Évolution d'architectures de contrôle pour robots volants

Stéphane Doncieux

Equipe AnimatLab
Laboratoire d'Informatique de Paris 6 (LIP6)
8, rue du Capitaine Scott
75015 Paris, France.
E-mail :stephane.doncieux@poleia.lip6.fr

RÉSUMÉ. *Ce projet a pour but d'obtenir des architectures de type réseaux de neurones qui doivent contrôler des robots volants, tels que des dirigeables. La méthode employée ici est une méthode évolutionniste. L'objectif est que les architectures obtenues soient adaptées à un environnement dont les caractéristiques peuvent changer. Dans le cas du dirigeable, nous nous sommes intéressés à un environnement soumis à un vent quelconque variable dans le temps.*

ABSTRACT. *A evolutionnary approach is used to design neural control architectures for flying robots like blimps. The goal is to obtain control architectures adapted to environments whose features may change. In the case of the blimp, we have considered an environment with a variable wind.*

MOTS-CLÉS: *Évolution, Asservissement de Vitesse, Dirigeable, Réseaux de Neurones Artificiels, SGOCE.*

KEY WORDS: *Evolution, Speed Control, Blimp, Artificial Neural Network, SGOCE.*

1. Introduction

Les robots conçus selon une approche traditionnelle de l'Intelligence Artificielle utilisent généralement des informations de haut niveau, données par le concepteur. L'Intelligence Située [TSC 99, CLA 97] est une autre approche de la robotique : le robot ne dispose que des informations fournies par ses capteurs et doit être autonome. Dans l'optique d'applications réelles, l'environnement du robot peut être très complexe. Concevoir l'architecture de contrôle d'un tel robot est une tâche délicate parce que le robot peut être confronté à de nombreux problèmes parfois difficilement prévisibles, problèmes qui doivent cependant tous être résolus par le concepteur pour que le robot accomplisse efficacement sa tâche. Pour faire face à ce type de difficulté, de nombreux chercheurs se sont intéressés à la conception automatique d'architectures de contrôle [KOZ 92]. Le robot est placé dans son environnement et son architecture est conçue progressivement en fonction des problèmes rencontrés. Une solution est d'utiliser une approche inspirée de l'évolution naturelle [GOM 96]. On considère une population de robots dont on note les performances. Les individus les mieux notés se reproduisent et génèrent ainsi de nouveaux individus potentiellement plus performants. En utilisant de telles approches, des robots insectes ont appris à marcher [KOD 98b], des robots lamproies ont appris à nager [IJS 99]. Pour notre part, nous avons travaillé sur des robots volants.

Le milieu aérien est un environnement très dynamique dans lequel un robot doit être capable de corriger les perturbations auxquelles il pourrait être soumis : vent, trous d'air... L'objectif du travail décrit ici est d'obtenir automatiquement des architectures de contrôle capables de corriger ces perturbations. L'application concerne les dirigeables, aéronefs très stables mais dont le grand volume est un handicap important en cas de vent. La tâche que l'on cherche à résoudre est de contrer l'effet du vent et pour ce faire les robots sont contrôlés par des réseaux de neurones artificiels. L'évolution porte sur le programme de développement de ces réseaux [GRU 94].

Dans cet article, nous allons tout d'abord décrire le paradigme SGOCE que nous avons utilisé ici. Nous nous intéresserons ensuite aux applications que nous avons réalisées sur le dirigeable, avant de conclure et d'annoncer les axes des recherches à venir.

2. Le paradigme SGOCE

Le paradigme SGOCE [KOD 98a, KOD 97, KOD 98b] - Simple Geometry Oriented Cellular Encoding - permet de faire évoluer le programme de développement d'un

réseau de neurones dont l'architecture peut être arbitrairement complexe. Dans ce paradigme, chaque cellule du réseau est située dans un substrat à deux dimensions. Initialement, le substrat ne contient que les neurones capteurs ou effecteurs et des cellules de développement, tous fixés par le concepteur. Chaque cellule de développement va exécuter un programme pour donner naissance à un réseau neuronal, l'ensemble formant le réseau qui constitue l'architecture de contrôle proprement dite (figure 1). Cette architecture est utilisée pour contrôler un robot dont le comportement est noté par une fonction d'évaluation souvent appelée fitness (même dans les publications française). Les notes ainsi obtenues permettront de faire une sélection lors de la génération de nouveaux individus.

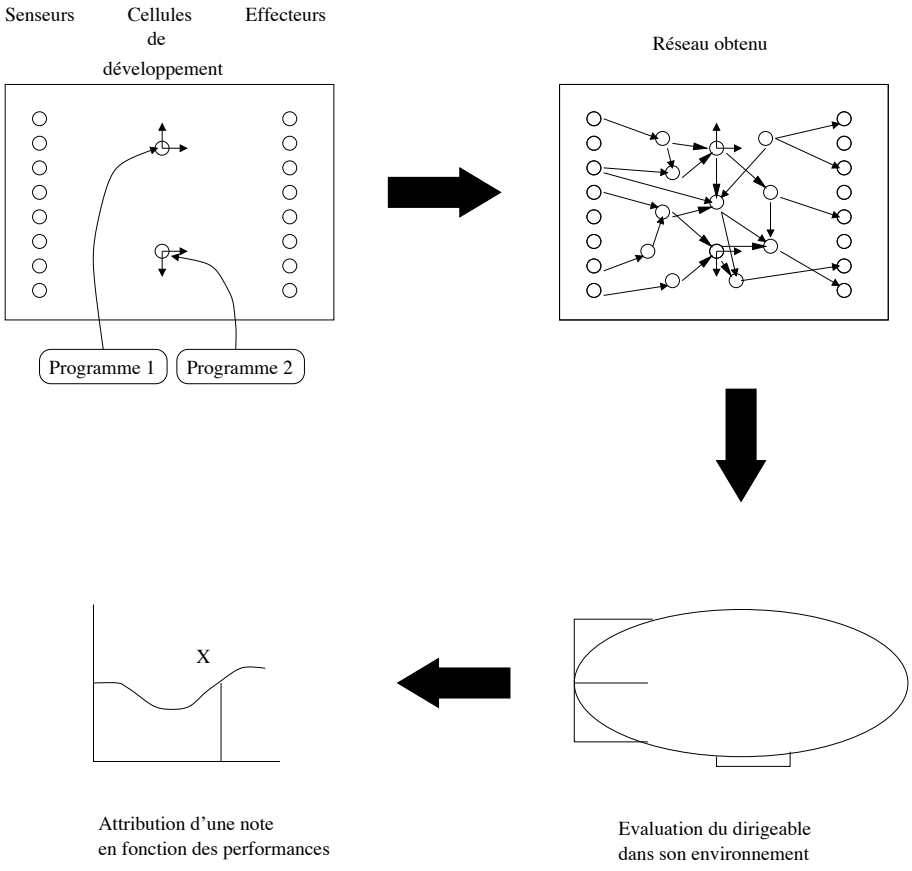

Figure 1. *Principe de SGOCE. Le programme de développement d'un individu est exécuté pour obtenir le réseau de neurone correspondant. Ce réseau est utilisé pour contrôler un dirigeable simulé et une note de fitness est attribuée au comportement résultant*

2.1. *Codage des réseaux*

Le programme de développement codant les réseaux a une forme arborescente. Chaque noeud correspond à une instruction de développement.

Les instructions peuvent être d'arité 0, 1 ou 2 selon leur type. Les instructions d'arité 2 sont les instructions de division : après division, une nouvelle cellule de développement apparaît : elle exécute alors le sous-arbre droit du noeud 'DIVISION' alors que la cellule 'mère' exécute le sous arbre gauche (figure 2). Les instructions permettent donc à une cellule de se diviser, elles permettent aussi de créer des connections ou de fixer des valeurs de paramètres :

DIVIDE α r : création d'une cellule de coordonnées polaire α r

GROW α r ω : création d'une connexion sortante de poids ω vers la zone de coord. polaires α r

DRAW α r ω : création d'une connexion entrante de poids ω vers la zone de coord. polaires α r

SETBIAS β : modification du biais

SETTAU τ : modification de la constante de temps

DIE : mort de la cellule

Chaque cellule de développement est munie d'un repère qui lui est propre. L'orientation de ces repères peut changer d'une cellule à l'autre pour mieux exploiter d'éventuelles propriétés de symétrie. α et *r* sont les coordonnées polaires du point visé dans le repère de la cellule exécutant l'instruction. Ainsi 'DIVIDE α r' crée une cellule identique à la cellule exécutant cette instruction, à la distance 'r' de la cellule mère et faisant un angle α avec l'axe des abscisses associé à la cellule. GROW (DRAW) crée une connexion sortante (entrante) vers la zone de l'espace centrée sur le point de coordonnée α r. La connexion se fera avec le neurone le plus proche du point de coordonnées α r (qui peut être le neurone initiateur, ce qui crée alors une connexion récurrente).

La définition aléatoire et sans restriction d'un programme utilisant ces instructions est peu efficace, car de nombreuses instructions seront sans effet sur le réseau final. Si un programme contient, par exemple, plusieurs instructions SETBIAS ou SETTAU à la suite, seule la dernière sera utile, les autres seront sans effet. Un moyen simple de restreindre l'espace de recherche est donc de contraindre les programmes à être conformes à une grammaire définie par le programmeur.

Par exemple, dans la grammaire de la figure 3, le nombre de neurones est limitée à 8 par programme de développement (3 divisions successives au maximum) et les neurones peuvent établir au plus 4 connexions. Les neurones utilisent soit des paramètres donnés dans le programme de développement (SETBIAS, SETTAU), soit les valeurs par défaut de ces paramètres (DEFBIAS, DEFTAU). Les instructions NOLINK et DIE permettent d'obtenir des réseaux plus simples que ceux que la grammaire autorise : ici

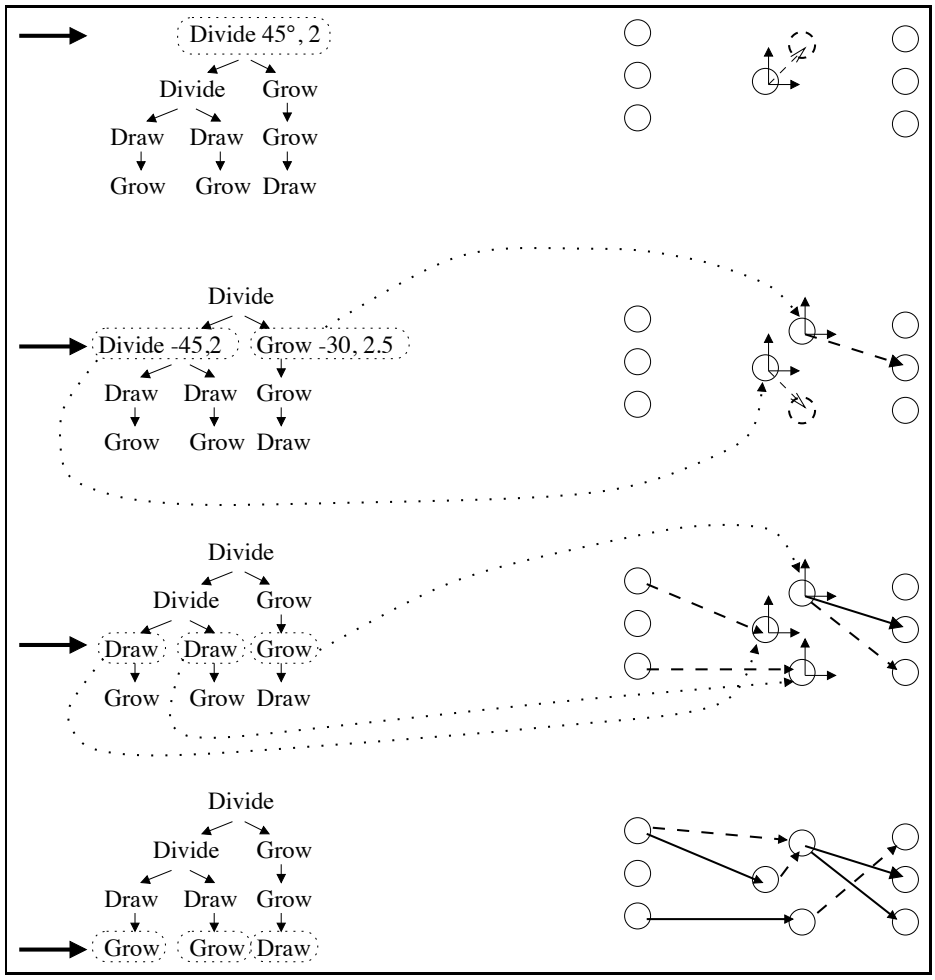

Figure 2. *Exemple de programme. Les instructions à la même profondeur dans l'arbre du programme sont exécutées simultanément. Ce programme crée une réseau de trois neurones intermédiaires partant d'une cellule initiale*

cela permet d'obtenir des réseaux comprenant moins de huit neurones et des neurones présentant moins de quatre connexions.

Les neurones sont des intégrateurs à fuite, ce qui implique que chaque neurone est caractérisé par d'un potentiel de membrane m_i qui varie selon l'équation :

$$\tau_i \frac{dm_i}{dt} = -m_i + \sum_j \omega_{i,j} x_j + I_i$$

Symboles terminaux DIVIDE, GROW, DRAW, SETBIAS, SETTAU, DIE, NOLINK, DEFBIAS, DEFTAU, PROGN3, PROGN4.

Variables Start1, Level1, Level2, Neuron, Bias, Tau, Connex, Link

Règles de production

$$
\begin{array}{lcl}
Start1 & \rightarrow & DIVIDE(Level1, Level1) \\
Level1 & \rightarrow & DIVIDE(Level2, Level2) \\
Level2 & \rightarrow & DIVIDE(Neuron, Neuron) \\
Neuron & \rightarrow & PROGN3(Bias, Tau, Connect) | DIE \\
Bias & \rightarrow & SETBIAS | DEFBIAS \\
Tau & \rightarrow & SETTAU | DEFTAU \\
Connex & \rightarrow & PROGN4(Link, Link, Link, Link) \\
Link & \rightarrow & GROW | DRAW | NOLINK
\end{array}
$$

Symbole de départ Start1

Figure 3. *Exemple de grammaire*

τ_i est la constante de temps

$\omega_{i,j}$ est le poids de la connexion reliant *i* à *j*

I_i est l'entrée que le neurone peut recevoir de l'extérieur

x_j est la sortie du neurone *j* calculée à partir de m_i par la formule :

$$x_j = \frac{1}{1 + e^{-(m_j - B_j)}}$$

B_j est le biais du neurone j. Ce type de neurone offre des possibilités dynamiques importantes et les réseaux générés forment une famille d'approximateurs dynamiques universels [BEE 95].

2.2. *Algorithme d'évolution*

Les programmes de développement générés par l'évolution ont une structure arborescente inspirée de la programmation génétique de Koza [KOZ 92]. Les opérateurs génétiques utilisés sont de deux types :

croisement : deux sous arbres compatibles[1] sont sélectionnés et échangés entre deux programmes. Cette opération se faisant avec la probabilité p_c

mutations : elles sont de deux types dans cette application. La première consiste à remplacer un sous-arbre du programme par un autre généré aléatoirement conformément à la grammaire utilisée, opération réalisée avec la probabilité p_m. La deuxième mutation est appliquée avec une probabilité de 1, elle consiste

[1] Deux sous-arbres compatibles sont des sous arbres de la même classe grammaticale.

à modifier un nombre aléatoire de paramètres. Le nombre de paramètres à modifier est tiré d'une distribution binomiale de paramètres n et p.

Dans les expériences réalisées ici, $p_c = 0.4, p_m = 0.4$.

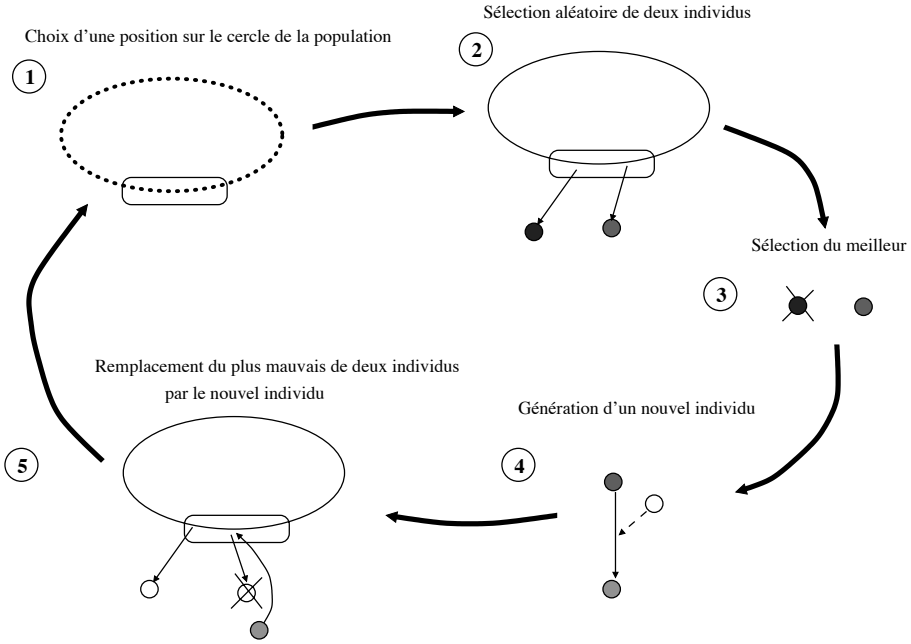

Figure 4. *Etapes de la génération d'un nouvel individu*

Un algorithme spécial a été employé (figure 4) pour éviter de rester bloqué sur un extrémum local. La population, constituée de cent individus, est située sur un cercle et les opérateurs génétiques sont utilisés localement. La génération d'un nouvel individu est effectuée de la façon suivante :

- une position P est choisie sur le cercle.
- deux individus appartenant au voisinage de P sont sélectionnés avec une probabilité diminuant avec la distance à P et nulle en dehors du voisinage.
- le meilleur des deux individus est sélectionné.
- les opérateurs génétiques sont appliqués à cet individu. Le deuxième individu nécessaire pour le croisement est choisi aléatoirement dans le voisinage.
- le nouvel individu est évalué.
- deux nouveaux individus sont choisis dans le voisinage de P.
- le plus mauvais est remplacé par le nouvel individu.

Le calcul de la performance d'un individu est effectué en trois étapes évoquées sur la figure 1 :

- expression du programme de développement.

– simulation du comportement de l'animat obtenu.
– calcul de la fitness de l'animat en fonction de son comportement en simulation.

On considère que l'on a changé de génération lorsque l'on a généré 100 nouveaux individu (100 étant la taille de la population).

2.3. *Méthode incrémentale*

Pour obtenir des comportements plus complexes, il devient intéressant d'utiliser une approche incrémentale qui accélère la convergence en guidant l'évolution. Le principe est le suivant : une architecture réalisant un comportement 'simple' est d'abord recherchée, puis le réseau obtenu est figé. Une nouvelle évolution est ensuite réalisée, le nouveau réseau ainsi généré étant susceptible de créer des connexions avec le précédent. Son comportement vient alors moduler celui du premier réseau pour obtenir le comportement désiré.

3. Application au dirigeable

3.1. *Problème choisi*

Un dirigeable est un grand réservoir d'hélium sous lequel on dispose une nacelle et des moteurs. Par sa conception, il est donc très stable et facile à piloter, mais il est sensible au vent.

Dans cette application, nous avons voulu obtenir un réseau capable de contrer l'effet d'un vent quelconque et de maintenir une vitesse constante. Ce réseau ne pourra fonctionner seul : il devra recevoir la valeur de la vitesse à maintenird'un module de plus haut niveau (module de navigation ou opérateur humain). Sa tâche est uniquement de maintenir la vitesse du dirigeable relative au sol à la valeur qui lui est fournie : c'est un asservissement de vitesse.

3.2. *Préparation de l'évolution*

Simulation

Le dirigeable, de masse $M = 7g$, a deux propulseurs de poids $M_p = 1g$, qui sont chacun à une distance $l = 10cm$ du centre de gravité (figure 5). La force maximale qu'ils peuvent fournir est de $15N$. Dans cette simulation, le dirigeable est considéré

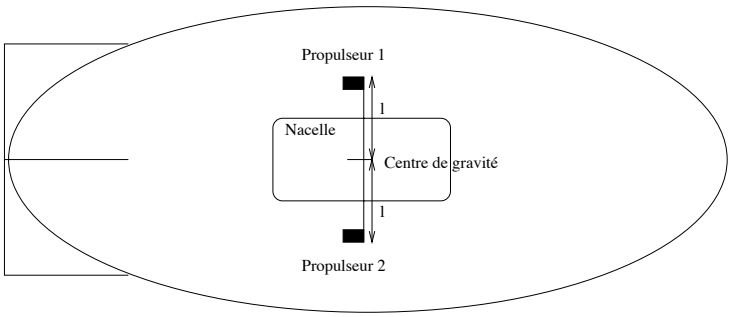

Figure 5. *Dirigeable vu de dessous*

comme ponctuel, il reste à altitude constante et il se déplace dans un plan infini. La simulation utilise les équations suivantes :

$$\begin{cases} M\ddot{x} + D_v\dot{x} + F_x = 0 \\ M\ddot{y} + D_v\dot{y} + F_y = 0 \\ I\ddot{\theta} + D_w\dot{\theta} + \tau = 0 \end{cases}$$

dans lesquelles $D_v = 17kg.s^{-1}$ est le coefficient de frottement en translation et $D_w = 70kg.s^{-1}$ est le coefficient de frottement en rotation.

(F_x, F_y) est le vecteur résultante des forces et τ est le torseur.

$I = l^2M + l^2M$ est le moment d'inertie de rotation.

Ces équations permettent de calculer une vitesse relative à la masse d'air, à laquelle s'ajoute la vitesse de la masse d'air environnante - autrement dit la vitesse du vent - pour obtenir une vitesse relative au sol. Ces équations et les valeurs des différentes constantes sont tirées de [AKI 98].

Capteurs et effecteurs

Les effecteurs sont les propulseurs qui peuvent entraîner le dirigeable aussi bien vers l'avant que vers l'arrière. Il reste à déterminer les capteurs nécessaires pour résoudre la tâche fixée.

Afin de corriger l'effet du vent, le réseau a besoin de connaître la vitesse du dirigeable relativement au sol. Il n'existe pas de capteurs susceptibles de fournir directement cette information, mais ces valeurs peuvent être obtenues à l'aide d'une caméra pointée vers le sol ou au moyen d'un système de triangulation. Ce problème a été contourné ici, la valeur de la vitesse relative au sol étant considérée comme disponible. Lors d'applications réelles ultérieures, ces valeurs seront déterminées en

calculant la vitesse par traitement d'images provenant d'une caméra dirigée vers le sol. La valeur de l'angle que fait le dirigeable par rapport à sa trajectoire désirée a aussi été mise à disposition du réseau, ce type d'information pouvant servir à anticiper et optimiser les commandes à envoyer aux propulseurs. Ces informations ne sont pas indispensables pour résoudre le problème, mais nous avons voulu les fournir afin de laisser à l'évolution la possibilité de s'en servir.

Les premières expériences réalisées concernaient la tâche qui semblait la plus simple à résoudre, c'est-à-dire la correction d'un vent de face ou de dos uniquement. Puis, après avoir obtenu des résultats satisfaisants, nous avons réalisé des expériences avec un vent quelconque.

3.3. *Vent dans l'axe du dirigeable uniquement*

Nous avons commencé nos expériences en donnant pour but à l'évolution de contrer l'effet d'un vent dans l'axe du dirigeable, autrement dit un vent de face ou de dos. L'objectif de l'évolution est de trouver un réseau capable de maintenir la vitesse relative au sol à la valeur d'une consigne.

Préparation de l'évolution

Les senseurs nécessaires pour cette expérience sont les senseurs indiquant la vitesse dans l'axe du dirigeable. Les effecteurs sont les commandes des propulseurs. Afin de profiter des propriétés de symétrie inhérentes au problème traité ici, nous avons disposé les deux commandes "accélérer" de façon symétrique (de même pour les commandes "ralentir") et nous avons disposé les neurones senseurs sur l'axe de la symétrie.

Le calcul de la fitness doit défavoriser tout écart par rapport à la consigne. Nous avons utilisé la fonction suivante :

$$\frac{100}{1 + \sum_{t \leq T} |v_x(t) - v_{consigne}|}$$

et évalué les performances sur une durée de 200 itérations (une itération correspondant à une seconde).

Enfin, le réseau que nous voulons obtenir doit fonctionner pour n'importe quelle vitesse de vent et notamment pour des vitesses variables. Pour que la note de fitness tienne compte de cette contrainte, nous avons choisi de tester cinq fois chaque individu : le premier test place l'individu dans un environnement où le vent est constamment nul, tandis que les quatre tests suivants placent l'animat dans un environnement

où le vent s'inverse brutalement à peu près au milieu de l'expérience. Dans la première de ces expériences, le vent a une vitesse initial positive et faible, dans la deuxième positive et forte, dans la troisième négative et faible et dans la quatrième négative et forte.

Résultats

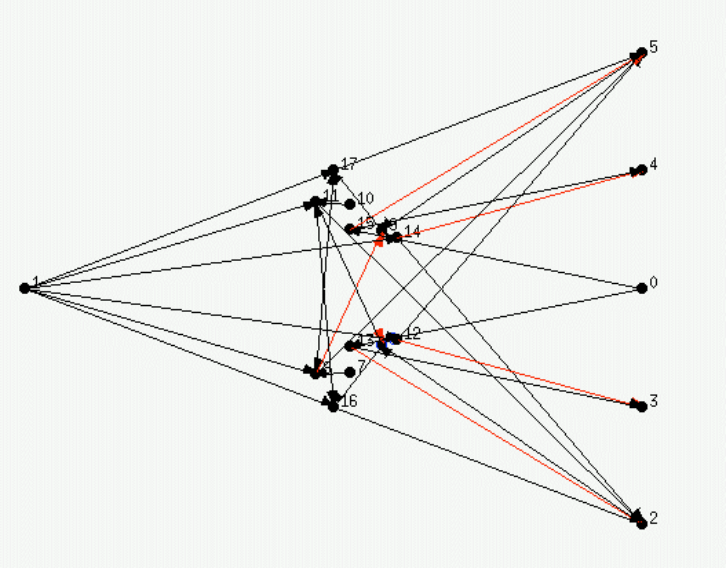

Figure 6. *Meilleur réseau de la 400ème génération (voir texte pour analyse). Neurone 0 :Vitesse positive, Neurone 1 :Vitesse négative, Neurone 5 :Commande "marche avant" propulseur 1, Neurone 4 :Commande "marche arrière" propulseur 1, Neurone 3 :Commande "marche arrière" propulseur 2, Neurone 2 :Commande "marche avant" propulseur 2*

Après 400 générations, le réseau du meilleur individu de la génération (figure 6) utilise tous les capteurs et tous les effecteurs et génère un comportement tout à fait acceptable. Dans ce réseau, il existe trois circuits différents qui relient les entrées aux sorties (avec la symétrie, cela en fait six). Ces circuits relient les capteurs "vitesse trop faible" aux neurones effecteurs "marche avant" de façon excitatrice et aux neurones "marche arrière" de façon inhibitrice. De même, les neurones "vitesse trop importante" sont reliés aux neurones effecteurs "marche avant" de façon inhibitrice et aux neurones effecteurs "marche arrière" de façon excitatrice. Le réseau comprend des neurones de constante de temps très faible. Ces neurones ont un biais négatif et sont donc excités par défaut, ainsi que les équations du chapitre 2.1 l'imposent. Leur faible constante de temps leur permet d'être actifs dès les premières itérations. Leur potentiel de membrane dérive cependant très rapidement et, après quelques itérations, ils

deviennent inactifs et le restent, et ce quelles que soient leurs entrées. Ces neurones accélèrent l'initialisation du réseau : nous avons fait des expériences dans lesquelles nous les avons supprimés et le résultat était un ralentissement de la réponse du réseau. Ce sont donc des neurones d'aide au démarrage.

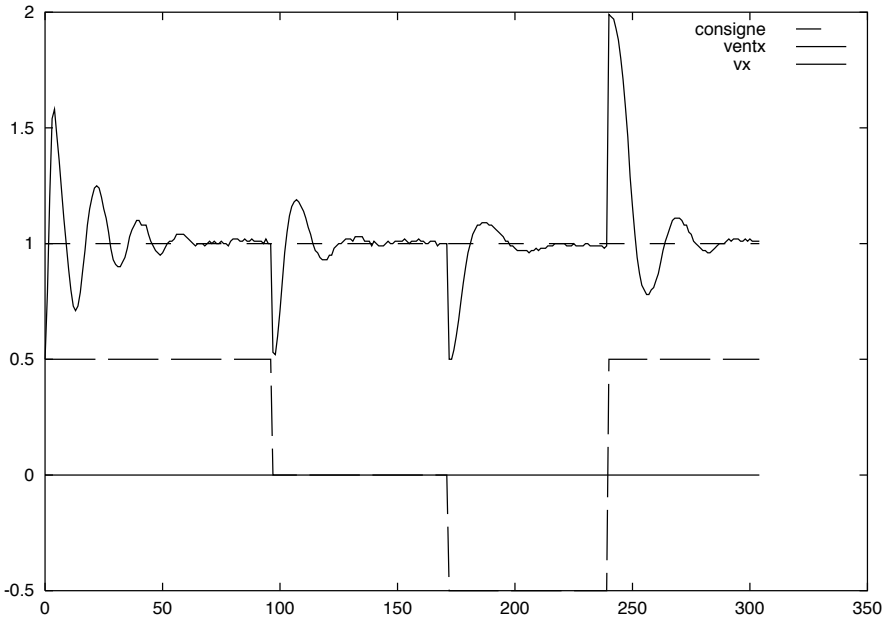

Figure 7. *Performances d'un réseau obtenu après 400 générations. En abscisse itérations du réseau (une itération par seconde de simulation) ; en ordonnée, valeur de la grandeur considérée. Au delà d'un vent de* $0.5m.s^{-1}$*, les propulseurs, tels qu'ils sont définis ici, ne sont pas assez puissants pour s'opposer au vent. Les variations de vent imposées ici (manuellement) sont donc les variations les plus importantes auxquelles un dirigeable peut être soumis tout en étant susceptible de les corriger*

Les performances de ce réseau sont illustrées par la figure 7. Cette figure indique la valeur de la consigne, la valeur de la vitesse relative au sol et la valeur du vent, vent que nous avons fait varier manuellement pendant l'expérience pour tester les capacités du réseau.

Après les premières centaines de générations, les performances stagnent. Ce n'est qu'après plus de 12000 générations que se produit un nouveau bond de performance. Le réseau obtenu lorsque nous avons arrêté l'évolution (après 16000 générations) corrige la vitesse deux fois plus rapidement que le précédent. L'amortissement des oscillations est très rapide, il n'y a plus qu'une demi période d'oscillation.

Cette tâche étant résolue, nous allons passer à l'étape suivante celle de la correction d'un vent quelconque.

Réponse à une consigne variable

Nous avons étudié le comportement du réseau obtenu après 16000 génération lorsque l'on fait varier la consigne. Les résultats sont présentés sur la figure 8. Les évaluations sont faites avec une consigne constante, cependant le comportement du réseau est correct lorsque l'on fait varier la consigne : la vitesse suit la consigne. Ce comportement était prévisible, car du point de vue du réseau, faire varier le vent ou la consigne a le même effet : cela revient à introduire des perturbations dans les valeurs des capteurs d'erreur de vitesse et les évaluations ont été conçues pour que le réseau maintienne ces valeurs à 0.

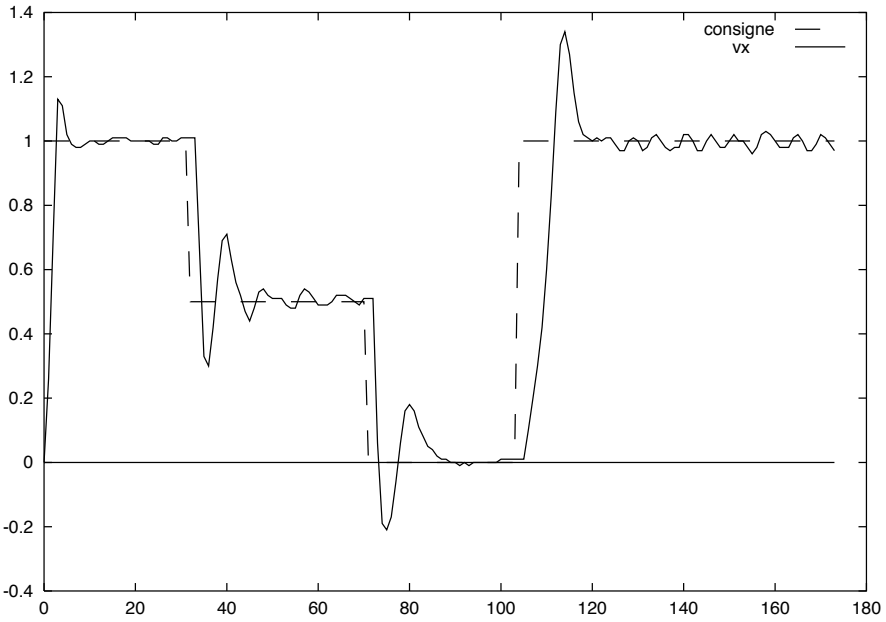

Figure 8. *Réponse d'un réseau à une consigne variable. Le réseau a été obtenu après 16000 générations sur le problème du vent dans l'axe du cap*

3.4. *Vent quelconque sans approche incrémentale*

Dans les expériences qui suivent, le dirigeable doit faire face à un environnement où le vent est quelconque : il doit être capable de maintenir une trajectoire rectiligne, autrement dit d'annuler la composante de vitesse perpendiculaire à son cap. Il doit

aussi rester à une vitesse donnée, ce qui implique que la composante de vitesse sur l'axe de sa trajectoire soit maintenue à la valeur de la consigne.

Préparation de l'évolution

Les senseurs utilisés ici sont les mêmes que précédemment, auxquels s'ajoutent des capteurs de vitesse perpendiculaire au cap et des capteurs indiquant l'angle entre l'axe du dirigeable et le cap (valeurs ne pouvant être fournies que par un autre module, un module de navigation à vue par exemple). Les effecteurs sont les mêmes que précédemment.

L'évaluation d'un individu se fait toujours dans cinq conditions différentes. Ce sont sensiblement les mêmes que pour les expériences précédentes, excepté le fait que la vitesse du vent n'est plus dirigée selon le cap, mais est orientée à environ 45 degrés par rapport au cap. Les intensités sont du même ordre de grandeur que précédemment. Les tests sont tous effectués avec une vitesse de vent perpendiculaire au cap positive car, par symétrie du réseau, le comportement du dirigeable est le même pour les valeurs positives et négatives de cette composante.

La durée de l'évaluation de la fitness a été doublée par rapport aux expériences avec un vent dans l'axe du dirigeable uniquement.

Résultats

Un contrôleur neuronal correct a été obtenu après 5600 générations. Le test des performances de la figure 9 a été réalisé de la façon suivante : le réseau a été initialisé avec un vent de $(0,5m.s^{-1};0,5m.s^{-1})$ (la première valeur est la projection du vent sur l'axe du cap, la deuxième est la projection sur un axe perpendiculaire au cap) ; après une centaine d'itérations, la projection du vent sur l'axe du cap a été fixée à 0 ; ensuite, après encore 60 itérations environ, le vent a été totalement annulé ; enfin, après 60 autres itérations, le vent a été ramené à sa valeur initiale.

Les performances sont moyennes en ce qui concerne la correction du vent dans l'axe de la trajectoire : la correction est rapide mais, en régime permanent, la vitesse oscille (l'amplitude des oscillations pouvant aller jusqu'à 5% de la valeur de la consigne) et il y a une erreur statique qui peut atteindre 7 à 8% de la valeur de la consigne. Par contre, le réseau se comporte très bien en ce qui concerne la correction du vent perpendiculaire au cap.

Le réseau est composé de neurones de sensibilité variable (figure 10). Certains neurones ne sont actifs que lorsque l'écart à la consigne dépasse un certain seuil[2]

[2]Les mêmes mécanismes existent pour la vitesse dans l'axe et perpendiculaire à l'axe.

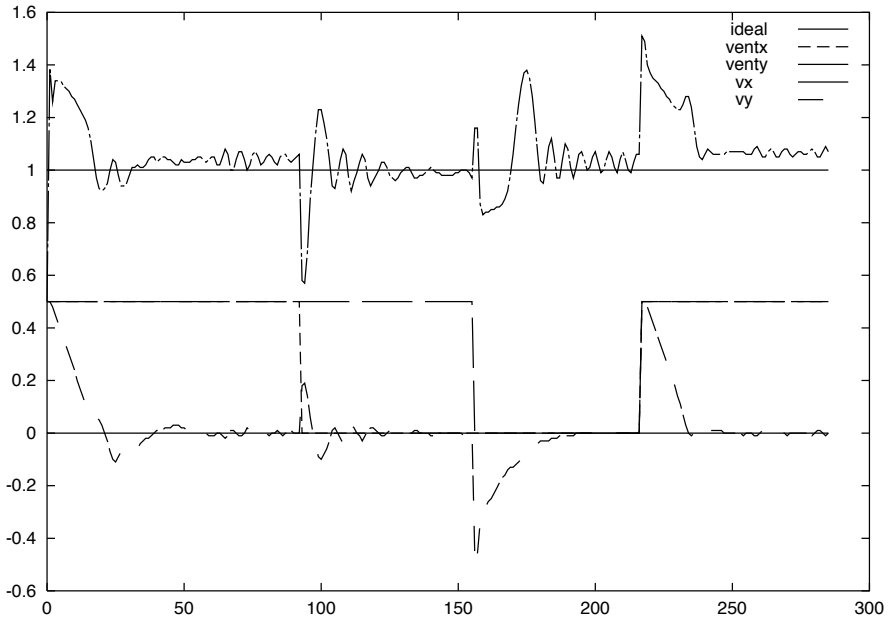

Figure 9. *Performances d'un réseau obtenu après 5600 générations sur le problème du vent quelconque*

(neurones 11, 12 et leurs symétriques par exemple) ; ils prennent alors le dessus sur les autres neurones pour imposer une correction rapide. Ces neurones ont en effet un biais positif plus important que celui des autres neurones et le poids des connexions les reliant aux neurones de sortie est plus important que les poids des autres connexions. D'autres neurones se chargent des corrections plus fines (neurones 13, 24 et leurs symétriques).

3.5. *Vent quelconque avec approche incrémentale*

Dans ces expériences, nous avons utilisé un des réseaux obtenus précédemment (asservissement de la vitesse dans l'axe du dirigeable, paragraphe 3.3) sur lequel nous avons fait évoluer un autre réseau. Ce deuxième réseau devait être capable de moduler le fonctionnement du premier, en créant des connexions vers celui-ci.

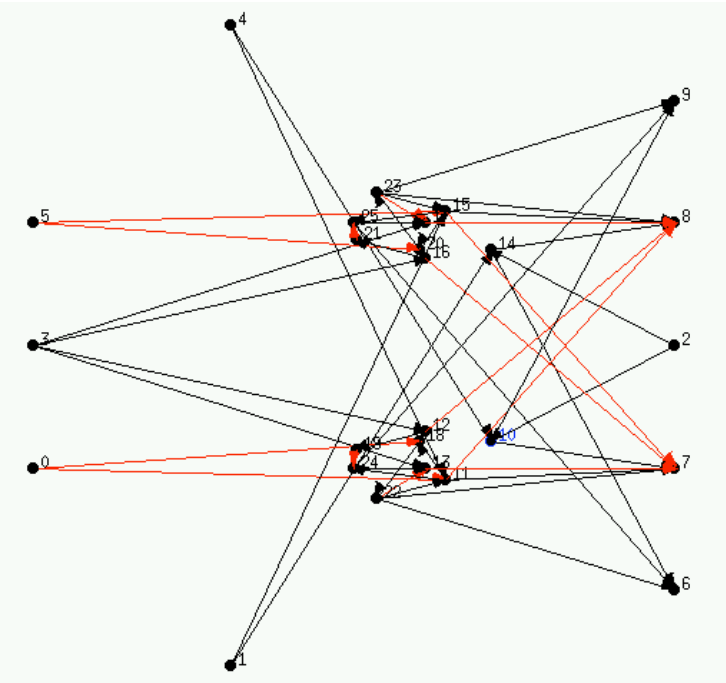

Figure 10. *Réseau obtenu après 5600 générations sur le problème du vent quelconque. Neurone 0 :Angle par rapport au cap négatif, Neurone 1 :Vitesse perpendiculaire au cap négative, Neurone 2 :Vitesse positive, Neurone 3 : Vitesse négative, Neurone 4 : Vitesse perpendiculaire au cap positive, Neurone 5 : Angle par rapport au cap positif, Neurone 6 :Commande "marche avant" propulseur 2, Neurone 7 :Commande "marche arrière" propulseur 2, Neurone 8 :Commande "marche arrière" propulseur 1, Neurone 9 :Commande "marche avant" propulseur 1*

Résultats

En 1900 générations, les réseaux utilisant le réseau d'asservissement du vent dans l'axe obtenu après 400 générations offrent des performances similaires à ceux qui sont obtenus par une approche directe. Ainsi, les performances d'un contrôleur utilisant le réseau obtenu après 16000 générations sont les mêmes que celles du réseau de base seul en ce qui concerne le vent dans l'axe du cap, et l'ajout du deuxième réseau permet de corriger aussi le vent de travers. Dans le cas de la correction du vent de travers, le temps de réponse est de 40 itérations environ et le dépassement est de l'ordre de 30% ; il n'y a pas d'erreur statique et les oscillations sont vite amorties.

Comme lors des expériences précédentes, le deuxième réseau comprend des neurones qui ne répondent qu'à de fortes stimulations, et d'autres neurones qui ont un comportement plus fin. L'action du deuxième réseau sur le comportement du dirigeable peut être soit directe (les neurones du deuxièmes réseau sont directement

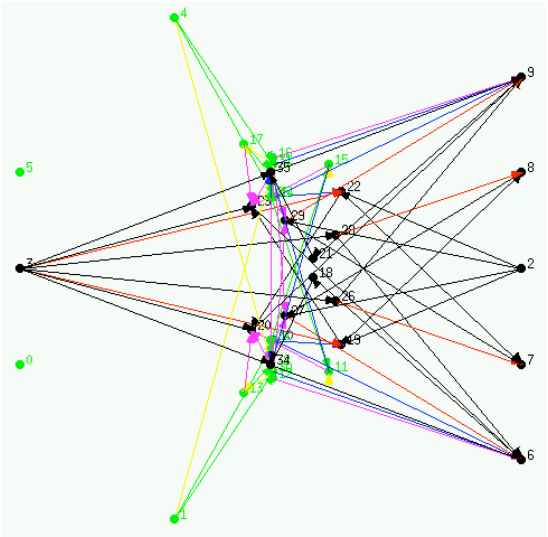

Figure 11. *Réseau obtenu après 1900 générations sur le problème du vent quelconque. Neurone 0 :Angle par rapport au cap négatif, Neurone 1 :Vitesse perpendiculaire au cap négative, Neurone 2 :Vitesse positive, Neurone 3 : Vitesse négative, Neurone 4 : Vitesse perpendiculaire au cap positive, Neurone 5 : Angle par rapport au cap positif, Neurone 6 :Commande "marche avant" propulseur 2, Neurone 7 :Commande "marche arrière" propulseur 2, Neurone 8 :Commande "marche arrière" propulseur 1, Neurone 9 :Commande "marche avant" propulseur 1*

connectés aux effecteurs) soit indirecte (les neurones du deuxième réseau modulent le fonctionnement des circuits du premier réseau) (figure 11).

3.6. *Comparaison approche directe - approche incrémentale*

Les réseaux performants sont obtenus plus rapidement avec une approche incrémentale que lors des expériences sans incrémentalité. En effet, l'évolution avec approche incrémentale aura nécessité en tout 2400 générations (2000+400) pour donner un résultat convenable alors que l'approche directe nécessite environ 5600 génération pour parvenir à un résultat comparable. De plus, l'évolution du premier réseau de l'approche incrémentale nécessitait moins de calcul car l'évaluation était plus deux fois plus courte. L'utilisation d'une méthode incrémentale a donc permis un gain de temps non négligeable. Ce résultat rejoint ceux obtenus par Kobjabachian et al [KOD 99] à propos de l'évolution de contrôleurs neuronaux par un robot Khépera.

4. Conclusions

Dans cet article, nous avons montré qu'il était possible d'obtenir automatiquement des architectures de contrôle pour robots volants. Le principal inconvénient de la méthode utilisée ici, du point de vue d'un ingénieur, est que l'on ne peut pas prévoir les performances des solutions obtenues. Mais cela découle directement de son principal avantage qui est que, excepté en ce qui concerne la programmation de la simulation, elle ne requiert pas d'expertise du domaine d'application. Aucune consigne concernant la méthode de résolution du problème n'est introduite. La fitness est choisie en fonction du comportement souhaité et c'est ensuite à l'évolution de trouver un réseau résolvant le problème. Ainsi, cette technique peut être utile lorsqu'il n'existe pas de méthode classique de résolution, ou si cette méthode est compliquée à mettre en oeuvre. Un autre intérêt de l'approche est la possibilité de réutiliser des réseaux déjà disponibles pour résoudre ensuite un problème plus complexe. Cela accélère les calculs et permet de guider l'évolution, soit pour donner plus d'importance à certains aspects d'un problème, soit pour résoudre des problèmes plus complexes.

5. Perspectives

Dans les applications présentées ici, le concepteur devait imposer le substrat initial et les paramètres des capteurs et effecteurs (constante de normalisation). Ce rôle pourrait être laissé à l'évolution à l'avenir.

Le principal axe de recherche de nos futurs travaux sera l'étude de procédés visant à accélérer l'évolution, d'une part, et à continuer à ajuster les paramètres du réseau pendant une expérience, d'autre part. Pour cela, nous considérerons de nouveaux types de neurones et de connexions, inspirés en partie de la biologie. L'objectif est de mêler apprentissage, développement et évolution afin d'obtenir des réseaux capables d'apprendre à améliorer leurs performances pour résoudre un problème donné.

Bibliographie

[AKI 98] Y. AKISATO, K. SUZUKI ET A. OHUCHI. Reinforcement learning of airship control with adaptive state space segmentation. In *Proceedings of Third International Symposium on Artificial Life, and Robotics (AROB)*, pages 372–376, 1998.

[BEE 95] R. BEER. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4) :469–510, 1995.

[CLA 97] W.J. CLANCEY. *Situated cognition : On human knowledge and computer representations*. Cambridge University Press, 1997.

[GOM 96] T. GOMI ET A. GRIFFITH. Evolutionnary robotics – an overview. In IEEE PRESS, ed, *Proceedings of the IEEE Third International Conference on Evolutionnary Computation*, 1996.

[GRU 94] F. GRUAU. *Synthèse de Réseaux de Neurones par Codage Cellulaire et Algorithmes Génétiques*. PhD thesis, ENS Lyon, Université Lyon I, 1994.

[IJS 99] 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.

[KOD 97] 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.

[KOD 98a] J. KODJABACHIAN. *Développement et évolution de réseaux de neurones artificiels*. PhD thesis, Université Pierre et Marie Curie, 1998.

[KOD 98b] 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.

[KOD 99] J. KODJABACHIAN, C. CORNE ET J.A. MEYER. Evolution of a robust obstacle avoidance behavior in khepera : A comparison of incremental and direct strategies. *Robotics and Autonomous Systems*, 1999. In Press.

[KOZ 92] J. KOZA. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.

[TSC 99] W. TSCHACHER ET J.P. DAUWALDER, eds. *Dynamics, synergetics and autonomous agents*. World Scientific, 1999.