parameter>
              <parameter name="m" type="float">0.9</parameter>
              <parameter name="powermax" type="float">500.0</parameter>
              <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
              <parameter name="propellerefficiency" type="float">
0.75</parameter>
              <parameter name="coord" type="vfloat">
-2.5 4.7 0.0</parameter>
    </effector>

            <!-- Moteur de sustentation 2 -->
    <effector type="moteur">
              <parameter name="alpha" type="float">0.0</parameter>
              <parameter name="beta" type="float">1.57</parameter>
              <parameter name="m" type="float">0.9</parameter>
              <parameter name="powermax" type="float">500.0</parameter>
              <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
              <parameter name="propellerefficiency" type="float">
0.75</parameter>
              <parameter name="coord" type="vfloat">
-2.5 -4.7 0.0</parameter>
    </effector>

            <!-- Moteur de propulsion 3 -->
    <effector type="moteur">
              <parameter name="alpha" type="float">0.0</parameter>
              <parameter name="beta" type="float">0.0</parameter>
              <parameter name="m" type="float">0.9</parameter>
              <parameter name="powermax" type="float">500.0</parameter>
              <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
```

```
            <parameter name="propellerefficiency" type="float">
0.75</parameter>
            <parameter name="coord" type="vfloat">
-4.75 1.25 0.0</parameter>
    </effector>

          <!-- Moteur de propulsion 4 -->
    <effector type="moteur">
            <parameter name="alpha" type="float">0.0</parameter>
            <parameter name="beta" type="float">0.0</parameter>
            <parameter name="m" type="float">0.9</parameter>
            <parameter name="powermax" type="float">500.0</parameter>
            <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
            <parameter name="propellerefficiency" type="float">
0.75</parameter>
            <parameter name="coord" type="vfloat">
-4.75 -1.25 0.0</parameter>
    </effector>

          <!-- Moteur de propulsion+lacet 5 -->
    <effector type="moteur">
            <parameter name="alpha" type="float">0.0</parameter>
            <parameter name="beta" type="float">0.0</parameter>
            <parameter name="m" type="float">0.9</parameter>
            <parameter name="powermax" type="float">500.0</parameter>
            <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
            <parameter name="propellerefficiency" type="float">
0.75</parameter>
            <parameter name="coord" type="vfloat">
0.0 4.75 0.0</parameter>
    </effector>


          <!-- Moteur de propulsion+lacet 6 -->
    <effector type="moteur">
            <parameter name="alpha" type="float">0.0</parameter>
            <parameter name="beta" type="float">0.0</parameter>
            <parameter name="m" type="float">0.9</parameter>
```

```
            <parameter name="powermax" type="float">500.0</parameter>
            <parameter name="mecanicalefficiency" type="float">
0.7</parameter>
            <parameter name="propellerefficiency" type="float">
0.75</parameter>
            <parameter name="coord" type="vfloat">
0.0 -4.75 0.0</parameter>
    </effector>

          <!-- Gouverne Horizontale 0 -->
    <effector type="gouverne">
            <parameter name="numgouv" type="uint">0</parameter>
            <parameter name="max" type="float">1.0</parameter>
    </effector>

          <!-- Gouverne Horizontale 1 -->
    <effector type="gouverne">
            <parameter name="numgouv" type="uint">1</parameter>
            <parameter name="max" type="float">1.0</parameter>
    </effector>

          <!-- Gouverne Verticale 2 -->
    <effector type="gouverne">
            <parameter name="numgouv" type="uint">2</parameter>
            <parameter name="max" type="float">1.0</parameter>
    </effector>

         <!-- Arch 0: AdC Thetal -->
         <architecture type="simplePID" input="1">
    <target type="arch" index="6" input="1"/>
          <parameter name="numchr" type="uint">0</parameter>
          <parameter name="deltat" type="float">0.025</parameter>
          <parameter name="internalcomputation" type="uint">1</parameter>
          <parameter name="integralmax" type="float">10.0</parameter>
         </architecture>

         <!-- Arch 1: AdC Phi1 -->
         <architecture type="simplePID" input="1">
    <target type="arch" index="6" input="0"/>
          <parameter name="numchr" type="uint">1</parameter>
```

```
         <parameter name="deltat" type="float">0.025</parameter>
         <parameter name="internalcomputation" type="uint">1</parameter>
         <parameter name="integralmax" type="float">10.0</parameter>
        </architecture>

        <!-- Arch 2: AdC X cible (cst) -->
        <architecture type="cst" input="1">
<target type="arch" index="6" input="3"/>
         <parameter name="value" type="float">0.0</parameter>
        </architecture>

        <!-- Arch 3: AdC Y cible (cst) -->
        <architecture type="cst" input="1">
<target type="arch" index="6" input="5"/>
         <parameter name="value" type="float">0.0</parameter>
        </architecture>

        <!-- Arch 4: AdC Z  (cst) -->
        <architecture type="simplePID" input="1">
<target type="arch" index="6" input="2"/>
         <parameter name="numchr" type="uint">2</parameter>
         <parameter name="deltat" type="float">0.025</parameter>
         <parameter name="internalcomputation" type="uint">1</parameter>
         <parameter name="integralmax" type="float">10.0</parameter>
        </architecture>

        <!-- Arch 5: AdC Psir -->
        <architecture type="cst" input="1">
<target type="arch" index="6" input="4"/>
         <parameter name="value" type="float">0.0</parameter>
        </architecture>

        <!-- Arch 6: Conversion contrôleurs vers moteurs -->
        <architecture type="alphaCtoM" input="6">
<target type="eff" index="0"/>
<target type="eff" index="1"/>
<target type="eff" index="2"/>
<target type="eff" index="3"/>
<target type="eff" index="4"/>
<target type="eff" index="5"/>
```

```
<target type="eff" index="6"/>
        <parameter name="numchr" type="int">-1</parameter>
        <parameter name="coeffrotate" type="float">0.5</parameter>
        <parameter name="coeffsustenttangage" type="float">0.25</parameter>
        <parameter name="coeffsustentmontee" type="float">0.5</parameter>
       </architecture>

       <!-- Arch 7: Conversion contrôleurs vers gouvernes -->
       <architecture type="alphaCtoG" input="3">
<target type="eff" index="7"/>
<target type="eff" index="8"/>
<target type="eff" index="9"/>
       </architecture>

       <!-- Arch 8: AdC Gouverne 0 (cst) -->
       <architecture type="cst" input="0">
<target type="arch" index="7" input="0"/>
        <parameter name="value" type="float">0.0</parameter>
       </architecture>

       <!-- Arch 9: AdC Gouverne 0 (cst) -->
       <architecture type="cst" input="0">
<target type="arch" index="7" input="1"/>
        <parameter name="value" type="float">0.0</parameter>
       </architecture>

       <!-- Arch 10: AdC Gouverne 0 (cst) -->
       <architecture type="cst" input="0">
<target type="arch" index="7" input="2"/>
        <parameter name="value" type="float">0.0</parameter>
       </architecture>


       <parameter name="a" type="float">5.0</parameter>
       <parameter name="b" type="float">1.67</parameter>
       <parameter name="muex" type="float">0.12</parameter>
       <parameter name="muin" type="float">0.2</parameter>
       <parameter name="nbmoteurs" type="uint">7</parameter>
       <parameter name="gouv" type="vfloat">-5.6 0.0 0.0</parameter>
       <parameter name="gouv1" type="vfloat">-5.6 2.5 0.0</parameter>
```

```
          <parameter name="gouv2" type="vfloat">-5.6 -2.5 0.0 </parameter>
          <parameter name="gouv3" type="vfloat">-5.6 0.0 0.0</parameter>
          <parameter name="mgouv" type="float">12.0</parameter>
          <parameter name="sgouv1" type="float">4.0</parameter>
          <parameter name="sgouv2" type="float">4.0</parameter>
          <parameter name="sgouv3" type="float">4.0</parameter>
          <parameter name="nac" type="vfloat">1.8 0.0 -1.7</parameter>
          <parameter name="mnac" type="float">42.</parameter>
          <parameter name="remplissage" type="float">0.85</parameter>
          <parameter name="cx" type="float">0.07</parameter>
          <parameter name="czi" type="float">1.</parameter>
          <parameter name="cmi" type="float">0.04</parameter>
          <parameter name="optparam" type="int">-1</parameter>
  </agent>
</initagents>
<indivagent>
  0
</indivagent>
<scheduler>
  <event at="0" do="NewExperiment" every="0"/>
  <event at="0" do="Initialize" every="0"/>
  <event at="0" do="TrimChange" every="0">
          <parameter type="float">0.2</parameter>
          <parameter type="float">0.15</parameter>
          <parameter type="float">0.</parameter>
  </event>
  <event at="0" do="OrderChange" every="0">
          <parameter type="float">0.0</parameter>
          <parameter type="float">100.0</parameter>
          <parameter type="float">0.0</parameter>
  </event>
  <event at="0" do="WindChange" every="0">
          <parameter type="float">0.0</parameter>
          <parameter type="float">1.0</parameter>
          <parameter type="float">60</parameter>
          <parameter type="float">1.87</parameter>
          <parameter type="float">0.</parameter>
  </event>
  <event at="1" do="RefreshAgents" every="1"/>
  <event at="1" do="RefreshWorld" every="1">
```

```
</event>
<event at="1000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">-0.15</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="1000" do="WindChange" every="0">
         <parameter type="float">4.0</parameter>
         <parameter type="float">1.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">-2.5</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="2000" do="TrimChange" every="0">
         <parameter type="float">0.1</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="3000" do="TrimChange" every="0">
         <parameter type="float">-0.1</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="4000" do="TrimChange" every="0">
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="5000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="5000" do="OrderChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">150.0</parameter>
         <parameter type="float">0.0</parameter>
</event>
<event at="5000" do="WindChange" every="0">
         <parameter type="float">3.0</parameter>
```

```
         <parameter type="float">1.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">2.57</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="6000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="7000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="8000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="9000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>

  <event at="10000" do="NewExperiment" every="0"/>

<event at="10000" do="Initialize" every="0"/>
   <event at="10000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="10000" do="WindChange" every="0">
         <parameter type="float">4.0</parameter>
         <parameter type="float">2.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">2.</parameter>
         <parameter type="float">0.</parameter>
```

```
</event>
<event at="10000" do="OrderChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">150.0</parameter>
         <parameter type="float">0.0</parameter>
</event>
<event at="10001" do="RefreshAgents" every="1"/>
<event at="10001" do="RefreshWorld" every="1">
</event>
   <event at="11000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="12000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="13000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">0.1</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="14000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="15000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="15000" do="WindChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">1.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">-2.</parameter>
         <parameter type="float">0.</parameter>
```

```
</event>
<event at="15000" do="OrderChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">50.0</parameter>
         <parameter type="float">0.0</parameter>
</event>
   <event at="16000" do="TrimChange" every="0">
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="17000" do="TrimChange" every="0">
         <parameter type="float">0.25</parameter>
         <parameter type="float">-0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="18000" do="TrimChange" every="0">
         <parameter type="float">0.1</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
   <event at="19000" do="TrimChange" every="0">
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>

<event at="20000" do="NewExperiment" every="0"/>

<event at="20000" do="Initialize" every="0"/>
<event at="20000" do="TrimChange" every="0">
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.15</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="20000" do="WindChange" every="0">
         <parameter type="float">6.0</parameter>
         <parameter type="float">2.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">-2.57</parameter>
```

```
        <parameter type="float">0.</parameter>
</event>
<event at="20000" do="CoordChange" every="0">
        <parameter type="float">0.0</parameter>
        <parameter type="float">0.0</parameter>
        <parameter type="float">200.0</parameter>
</event>
<event at="20000" do="OrderChange" every="0">
        <parameter type="float">0.0</parameter>
        <parameter type="float">100.0</parameter>
        <parameter type="float">0.0</parameter>
</event>
<event at="20001" do="RefreshAgents" every="1"/>
<event at="20001" do="RefreshWorld" every="1">
</event>
<event at="24000" do="TrimChange" every="0">
        <parameter type="float">0.25</parameter>
        <parameter type="float">0.15</parameter>
        <parameter type="float">0.</parameter>
</event>
<event at="25000" do="OrderChange" every="0">
        <parameter type="float">0.0</parameter>
        <parameter type="float">50.0</parameter>
        <parameter type="float">0.0</parameter>
</event>
<event at="25000" do="WindChange" every="0">
        <parameter type="float">3.0</parameter>
        <parameter type="float">0.0</parameter>
        <parameter type="float">60</parameter>
        <parameter type="float">3.14</parameter>
        <parameter type="float">0.</parameter>
</event>
<event at="28000" do="TrimChange" every="0">
        <parameter type="float">0.2</parameter>
        <parameter type="float">-0.15</parameter>
        <parameter type="float">0.</parameter>
</event>

<event at="30000" do="NewExperiment" every="0"/>
```

```
<event at="30000" do="Initialize" every="0"/>
<event at="30000" do="TrimChange" every="0">
         <parameter type="float">-0.2</parameter>
         <parameter type="float">-0.1</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="30000" do="WindChange" every="0">
         <parameter type="float">2.0</parameter>
         <parameter type="float">1.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">2.57</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="30000" do="CoordChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">0.0</parameter>
         <parameter type="float">50.0</parameter>
</event>
<event at="30000" do="OrderChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">200.0</parameter>
         <parameter type="float">0.0</parameter>
</event>
<event at="30001" do="RefreshAgents" every="1"/>
<event at="30001" do="RefreshWorld" every="1">
</event>
<event at="35000" do="WindChange" every="0">
         <parameter type="float">0.0</parameter>
         <parameter type="float">0.0</parameter>
         <parameter type="float">60</parameter>
         <parameter type="float">2.57</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="35000" do="TrimChange" every="0">
         <parameter type="float">0.2</parameter>
         <parameter type="float">0.25</parameter>
         <parameter type="float">0.</parameter>
</event>
<event at="40000" do="NewExperiment" every="0"/>
```

```
      </scheduler>
      <parameter name="step" type="float">0.025</parameter>
      <parameter name="displaySize" type="vuint">352 288</parameter>
     </simulator>
      <population type="ranksimple" size="100" >
<models>
  <individual type="individual">
    <genome>
      <chromosome type="es">
<parameter type="esparam" name="es">
                 <parameter name="crosstype" type="uint">0</parameter>
                 <parameter name="xmax" type="float">10.0</parameter>
                 <parameter name="xmin" type="float">-10.0</parameter>
                 <parameter name="sigmamax" type="float">0.10</parameter>
                 <parameter name="sigmamin" type="float">0.001</parameter>
                 <parameter name="x" type="vfloat">0. 0. 0.</parameter>
</parameter>
      </chromosome>
      <chromosome type="es">
<parameter type="esparam" name="es">
                 <parameter name="crosstype" type="uint">0</parameter>
                 <parameter name="xmax" type="float">10.0</parameter>
                 <parameter name="xmin" type="float">-10.0</parameter>
                 <parameter name="sigmamax" type="float">0.10</parameter>
                 <parameter name="sigmamin" type="float">0.001</parameter>
                 <parameter name="x" type="vfloat">0. 0. 0.</parameter>
</parameter>
      </chromosome>
      <chromosome type="es">
<parameter type="esparam" name="es">
                 <parameter name="crosstype" type="uint">0</parameter>
                 <parameter name="xmax" type="float">10.0</parameter>
                 <parameter name="xmin" type="float">-10.0</parameter>
                 <parameter name="sigmamax" type="float">0.10</parameter>
                 <parameter name="sigmamin" type="float">0.001</parameter>
                 <parameter name="x" type="vfloat">0. 0. 0.</parameter>
</parameter>
      </chromosome>
    </genome>
    <fitness type="alphaVCetDECNom">
```

```
     <parameter name="fitvalue" type="vfloat" >0. 0. 0. 0. 0. 0.</parameter>
     <parameter name="coeff" type="vfloat" >1. 1. 0. 0. 1. 0.</parameter>
     <parameter name="thetalmax" type="float" >0.35</parameter>
     <parameter name="phi1max" type="float" >0.35</parameter>
     <parameter name="deltaxmax" type="float" >20.</parameter>
     <parameter name="deltaymax" type="float" >20.</parameter>
     <parameter name="deltazmax" type="float" >100.</parameter>
     <parameter name="mortsubite" type="vuint" >1 1 0 0 0 0</parameter>
            </fitness>
  </individual>
</models>
<parameter type="float" name="keeprate">0.6</parameter>
<parameter type="float" name="initialAleatPop">100.</parameter>
      </population>
</context>
     <statistics>
      <statistic type="best"/>
      <statistic type="bestfit" show="yes">
<parameter type="ushort" name="precision">6</parameter>
<parameter type="ushort" name="width">6</parameter>
      </statistic>
      <statistic type="bestindiv" show="no"/>
      <statistic type="average" show="yes"/>
     </statistics>
   </generation>
 </sferes>
```

Bibliographie

[Anderson et Miller, 1990] C. W. Anderson et W. T. Miller. *Neural Networks for Control*, chapitre Challenging Control Problems. MIT Press, Cambridge, 1990.

[Astor et Adami, 2000] J. C. Astor et C. Adami. A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6 :189–218, 2000.

[Astor, 1998] J. C. Astor. A developmental model for the evolution of artificial neural networks : Design, implementation, evaluation. Master's thesis, University of Heidelberg, September 1998.

[Atkins, 1999] E. M. Atkins. *Plan Generation and Hard Real-time Execution with Application to Safe, Autonomous Flight*. PhD thesis, University of Michigan, 1999.

[Atkinson et Birch, 1970] J. Atkinson et D. Birch. *The dynamics of action*. John Wiley and Sons, 1970.

[Augustsson *et al.*, 2002] P. Augustsson, K. Wolff, et P. Nordin. Creation of a learning, flying robot by means of evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO02)*, 2002.

[Bäck *et al.*, 1991] Bäck, F. Hoffmeister, et H. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991.

[Bagnell et Schneider, 2001] J. Bagnell et J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation 2001, IEEE*, May 2001.

[Baldwin, 1896] J. M. Baldwin. A new factor in evolution. *American Naturalist*, 30 :441–451, 1896.

[Bellili *et al.*, 2002] A. Bellili, M. Gilloux, et P. Gallinari. Reconnaissance de chiffres manuscrits par un système hybride mlp-svm. In *13 ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et d'Intelligence Artificielle*, Angers, janvier 2002.

[Bellman, 1957] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Belton *et al.*, 1997] W. A. Belton, C. Bil, F. Valentinis, et G. A. DeCosta. An autonomous system for atmospheric research applications. In *Proceedings AAC7*, Sydney, Australia, 1997.

[Ben-Tal, 1979] A. Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In *Multiple Criteria Decision Making Theory and Application*, éditeurs Fandel et Gal, pages 1–11. SpringerVerlag, 1979.

[Blynel et D., 2002] J. Blynel et Floreano D. Levels of dynamics and adaptive behavior in evolutionary neural controllers. In *Proceedings of the seventh International Conference on Simulation of Adaptive Behavior (SAB 2002)*, éditeurs B. Hallam, D. Floreano, J. Hallam, G. Hayes, et Meyer J.-A., 2002.

[Boers *et al.*, 1993] J. W. Boers, E., H. Kuiper, B. L. M. Happel, et I. G. Springhuizen-Kuiper. Designing modular artificial neural networks. Rapport technique, 1993.

[Bongard et Pfeifer, 2002] J. C. Bongard et R. Pfeifer. A method for isolating morphological effects on evolved behaviour. In *Proceedings of the seventh International Conference on Simulation of Adaptive Behavior (SAB 2002)*, éditeurs B. Hallam, D. Floreano, J. Hallam, G. Hayes, et Meyer J.-A., 2002.

[Bongard, 2002] J. C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, éditeurs D. B. Fogel, M. A. El-Sharkawi, Yao X., Greenwood G., H. Iba, P. Marrow, et M. Shackleton, pages 1872–1877. IEEE Press, 2002.

[Bosse, 1997] M. C. Bosse. A vision augmented navigation system for an autonomous helicopter. Master's thesis, Boston University, 1997.

[Bouchon-Meunier, 1995] B. Bouchon-Meunier. *La logique floue et ses applications*. Addisson-Wesley, 1995.

[Boyer, 2002] G. Boyer. Projet remanta : une initiative onera sur les drones miniatures à comportement biomimétique. Journées MicroDrones, Toulouse, 2002.

[Braitenberg, 1984] V. Braitenberg. *Vehicles*. Cambridge, MA :MIT Press, 1984.

[Brave, 1996] S. Brave. Evolving deterministic finite automata using cellular encoding. In *Genetic Programming 1996 : Proceedings of the first annual conference*, éditeurs David Goldberg E. David Fogel B. John Koza, R. et L. Rick Riolo, Stanford university, 28-31 july 1996. Cambridge, MA : MIT Press.

[Brinker et Wise, 2000] J. S. Brinker et K. A. Wise. Flight testing of a reconfigurable flight control law on the x-36 tailless fighter aircraft. In *AIAA-2000-3941*, 2000.

[Buessler et Urban, 2002] J. L. Buessler et J. P. Urban. *Biologically Inspired Robot Behavior Engineering*, chapitre Modular Neural Architectures for Robotics. Springer Verlag, 2002.

[Bäck et Schwefel, 1995] T. Bäck et H. P. Schwefel. Evolution strategies i : Variants and their computational implementation. In *Genetic Algorithms in Engineering and Computer Science, Proc. First Short Course EUROGEN-95*. 1995.

[Carvalho *et al.*, 2001] J. Carvalho, E. Paiva De, J. Azinheira, P. Ferreira, S. Ramos, J. nad Bueno, M. Bergerman, S. Maeta, L. Mirisola, B. Faria, et A. Elfes. Classic and robust pid heading control of an unmanned robotic airship. In *International Symposium on Intelligent Robotic Systems*, Toulouse, France, 2001.

[Chapuis *et al.*, 2001] J. Chapuis, C. Eck, M. Kottmann, M. Sanvido, et O. Tanner. Control of helicopters. In *Control of Complex Systems*, éditeur Albertos P. Blanke M. Isidori A. Schaufelberger W. Sanz R. Astr, K. J.öm. Springer, 2001.

[Coelho *et al.*, 1998] L. Coelho, M. Campos, et V. Kumar. Computer-vision based navigation for autonomous blimps. In *European Conference on Computer Vision*, Germany, 1998.

[Cooper et Vidal, 1994] M. G. Cooper et J. J. Vidal. Genetic design of fuzzy controllers : The cart and jointed-pole problem. In *Third IEEE International Conference on Fuzzy Systems*, 1994.

[Coulom, 2002] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[Crosnier *et al.*, 2001] A. Crosnier, B. Jouvencel, G. Abba, et R. Zapata. *Ingénierie de la commande des systèmes : techniques de base*. TECHNOSUP, 2001.

[Cybenko, 1988] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Rapport technique, Departement of computer science, Tufts University, Medford, MA, 1988.

[Cybenko, 1989] G. Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signal and Systems*, (2) :303–314, 1989.

[Dandekar *et al.*, 2001] T. Dandekar, F. Du, R. Schirmer, et S. Schmidt. Medical target prediction from genome sequence : combining different sequence analysis algorithms with expert knowledge and input from artificial intelligence approaches. *Comput Chem*, 26 :15–21, 2001.

[Deb *et al.*, 2000] K. Deb, S. Agrawal, A. Pratab, et T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization : NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, éditeurs Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, J. J. Merelo, et Hans-Paul Schwefel, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

[DiFerdinando *et al.*, 2001] A. DiFerdinando, R. Calabretta, et D. Parisi. Evolving modular architectures for neural networks. In *Proceedings Sixth Neural Computation and Psychology Workshop Evolution, Learning, and Development*, éditeurs Robert French et Jacques Sougné, 2001.

[Doncieux et Meyer, 2003] S. Doncieux et J.-A. Meyer. Evolving neural networks for the control of a lenticular blimp. In *Applications of Evolutionary Computing, EvoWorkshops2003 : EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, EvoSTIM*, éditeurs G. R. Raidl, S. Cagnoni, J. J. Romero Cardalda, D. W. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, E. Marchiori, J.-A. Meyer, et M. Middendorf. Springer Verlag, 2003.

[Doncieux, 1999] S. Doncieux. Évolution d'architectures de contrôle pour robots volants. Master's thesis, LIP6 - ENSEA, 1999.

[Durand *et al.*, 2000] N. Durand, J.-M. Alliot, et F. Medioni. Neural nets trained by genetic algorithms for collision avoidance. *Applied Artificial Intelligence*, 13(3), 2000.

[Duro *et al.*, 2002] R. J. Duro, J. Santos, et J. A. Becerra. *Biologically Inspired Robot Behavior Engineering*, chapitre Some Approaches for Reusing Behavior Based Robot Cognitive Architectures Obtained Through Evolution. Springer Verlag, 2002.

[Dzul *et al.*, 2002] A. Dzul, T. Hamel, et R. Lozano. Modeling and nonlinear control for a coaxial helicopter. In *IEEE 2002 International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, October 2002.

[Eggenberger, 2000] P. Eggenberger. Evolving neural network structures using axonal growth mechanisms. In *Proceedings of The Fifth Int. Symp. on Artificial Life and Robotics (AROB 5th'00)*, 2000.

[Elfes *et al.*, 1990] A. Elfes, S. S. Bueno, M. Bergerman, et J. G. J.Ramos. A semi-autonomous robotic airship for environmental monitoring mission. In *IEEE International Conference on Robotics and Automation*, pages 3449–3455, 1990.

[et al., 2003] éditeur Raidl G. et al. *Applications of Evolutionary Computing. EvoWorkshop 2003*, volume LNCS 2611. Springer, 2003.

[Fahlman et Lebiere, 1991] S. E. Fahlman et C. Lebiere. The cascade-correlation learning architecture. Rapport technique, School of Computer Science, Carnegie Mellon University, 1991.

[Filliat et Meyer, 2003] D. Filliat et J.-A. Meyer. Map-based navigation in mobile robots - i. a review of localization strategies. *Journal of Cognitive Systems Research*, 2003. In press.

[Floreano et Urzelai, 2001] D. Floreano et J. Urzelai. Evolution of plastic control networks. *Autonomous Robots*, 11 :311–317, 2001.

[Fogel et Walsh, 1966] Owens A. J. Fogel, L. J. et M. J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, 1966.

[Fogelman-Soulie et Gallinari, 1998] éditeurs F. Fogelman-Soulie et P. Gallinari. *Industrial Applications of Neural Networks*. World Scientific Publishing Co, 1998.

[Fonseca et Fleming, 1995] C. M. Fonseca et P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1) :1–16, 1995.

[Franceschini, 2002] N. Franceschini. From fly vision to robot vision : Re-construction as a mode of discovery. In *Sensors ans Sensing in Biology and Engineering*, éditeurs F. Barth, J. A. C. Humphrey, et Secomb T. Springer, Berlin, 2002.

[Friedrich et Moraga, 1997] C. M. Friedrich et C. Moraga. Using genetic engineering to find modular structures and activation functions for architectures of artificial neural networks. In *Computational Intelligence : Theory and Applications ; Proceedings of the 5th Fuzzy Day (LNCS 1226)*, Dortmund, Germany, 1997.

[Fullmer et Miikkulainen, 1991] B. Fullmer et R. Miikkulainen. Using marker-based genetic encoding of neural networks to evolve finite-state behaviour. In *Proceedings of the first European Conference on Artificial Life (ECAL-91)*, Paris, 1991.

[Gallinari, 1995] P. Gallinari. Modular neural net systems, training of. In *The Handbook of Brain Theory and Neural Networks*, éditeur M.A. Arbib. Bradford Books/MIT Press, 1995.

[Gallistel, 1980] C. R. Gallistel. *The Organization of Action : A New Synthesis.* Erlbaum, 1980.

[Garnier et Fraichard, 1997] P. Garnier et T. Fraichard. A fuzzy motion controller for a car-like vehicle. Rapport Technique 3200, INRIA, 1997.

[Gen et Cheng, 1997] éditeurs M. Gen et R. Cheng. *Genetic Algorithms and Engineering Design*. J. Wiley and Sons, Inc., Ashikaga Institute of Technology, Ashikaga, Japan, 1997.

[Geva et Sitte, 1993] S. Geva et J. Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems*, pages 40–51, 1993.

[Girard *et al.*, 2002] B. Girard, V. Cuzin, A. Guillot, K. Gurney, et T. Prescott. Comparing a bio-inspired robot action selection mechanism with winner-takes-all. In *From Animals to Animats 7. Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior*, éditeur Hallam et al. The MIT Press, 2002.

[Goldberg et Richardson, 1987] D. E. Goldberg et J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, éditeur J. J. Grefenstette, pages 148–154, San Francisco, 1987. CA : Morgan Kaufmann.

[Goldberg et Samtani, 1986] D. Goldberg et M. Samtani. Engineering optimization via genetic algorithm. In *Ninth Conference on Electronic Computation*, 1986.

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

[Gomes et Ramos, 1998] S. B. V. Gomes et J. J. G. Ramos. Airship dynamic modeling for autonomous operation. In *IEEE International Conference on Robotics and Automation*, pages 3462–3467, 1998.

[Gomez et Miikkulainen, 1998] F. Gomez et R. Miikkulainen. 2-d pole balancing with recurrent evolutionary networks. In *Proceedings of the International Conference on Neural Networks (ICANN 98*, 1998.

[Gomez et Miikkulainen, 2002] F. J. Gomez et R. Miikkulainen. Robust non-linear control through neuroevolution. Rapport Technique TR-AI-02-292, Department of Computer Sciences, The University of Texas at Austin, October 2002.

[Gomi et Griffith, 1996] T. Gomi et A. Griffith. Evolutionnary robotics – an overview. In *Proceedings of the IEEE Third International Conference on Evolutionnary Computation*, éditeur IEEE Press, 1996.

[Gourichon *et al.*, 2002] S. Gourichon, J.-A. Meyer, et P. Pirim. Using colored snapshots for short-range guidance in mobile robots. *International Journal of Robotics and Automation : Special issue on Biologically Inspired Robotics*, 17(4) :154–162, 2002.

[Gourichon *et al.*, 2003] S. Gourichon, J.-A. Meyer, S. H. Ieng, L. Smadja, et R. Benosman. Estimating ego-motion using a panoramic sensor : Comparison between a bio-inspired and a camera-calibrated method. In *AISB Symposium on Biologically-Inspired Machine Vision, Theory and Application*, 2003.

[Grefenstette et Schultz, 1994] J. Grefenstette et A. Schultz. An evolutionary approach to learning in robots. In *Machine Learning Workshop on Robot Learning*, 1994.

[Gruau *et al.*, 1996] F. Gruau, D. Whitley, et L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996 : Proceedings of the First Annual Conference*, éditeurs R. John Koza, E. David Goldberg, B. David Fogel, et L. Rick Riolo, pages 81–89, Stanford University, CA, USA, 28–31 1996. MIT Press.

[Gruau et Whitley, 1993] F. Gruau et D. Whitley. Adding learning to the cellular development of neural networks : Evolution and the baldwin effect. *Evolutionary Computation*, 1993.

[Gruau, 1992] F. Gruau. Cellular encoding of genetic neural networks. Rapport Technique 9221, Laboratoire de l'informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 1992.

[Gruau, 1994] F. Gruau. *Synthèse de Réseaux de Neurones par Codage Cellulaire et Algorithmes Génétiques*. PhD thesis, ENS Lyon, Université Lyon I, 1994.

[Gruau, 1995] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2) :151–183, 1995.

[Guccione *et al.*, 2000] S. Guccione, G. Muscato, G. Nunnari, G. S. Virk, A. K.M. Azad, A. Semerano, M. Ghrissi, T. White, et C. Glazebrook. Robots for volcanos : The state of the art. In *Proceedings 3rd Int Conf on Climbing and Walking Robots (CLAWAR 2000)*, pages 777–788, Madrid, Spain, 2000.

[Guessoum, 1998] Z. Guessoum. Dima : Une plate-forme multi-agents en smalltalk. *Revue Objet*, 3(4) :393–410, 1998.

[Guillot et Meyer, 2001] A. Guillot et J.-A. Meyer. The animat contribution to cognitive systems research. *Journal of Cognitive Systems Research*, 2(2) :157–165, 2001.

[Guillot et Meyer, 2003] A. Guillot et J.-A. Meyer. La contribution de l'approche animat aux sciences cognitives. *Cahiers Romans de Sciences Cognitives*, 1(1) :1–26, 2003.

[Gérard, 2002] P. Gérard. *Systèmes de classeurs : étude de l'apprentissage latent*. PhD thesis, Université Paris 6, 2002.

[Handley, 1995] S. Handley. Classifying nucleic acid sub-sequences as introns or exons using genetic programming. In *Proc Int Conf Intell Syst Mol Biol*, 1995.

[Happel et Murre, 1994] L. M. Happel, B. et M. J. Murre, J. The design and evolution of modular neural network architectures. *Neural Networks*, 7 :985–1004, 1994.

[Harvey, 1993] I. Harvey. *The Artificial Evolution of Adaptive Behavior*. PhD thesis, University of Sussex, 1993.

[Hebb, 1949] D. O. Hebb. *The organisation of behavior*. Wiley, New York, 1949.

[Hénaff *et al.*, 1994] P. Hénaff, M. Milgram, et J. Rabit. Adaptive neural control in mobile robotics : Experimentation for a wheeled cart. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, Texas, San Antonio, 1994.

[Hénaff et S., 1996] P. Hénaff et Delaplace S. Using backpropagation algorithm for neural adaptive control : Experimental validation on an industrial mobile robot. In *Proceedings of the 11th CISM-IFtoMM Symposium Theory and Practice of Robots and Manipulators (ROMANSY 11)*, 1996.

[Herwitz, 2000] S. R. Herwitz. Coffee harvest optimization using uav platforms for the acquisition of high resolution multispectral imagery. In *Proceedings of the NASA ERAST*, numéro NASA/CP-2000-209031, pages 49–54, Dryden Flight Research Center, 2000.

[Hinterding *et al.*, 1997] R. Hinterding, Z. Michalewicz, et A. E. Eiben. Adaptation in evolutionary computation : A survey. In *4th IEEE International Conference on Evolutionary Computation*, 1997.

[Hinton et Nowlan, 1987] G. E. Hinton et S. J. Nowlan. How learning guides evolution. *Complex Systems*, 1 :495–502, 1987.

[Hoffmann *et al.*, 1998] F. Hoffmann, T. J. Koo, et O. Shakernia. Evolutionary design of a helicopter autopilot. In *3rd On-Line World Conf. on Soft Computing (WSC3)*, 1998.

[Holland et Reitman, 1978] J. H. Holland et J. S. Reitman. *Pattern-Directed Inference Systems*, chapitre Cognitive Systems Based on Adaptive Algorithms. Academic Press, NY, 1978.

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MI : University of Michigan Press, 1975.

[Holland, 1986] J. H. Holland. Escaping brittleness : The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *Machine Learning : An Artificial Intelligence Approach*, 2, 1986.

[Holland, 2002] G. J. Holland. Tropical cyclone reconnaissance using the aerosonde uav. *World Meteorological Organization Bulletin*, 51(3), 2002.

[Hornby *et al.*, 2000] G. S. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, et M. Fujita. Evolving robust gaits with aibo. In *IEEE International Conference on Robotics and Automation*, pages 3040–3045, 2000.

[Hornik, 1993] K. Hornik. Some new results on neural network approximation. *Neural Networks*, (6) :1069–1072, 1993.

[Husken et Igel, 2002] M. Husken et C. Igel. Balancing learning and evolution. In *GECCO 2002*, éditeurs W. B. Langdon, E. Cantú-Paz, K. E. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, et N. Jonoska, pages 391–398. Morgan Kaufmann, 2002.

[Iida et Lambrinos, 2000] F. Iida et D. Lambrinos. Navigation in an autonomous flying robot by using a biologically inspired visual odometer. In *Sensor Fusion and Decentralized Control in Robotic System III, Photonics East, Proceeding of SPIE*, volume 4196, pages 86–97, 2000.

[Iida, 2001] F. Iida. Goal-directed navigation of an autonomous flying robot using biologically inspired cheap vision. In *Proceedings of the 32nd International Symposium on Robotics*, 2001.

[Ijspeert *et al.*, 1999] A. J. Ijspeert, J. Hallam, et D. Willshaw. Evolving swimming controllers for a simulated lamprey with inspiration from neurobiology. *Adaptive Behavior*, pages 151–172, 1999.

[Ijspeert et Kodjabachian, 1999] A. J. Ijspeert et J. Kodjabachian. Evolution and development of a central pattern generator for the swimming of a lamprey. *Artificial Life*, 1999. In Press.

[Jakobi, 1997] N. Jakobi. Evolutionary robotics and the radical envelop of noise hypothesis. *Adaptive Behavior*, 6(1) :131–174, 1997.

[Johnson *et al.*, 1996] E. N. Johnson, P. A. DeBitetto, C. A. Trott, et M. C. Bosse. The 1996 mit/boston university/draper laboratory autonomous helicopter system. In *15th DASC. AIAA/IEEE Digital Avionics Systems Conference (Cat. No.96CH35959). IEEE*, 1996.

[Kahn et Kannan, 2000] A. Kahn et S. Kannan. Georgia institute of technology, entry for the international aerial robotics competition hb-1. Rapport technique, Georgia Institute of Technology, 2000.

[Karr, 1991] C. L. Karr. Design of a cart-pole balancing fuzzy logic controller using a genetic algorithm. In *SPIE Conference on the Applications of Artificial Intelligence*, 1991.

[Keidel, 2000] B. Keidel. Design of a solar-powered hale aircraft for year-round operation at intermediate latitudes. In *Unmanned Vehicles (UV) for Aerial, Ground and Nval Military Operations. RTO meeting Proceedings 52*, 2000.

[Kelso *et al.*, 1986] S. Kelso, A. Ganong, et T. Brown. Hebbian synapses in hippocampus. In *Proceedings of the National Academy of Sciences USA*, volume 83, 1986.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. .Jr. Gelatt, et M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, 1983.

[Kitano, 1990a] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4 :461–476, 1990.

[Kitano, 1990b] H. Kitano. Empirical studies on the speed of the convergence of neural network training using genetic algorithms. In *Proceedings of the Eighth Ntat'l Conf on AI (AAAI-90)*, pages 789–795. MIT Press, Cambridge, MA, 1990.

[Knerr *et al.*, 1992] S. Knerr, L. Personnaz, et G. Dreyfus. Handwritten digit recognition by neural networks with single-layer training. *IEEE Transactions on Neural Networks*, 3(962), 1992.

[Kodjabachian et Meyer, 1995] J. Kodjabachian et J.-A. Meyer. Evolution and development of control architectures in animats. *Robotics and Autonomous Systems*, 16 :161–182, 1995.

[Kodjabachian et Meyer, 1997] J. Kodjabachian et J.-A. Meyer. Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9 :796–812, 1997.

[Kodjabachian et Meyer, 1998] J. Kodjabachian et J.-A. Meyer. Evolution and development of modular control architectures for 1-d locomotion in six-legged animats. *Connection Science*, 10 :211–237, 1998.

[Kodjabachian, 1998] J. Kodjabachian. *Développement et évolution de réseaux de neurones artificiels*. PhD thesis, Université Pierre et Marie Curie, 1998.

[Koza *et al.*, 1999] J. R. Koza, D. Andre, F. H. Bennett III, et M. Keane. *Genetic Programming 3 : Darwinian Invention and Problem Solving*. Morgan Kaufman, 1999.

[Koza *et al.*, 2003] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, et G. Lanza. *Genetic Programming IV : Routine Human-Competitive Machine Intelligence*. Kluwer Academic, 2003.

[Koza, 1992] J. R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Koza, 1994] J. R. Koza. *Genetic Programming II : Automatic Discovery of reusable Programs*. MIT Press, 1994.

[Kungl *et al.*, 2001] P. Kungl, M. Schlenker, et B. Krplin. Research and testing activities with the solar powered airship lotte within the scope of the airship research group at the university of stuttgart. In *AIAA 14th Lighter Than Air Convention and Exhibition*, Akron, USA, 2001.

[Lacroix *et al.*, 2002] S. Lacroix, K. I-Jung, P. Soueres, E. Hygounenc, et J-P. Berry. The autonomous blimp project of laas/cnrs : Current status and research challenges. 2002.

[Landau *et al.*, 2001a] S. Landau, S. Doncieux, A. Drogoul, et J.-A. Meyer. SFERES un framework pour la conception de systèmes multi-agents adaptatifs. Rapport technique, Université Paris IV, 2001.

[Landau *et al.*, 2001b] S. Landau, S. Picault, et A. Drogoul. ATNoSFERES : a Model for Evolutive Agent Behaviors. In *Proc. of AISB'01 symposium on Adaptive Agents and Multi-agent +systems*, pages 95–99, march 2001. ISBN 1 902956 17 0.

[Landau *et al.*, 2002] S. Landau, S. Doncieux, A. Drogoul, et J.-A. Meyer. Sferes : un framework pour la conception de systèmes multi-agents adaptatifs. *Technique et Science Informatique*, 21(4), 2002.

[Lapedes et Farber, 1987] A. Lapedes et R. Farber. *Neural Information Processing Systems*, chapitre How neural nets work. New York : American Institute of Physics, 1987.

[Lara *et al.*, 2001] B. Lara, M. Hulse, et F. Pasemann. Evolving neuro-modules and their interfaces to control autonomous robots. In *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI2001*, 2001.

[LeCun *et al.*, 1998a] Y. LeCun, L. Bottou, Y. Bengio, et P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[LeCun *et al.*, 1998b] Y. LeCun, L. Bottou, G. Orr, et K. Muller. Efficient backprop. In *Neural Networks : Tricks of the trade*, éditeurs G. Orr et Muller K. Springer, 1998.

[LeCun, 1987] Y. LeCun. *Modeles connexionnistes de l'apprentissage (connectionist learning models)*. PhD thesis, Université P. et M. Curie (Paris 6), 1987.

[LeFablec et Alliot, 1999] Y. LeFablec et J.-M. Alliot. Using neural networks to predict aircraft trajectories. In *Proceedings of the International Conference on Artificial Intelligence (ICAI'99)*, Las Vegas, 1999.

[Lindenmayer, 1968] A. Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of theoretical biology*, (18) :280–315, 1968.

[Luke et Spector, 1996] S. Luke et L. Spector. Evolving graphs and networks with edge encoding : Preliminary report. In *Late Breaking Papers of the Genetic Programming 96 (GP96)*, 1996.

[Mamdani, 1974] E. H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plants. In *Institution of Electrical Engineers*, 1974.

[Maris et te Boekhorst, 1996] M. Maris et R. te Boekhorst. Exploiting physical constraints : heap formation through behavioral error in a group of robots. In *Proceedings of IROS'96, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1655–1660, 1996.

[Mayley, 1996] G. Mayley. the evolutionary cost of learning. In *From Animals to Animats 4, Proceedings of the International Conference on Simulation of Adaptive Behavior*, éditeurs P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, et S. Wilson. MIT Press, 1996.

[Mayley, 1997] G. Mayley. Guiding or hiding : Explorations into the effects of learning on the rate of evolution. In *Fourth European Conference on Artificial Life*, éditeurs Husbands et Harvey. The MIT Press, 1997.

[McCulloch et Pitts, 1943] W. S. McCulloch et W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, (5) :115–133, 1943.

[Mel, 1990] B. W. Mel. The sigma-pi model neuron : Roles of the dendritic tree in associative learning. *Soc. Neuroscience Abstr.*, 16, 1990.

[Metropolis et Ulam, 1949] N. Metropolis et S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247) :335–341, 1949.

[Meyer *et al.*, 1998] J.-A. Meyer, P. Husbands, et I. Harvey. Evolutionary robotics : a survey of applications and problems. In *Proceedings of The First European Workshop on Evolutionary Robotics - EvoRobot98*, éditeurs P. Husbands et J.-A. Meyer. Springer Verlag, 1998.

[Meyer *et al.*, 2002] J.-A. Meyer, S. Doncieux, D. Filliat, et A. Guillot. *Biologically Inspired Robot Behavior Engineering*, chapitre Evolutionary Approaches to Neural Control of Rolling, Walking, Swimming and Flying Animats or Robots. Springer Verlag, 2002.

[Meyer et Filliat, 2003] J.-A. Meyer et D. Filliat. Map-based navigation in mobile robots - ii. a review of map-learning and path-planning strategies. *Journal of Cognitive Systems Research*, 2003. In Press.

[Meyer et Guillot, 2002] J.-A. Meyer et A. Guillot. Vers une robotique animale. *Pour la Science*, (300) :168–171, 2002.

[Meyer, 1997a] J.-A. Meyer. Biomimetic mechanisms of self-organization and adaptation in animats. In *Proceedings of the International Symposium on System Life. Japan Society of Mechanical Engineering*, éditeurs F. Hara et K. Yoshida, 1997.

[Meyer, 1997b] J.-A. Meyer. From natural to artificial life : Biomimetic mechanisms in animat designs. *Robotics and Autonomous Systems*, 22 :3–21, 1997.

[Meyer, 1998] J.-A. Meyer. Evolutionary approaches to neural control in mobile robots. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, 1998.

[Michalewicz *et al.*, 1996] Z. Michalewicz, D. Dasgupta, R. G LeRiche, et M. Shoenauer. Evolutionary algorithms for constrained engineering problems. *Computers and Industrial Engineering Journal*, 1996.

[Michel *et al.*, 1997] O. Michel, M. Clergue, et P. Collard. Artificial neurogenesis : Applications to the cart-pole problem and to an autonomous mobile robot. *International Journal on Artificial Intelligence Tools*, 1997.

[Michel, 1999] O. Michel. Evolutionary neurogenesis applied to mobile robotics. In *Advances In Evolutionary Synthesis of Neural Systems*, éditeurs M. Patel et V. Honavar. Cambridge, MA : MIT Press, 1999.

[Michelson et Reece, 1998] R. C. Michelson et S. Reece. Update on flapping wing micro air vehicle research : Ongoing work. In *13th Bristol International RPV Conference*, 1998.

[Miller *et al.*, 1989] G. F. Miller, P. M. Todd, et S. U. Hedge. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, 1989.

[Mitchell et Belew, 1996] éditeurs M. Mitchell et R. Belew. *Adaptive Individuals In Evolving Population Models*. Addison-Wesley, 1996.

[Montana et Davis, 1989] D. Montana et L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, 1989.

[Moriarty et Miikkulainen, 1997] D. E. Moriarty et R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4) :373–399, 1997.

[Musliner, 1993] D. J. Musliner. *CIRCA : The Cooperative Intelligent Real-Time Control Architecture*. PhD thesis, The University of Michigan, 1993.

[Naudin, 2002] J. L. Naudin. Propulsion électrocinétique : le projet arda/lifter. In *Colloque Microdrone 2002*, ENSICA, Toulouse, 2002.

[Nehmzow, 2002] U. Nehmzow. Physically embedded ga learning in multi-robot scenarios : The pega algorithm. In *Proceedings of the Second International Workshop on Epigenetic Robotics*, éditeurs C. G. Prince, Y. Demiris, Y. Marom, H. Kozima, et C. Balkenius, Edinburgh, Scotland, 2002.

[Nerrand *et al.*, 1993] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, et S. Marcos. Neural networks and non-linear adaptive filtering : Unifying concepts and new algorithms. *Neural Computation*, 5 :165–197, 1993.

[Netter et Franceschini, 1999] T. Netter et N. Franceschini. Neuromorphic optical flow sensing for nap-of-the-earth flight. In *Mobile Robots XIV*, volume 3838 de *SPIE*, pages 208–216, Bellingham, USA, 1999.

[Nolfi *et al.*, 1994] S. Nolfi, J. L. Elman, et D. Parisi. Learning and evolution in neural networks. Rapport technique, Institute of psychology. C.N.R.- Rome, 1994.

[Nolfi et Floreano, 2001] S. Nolfi et D. Floreano. *Evolutionary Robotics : Biology, Intelligence and Technology of Self-organizing Machines.* The MIT Press, 2001.

[Nolfi et Marocco, 2002] S. Nolfi et D. Marocco. Active perception : A sensorimotor account of object categorization. In *Proceedings of the seventh International Conference on Simulation of Adaptive Behavior (SAB 2002)*, éditeurs B. Hallam, D. Floreano, J. Hallam, G. Hayes, et Meyer J.-A., 2002.

[Nolfi, 1997] S. Nolfi. Using emergent modularity to develop control systems for mobile robots. *Adaptive Behavior*, 5(3/4) :343–363, 1997.

[Nordin et Banzhaf, 1997] P. Nordin et W. Banzhaf. An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adaptive Behaviour*, 1997.

[Notredame *et al.*, 1997] C. Notredame, E. O'brien, et D. Higgins. Raga : Rna sequence alignment by genetic algorithm. In *Nucleic Acids Res*, volume 25, pages 4570–80, 1997.

[Pasemann *et al.*, 2000] F. Pasemann, U. Steinmetz, et M. Hulse. Robot control an the evolution of modular neurodynamics. In *Workshop on Evolution and Neural Control of Autonomous Systems*, 2000.

[Pasemann et Dieckmann, 1997a] F. Pasemann et U. Dieckmann. Balancing rotators with evolved neurocontrollers, 1997. preprint n°37.

[Pasemann et Dieckmann, 1997b] F. Pasemann et U. Dieckmann. Evolved neurocontrollers for pole-balancing. In *Biological and Artificial Computation : From Neuroscience to Technology. IWANN'97*, 1997.

[Pasemann, 1997] F. Pasemann. Pole-balancing with different evolved neurocontrollers. In *ICANN'97 - International Conference on Artificial Neural Networks*, 1997.

[Pearlmutter, 1995] B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks : a survey. *IEEE Transactions on Neural Networks*, 1995. à vérifier (draft).

[Peter Angeline *et al.*, 1994] J. Peter Angeline, M. Gregory Saunders, et P. Jordan Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1) :54–65, January 1994.

[Pomerleau, 1991] D. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3 :88–97, 1991.

[Pujol et Poli, 1997] C. F. Pujol, J. et R. Poli. Evolving neural controllers using a dual representation. Rapport Technique CSRP-97-25, School of Computer Science, University of Birmingham, september 1997.

[Radcliffe, 1990] N. J. Radcliffe. *Genetic Neural Networks on MIMD Computers*. PhD thesis, Dept. of Theoretical Phys., University of Edinburgh, UK, 1990.

[Radcliffe, 1993] N. J. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural computing and applications*, 1(1) :67–90, 1993.

[Riedmiller, 1994] M. Riedmiller. Aspects of learning neural control. In *IEEE International Conference on System, Man and Cybernetics*, October 1994.

[Rivals et Personnaz, 2000] I. Rivals et L. Personnaz. A statistical procedure for determining the optimal number of hidden neurons of a neural model. In *Second International ICSC Symposium on Neural Computation (NC'2000)*, Berlin, Germany, 2000.

[Rivals, 1995] I. Rivals. *Modélisation et commande de processus par réseaux de neurones ; application au pilotage d'un véhicule autonome*. PhD thesis, Université Paris VI, 1995.

[Romeo *et al.*, 2000] G. Romeo, G. Frulla, et L. Fattore. Heliplat : High altitude very-long endurance solar powered uav for telecommunication applications. fem analysis, manufacturing and testing of 21m long cfrp wing box. In *Unmanned Vehicles (UV) for Aerial, Ground and Nval Military Operations. RTO meeting Proceedings 52*, 2000.

[Ronco et Gawthrop, 1997] E. Ronco et P. J. Gawthrop. Neural networks for modelling and control. Rapport technique, 1997.

[Rosenblatt, 1965] F. Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, pages 386–408, 1965.

[Rueckl *et al.*, 1989] J. G. Rueckl, K. R. Cave, et S. M. Kosslyn. Why are 'what' and 'where' processed by separate cortical visual systems ? a computational investigation. *Journal of Cognitive Neuroscience*, 1 :171–186, 1989.

[Rumelhart et McClelland, 1986] D. E. Rumelhart et J. L. McClelland. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*. The MIT Press/Bradford Books, 1986.

[Salustowicz, 1995] R. Salustowicz. A genetic algorithm for the topological optimization of neural networks. Master's thesis, Technische Universität Berlin, 1995.

[Sasaki et Tokoro, 1997] T. Sasaki et M. Tokoro. Adaptation toward changing environments : Why darwinian in nature ? In *Fourth European Conference on Artificial Life*, éditeurs Phil Husbands et Inman Harvey, page ?, Cambridge, MA, 1997. MIT Press.

[Schultz et Grefenstette, 2000] A. C. Schultz et J. J. Grefenstette. Continuous and embedded learning in autonomous vehicles : Adapting to sensor failures. In *SPIE Int. Symposium on Aerospace/Defense Sensing, Simulation and Controls (AeroSense 2000)*, 2000.

[Schwefel et Bäck, 1995] H. P. Schwefel et T. Bäck. Evolution strategies ii : Theoretical aspects. In *Genetic Algorithms iEngineering and Computer Science, Proc. First Short Course EUROGEN-95*. 1995.

[Schwefel et Rudolph, 1995] H.-P. Schwefel et G. Rudolph. Contemporary evolution strategies. Berlin Springer, 1995.

[Schwefel, 1977] H.-P. Schwefel. *Numerische Optimierung von ComputerModellen mittels der Evolutionsstrategie.* Birkhäuser, 1977.

[Schwefel, 1981] H. P. Schwefel. *Numerical optimization of computer models.* Wiley, New-York, 1981.

[Sharkey, 1996] A. J. C. Sharkey. On combining artificial neural nets. *Connection Science*, 8(3/4) :299–314, 1996.

[Sharkey, 1999] éditeur A. J. C. Sharkey. *Combining Artificial Neural Nets : Ensemble and Modular Multi-Net Systems.* Springer-Verlag, 1999.

[Shim *et al.*, 1998] H. Shim, T. J. Koo, F. Hoffmann, et S. Sastry. A comprehensive study of control design for an autonomous helicopter. In *IEEE conference on Decision and Control CDC'98 (submitted)*, 1998.

[Sigaud et Flacher, 2002] O. Sigaud et F. Flacher. Vers une approche dynamique de la sélection de l'action. In *Approche dynamique de la cognition artificielle*, éditeurs A. Guillot et E. Daucé, pages 163–178. Hermès, 2002.

[Sims, 1994] K. Sims. Evolving 3d morphology and behavior by competition. In *Proceedings of the Fourth International Workshop on Artificial Life*, éditeurs R. A. Brooks et P. Maes. The MIT Press, 1994.

[Singer, 1987] W. Singer. Activity-dependant self-organisation of synaptic connections as a substrate of learning. In *The Neural and Molecular Bases of Learning*, éditeurs J. P. Changeux et M. Konishi. Wiley, London, UK, 1987.

[Smith, 1980] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[Srinivasan *et al.*, 1998] M. V. Srinivasan, J. S. Chahl, K. Weber, S. Venkatesh, M. G. Nagle, et S. W. Zhang. Robot navigation inspired by principles of insect vision. In *Field and Service Robotics*, éditeur A. Zelinsky, pages 12–16, New York, 1998. Springer Verlag, Berlin.

[Stanley et Miikkulainen, 2001] K. O. Stanley et R. Miikkulainen. Evolving neural networks through augmenting topologies. Rapport Technique TR-AI-01-290, University of Texas at Austin, June 2001.

[Stanley et Miikkulainen, 2002a] K. O. Stanley et R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, 2002. Morgan Kaufmann.

[Stanley et Miikkulainen, 2002b] K. O. Stanley et R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2) :99–127, 2002.

[Stanton et Sejnowski, 1989] P. K. Stanton et T. J. Sejnowski. Associative long-term depression in the hippocampus induced by hebbian covariance. *Nature*, 339 :215–218, 1989.

[Stent, 1973] G. Stent. A physiological mechanism for hebb's postulate of learning. In *Proceedings of the National Academy of Science USA*, volume 70, 1973.

[Stephens *et al.*, 2000] G. L. Stephens, R. G. Ellingson, J. Vitko Jr, et W. Bolton. The department of energy's atmospheric radiation measurement (arm) unmanned aerospace vehicle (uav) program. *Bull. Am. Meteor. Soc.*, 81 :2915–2937, 2000.

[Sutton et Barto, 1998] R. S. Sutton et A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[Sutton, 1988] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3 :9–44, 1988.

[Takagi et Sugeno, 1985] T. Takagi et M. Sugeno. Fuzzy identification of systems and its application to modelling and control. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 15. 1985.

[Thompson *et al.*, 1996] A. Thompson, I. Harvey, et P. Husbands. Unconstrained evolution and hard consequences. In *Towards Evolvable Hardware : The evolutionary engineering approach*, éditeurs E. Sanchez et M. Tomassini, volume 1062 de *LNCS*, pages 136–165. Springer-Verlag, 1996.

[Tyrrell, 1993] T. Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, Centre for Cognitive Science, 1993.

[Urzelai *et al.*, 1998] J. Urzelai, D. Floreano, M. Dorigo, et M. Colombetti. Incremental robot shaping. *Connection Science Journal*, 10(384) :341–360, 1998.

[Vaario *et al.*, 1997] J. Vaario, A. Onitsuka, et K. Shimohara. Formation of neural structures. In *Proceedings of the Fourth European Conference on Articial Life, ECAL97*. The MIT Press, 1997.

[van der Zwaan *et al.*, 2000] S. van der Zwaan, Matteo Perrone, Alexandre Bernardino, et José Santos-Victor. Control of an aerial blimp based on visual input. In *8th International Symposium on Intelligent Robotic Systems - SIRS200 -*, Reading, UK, July 2000.

[Van Veldhuizen et Lamont, 1998] D. A. Van Veldhuizen et G. B. Lamont. Evolutionary computation and convergence to a pareto front. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, éditeur John R. Koza, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Stanford University Bookstore.

[Viollet et Franceschini, 2001] S. Viollet et N. Franceschini. Super-accurate visual control of an aerial minirobot. *Autonomous minirobots for Research and Edutainment*, pages 215–224, 2001.

[Wall, 2000] M. Wall. Galib. http ://lancet.mit.edu/ga/, 2000.

[Watson et Wiles, 2002] J. Watson et J. Wiles. The rise and fall of learning : A neural network model of the genetic assimilation of acquired traits. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, pages 600–605, 2002.

[Werbos, 1974] P. J. Werbos. *Beyond Regression : New Tools for Prediction and Analysis in the Behavior Sciences*. PhD thesis, Harvard University, Committee on Applied Mathematics, 1974.

[Whitley *et al.*, 1990] D. Whitley, T. Starkweather, et C. Bogart. Genetic algorithm and neural networks : Optimizing connexions and connectivity. *Parallel Computing*, pages 347–361, 1990.

[Whitley *et al.*, 1994] D. L. Whitley, V. S. Gordon, et K. E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *Parallel Problem Solving from Nature – PPSN III*, éditeurs Yuval Davidor, Hans-Paul Schwefel, et Reinhard Männer, pages 6–15, Berlin, 1994. Springer.

[Whitley *et al.*, 1995] D. Whitley, F. Gruau, et L Pyeatt. Cellular encoding applied to neurocontrol. In *Siwth International Conference on Genetic Algorithms*, 1995.

[Whitley et Hanson, 1989] D. Whitley et T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, éditeur J. D. Shaffer, pages 391–396. Morgan Kaufmann, 1989.

[Whitley, 1989] D. Whitley. Applying genetic algorithms to neuralnetwork learning. In *Proceedings of the 7th Conference for the Study of Artificial Intelligence and Adaptive Behavior*, 1989.

[Wieland, 1991] A. Wieland. Evolving neural network controllers for unstable systems. In *International Joint Conference on Neural Networks*, 1991.

[Willshaw et Dayan, 1990] D. Willshaw et P. Dayan. Optimal plasticity from matrix memories : Wwhat goes up must come down. *Neural Computation*, pages 85–93, 1990.

[Wong, 2001] K. C. Wong. Uav design activities in a university environment. In *9th Australian International Aerospace Congress*, Canberra, Australia, 2001.

[Wu *et al.*, 1999] A. Wu, A. Schultz, et A. Agah. Evolving control for distributed micro air vehicles. In *proc. of the IEEE 1999 Conference on Computational Intelligence in Robotics and Automation (CIRA-99)*, 1999.

[Wyeth et Barron, 1998] G. F. Wyeth et I. Barron. An autonomous blimp. *Field and Service Robotics*, pages 464–470, 1998.

[Yan *et al.*, 2001] J. Yan, R. Wood, S. Avandhanula, M. Sitti, et R. S. Fearing. Towards flapping wing control for a micromechanical flying insect. In *IEEE Int. Conf. Robotics and Automation*, 2001.

[Yao et Liu, 1997] X. Yao et Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3) :694–713, 1997.

[Yao, 1999] X. Yao. Evolving artificial neural networks. *PIEEE : Proceedings of the IEEE*, 87, 1999.

[Zadeh, 1965] L. Zadeh. Fuzzy sets. *Journal of Information and Control*, 8 :338–353, 1965.

[Zalzala, 1996] A. M. S. Zalzala. *Neural Networks for Robotic Control : Theory and Applications*. Ellis Horwood, 1996.

[Zhang *et al.*, 1997] B.-T. Zhang, P. Ohm, et H. Mühlenbein. Evolutionary induction of sparse neural trees. *Evolutionary Computation*, 5(2) :213–236, 1997.

[Zhang et Mühlenbein, 1993] B. T. Zhang et H. Mühlenbein. Genetic programming if minimal neural nets using occam's razor. In *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA-93)*, éditeur S. Forrest, pages 342–349. Morgan Kaufmann, 1993.

[Zhang et Ostrowski, 1999] H. Zhang et J. P. Ostrowski. Visual servoing with dynamics : Control of an unmanned blimp. In *proceeding of 1999 International Conference of Robotics and Automation*, 1999.

[Zufferey *et al.*, 2002] J. C. Zufferey, D. Floreano, M. van Leeuwen, et T. Merenda. Evolving vision-based flying robots. In *Proceedings of the 2nd International Workshop on Biologically Motivated Computer Vision LNCS*, éditeurs Bülthoff, Lee, Poggio, et Wallraven, 2002.