Drones : Simulateur d'Environnement et Apprentissage

Farès Belhadj

Laboratoire d'Intelligence Artificielle
Université de Paris 8
2, rue de la Liberté
93526 Saint Denis cedex
amsi@ai.univ-paris8.fr

Résumé : *Nous présentons un simulateur d'environnement temps réel pour l'entraînement de pilotes automatiques. Ce simulateur gère le comportement de véhicules de type hélicoptère dans un univers virtuel modélisé à partir de paysages fractals. Un pilote automatique, implanté sous la forme d'un Perceptron Multi-Couches, apprend à voler en rase-mottes et évite les obstacles, statiques ou dynamiques, rencontrés. Cet apprentissage est supervisé et effectué en deux phases : il apprend à voler en observant les réactions de deux superviseurs. Ces phases sont repectivement orchestrées par un programme, l'automate volant, et un utilisateur humain. Ce dernier pilote le véhicule via une interface dans laquelle le rendu de la simulation est effectué en images de synthèse.*

Mots-clés : paysages fractals, simulation, images de synthèse, temps réel, apprentissage, drone, planification de trajectoires, réseau de neurones.

1 Introduction

Les systèmes embarqués sont des entités autonomes capables de percevoir et d'interagir avec leur environnement. La perception est généralement réalisée à l'aide de capteurs. Un algorithme utilisant ces informations prend des décisions et réagit aux événements. Il est impératif de vérifier les réponses de l'algorithme dans diverses situations. Celles-ci peuvent être rencontrées dans des environnements reconstitués ou fictifs. Pour cela, il est nécessaire d'avoir un simulateur d'environnement dans lequel les modèles physiques sont respectés et où tout type d'environnement peut être créé. De nombreux travaux existent dans ce domaine, par exemple [Tho98, DRC+00].

Nous proposons une première approche dans laquelle un environnement est un univers virtuel composé d'un terrain (cartes fractales), d'obstacles et de véhicules de type hélicoptère interagissant avec l'ensemble. Nous synthétisons l'ensemble de notre interface et de ses éléments sous la forme d'une librairie. Chaque module de pilotage peut se connecter à cette librairie et utiliser les entrées/sortie proposées.
Nous décrivons, dans ce qui suit, les méthodes utilisées pour la génération des différents types de paysages, terrains et arbres. Nous donnons une modélisation du type de véhicules importés. Nous détaillons, pour ces véhicules, les possibilités de perception de leur environnement ainsi que le processus de commande. Enfin, nous réalisons et comparons dans ce cadre nos deux implémentations de pilotes automatiques, un *automate volant* et un *Perceptron Multi-Couches*.

2 Les composantes de l'environnement

2.1 La génération de terrains

Nous représentons les terrains selon un maillage régulier donné par une matrice de valeurs d'altitudes. Ces valeurs sont échelonnées sur un intervalle $[0, 255]$ et la matrice résultante produit une carte topographique du terrain. Pour générer ces cartes, nous utilisons au choix trois algorithmes fractals [Man95] (cf. figure 1). L'algorithme du *plasma* [Aud97, Ste90] ainsi que BROWN-GAUSS produisent ce que nous appelons des « nuages fractals ». Enfin, un troisième algorithme, basé sur la méthode du « Quick Union Find » [Sed98], produit des cartes de labyrinthes aléatoires.

Génération de nuages fractals :

Les deux algorithmes proposés sont basés sur un système d'interpolations bilinéaires par déplacement des milieux. Soit une matrice $\mathcal{A}_{(M \times N)}$ dont les valeurs aux quatre coins $p_1 = (0, 0)$, $p_2 = (M-1, 0)$, $p_3 = (0, N-1)$ et $p_4 = (M-1, N-1)$ sont prises aléatoirement. Nous calculons récursivement les valeurs aux points intermédiaires.

L'interpolation des points varie suivant l'algorithme utilisé.

a. Le plasma : nous calculons à partir du rectangle (p_1, p_2, p_3, p_4) les valeurs aux quatre points des milieux des bords — p_5, p_6, p_7 et p_8 — ainsi que la valeur au point central p_9. Ces valeurs $v(p_i)$ sont obtenues selon :

$$\begin{cases} v(p_5) = v(\frac{M-1}{2}, 0) = \frac{v(p_1)+v(p_2)}{2} + o(L) \;\Big|\; v(p_6) = v(\frac{M-1}{2}, N-1) = \frac{v(p_3)+v(p_4)}{2} + o(L) \\ v(p_7) = v(0, \frac{N-1}{2}) = \frac{v(p_1)+v(p_3)}{2} + o(L) \;\Big|\; v(p_8) = v(M-1, \frac{N-1}{2}) = \frac{v(p_2)+v(p_4)}{2} + o(L) \\ v(p_9) = v(\frac{M-1}{2}, \frac{N-1}{2}) = \frac{v(p_1)+v(p_2)+v(p_3)+v(p_4)}{4} + o(L) \end{cases}$$

où $o(L)$ est un entier relatif aléatoire proportionnel à L avec $L = max(M, N)$.
Chaque valeur $v(p_i)$ est ramenée dans l'intervalle $[0, 255]$ par modulo.

Nous obtenons, après une première itération, quatre nouveaux rectangles de dimensions $\frac{M}{2} \times \frac{N}{2}$: (p_1, p_5, p_7, p_9), (p_5, p_2, p_9, p_8), (p_7, p_9, p_3, p_6) et (p_9, p_8, p_6, p_4). Nous répétons récursivement ce calcul sur chaque sous rectangle obtenu tant que $L = max(\frac{M}{2^n}, \frac{N}{2^n}) > 2$ où n est la profondeur de récursion.
Remarque : Si une valeur a été attribuée à un point, alors cette valeur ne sera plus modifiée.

b. BROWN-GAUSS : de la même manière que précédemment, nous calculons, pour les quatre points initiaux, cinq nouvelles valeurs aux points p_5, p_6, p_7, p_8 et p_9. Les dépendances entre points ne sont plus les mêmes et la variable aléatoire $o(\Delta)$, prise ici, est régie par une distribution gaussienne. Ces valeurs sont calculées selon :

$$\begin{cases} v(p_9) = v(\frac{M-1}{2}, \frac{N-1}{2}) = \frac{v(p_1)+v(p_2)+v(p_3)+v(p_4)}{4} + o(\Delta) \\ v(p_5) = v(\frac{M-1}{2}, 0) = \frac{v(p_9)+v(p_1)+v(p_2)}{3} + o(\Delta) \;\Big|\; v(p_6) = v(\frac{M-1}{2}, N-1) = \frac{v(p_9)+v(p_3)+v(p_4)}{3} + o(\Delta) \\ v(p_7) = v(0, \frac{N-1}{2}) = \frac{v(p_9)+v(p_1)+v(p_3)}{3} + o(\Delta) \;\Big|\; v(p_8) = v(M-1, \frac{N-1}{2}) = \frac{v(p_9)+v(p_2)+v(p_4)}{3} + o(\Delta) \end{cases}$$

où :
- $o(\Delta)$ est un réel aléatoire égal à $\Delta \times Gauss()$.
- $\Delta = \frac{\sigma}{\sqrt{2^n}}$ avec n le numéro de l'itération en cours.
- σ est l'écart-type de la distribution gaussienne (≈ 1).
- $Gauss()$ est une fonction qui renvoie un réel compris entre $\pm 2\sigma$ suivant une distribution gaussienne.

Tant que $L = max(\frac{M}{2^n}, \frac{N}{2^n}) > 2$, nous réitérons cette opération sur chaque sous rectangle obtenu. Pour finir, nous échelonnons la matrice résultante sur l'intervalle [0, 255].

Génération de labyrinthes :

Nous proposons une méthode de production aléatoire de labyrinthes 1-connexes : un unique chemin relie deux positions distinctes (la connexité de ces labyrinthes peut être augmentée en supprimant certains murs). Pour un labyrinthe de dimensions $M \times N$, une matrice $\mathcal{L}$ de dimensions $(2M+1) \times (2N+1)$ est créée. Cette matrice représente les données topographiques du labyrinthe. Elle est initialisée telle que chaque position libre est entourée de murs (cf. exemple pour $M = N = 3$) où les murs sont représentés par la valeur -1. Pour construire le labyrinthe, nous sélectionnons aléatoirement une position murée séparant, en 4-connexité, deux positions libres. Si aucun chemin ne relie ces deux positions, alors le mur est supprimé. Cette opération est répétée jusqu'à ce que toutes les positions libres, à l'initialisation, soient connectées.
Nous donnons un exemple d'initialisation de $\mathcal{L}$ pour $M = N = 3$:

$$\mathcal{L} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \boxed{0} & -1 & \boxed{1} & -1 & \boxed{2} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \boxed{3} & -1 & \boxed{4} & -1 & \boxed{5} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \boxed{6} & -1 & \boxed{7} & -1 & \boxed{8} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

Donc pour M et N quelconques, $\mathcal{L}$ est initialisée par :
– Si la position (i, j) est telle que i ou j pair alors $\mathcal{L}[i][j] = -1$ et (i, j) est murée ;
– Sinon, $\mathcal{L}[i][j] = (M \times \lfloor \frac{i}{2} \rfloor) + \lfloor \frac{j}{2} \rfloor$ et (i, j) est non murée.

Puis nous posons que $\mathcal{L}$ est assimilée à un graphe à $M \times N$ composantes. Nous avons alors un nœud à identifiant unique par composante. Nous produisons, à partir de ce graphe, un graphe à une composante. Pour ce faire, une position murée séparant deux composantes disjointes est sélectionnée aléatoirement ; la position est libérée et le plus petit identifiant est propagé sur la nouvelle composante connexe. Nous posons que deux nœuds sont disjoints si et seulement si leurs identifiants respectifs sont différents. L'algorithme se termine quand les $M \times N$ nœuds du graphe ont un 0 comme identifiant. La matrice résultante est échelonnée[1] sur l'intervalle [0, 255].

[1] Par exemple : 0 pour les positions non murées et 255 pour les positions murées.

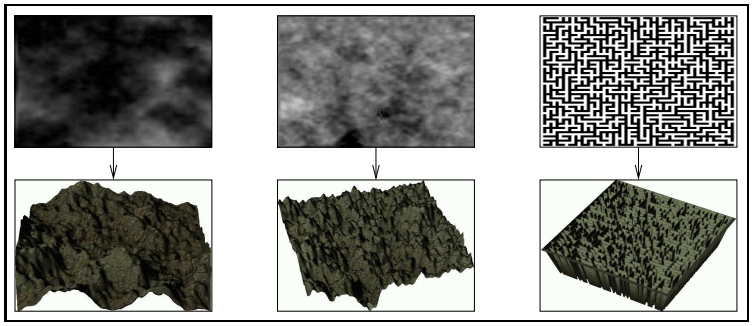

FIG. 1 – Cartes topographiques générées et rendu 3D ; de gauche à droite : plasma, BROWN-GAUSS et labyrinthe.

2.2 Les L-Systèmes

Plus qu'un aspect visuel, les arbres représentent des obstacles supplémentaires dans le parcours des drones. Nous implémentons, pour la génération d'arbres, un interprète de « Bracketed » L-Systèmes à composante stockastique [PL89, PL90, PH92]. Ce module crée les images représentant les différents types d'arbres à partir d'un fichier décrivant les règles de production. Nous utilisons ces images comme des textures 2D que nous plaquons sur deux rectangles perpendiculaires. Les arbres, ainsi créés, sont placés dans le paysage aléatoirement et par groupe de même famille. Un exemple de fichier descriptif ainsi que le rendu 2D de l'arbre obtenu est donné en figure 2.

```
1. Arbre001{                                    // Nom de l'objet
      axiom ++++++++++++X                       // L'axiome ω
      angle 7.2                                 // La variation d'angle σ
      angle0 85.0                               // L'angle initial α0
      step 50.0                                 // Le pas de déplacement d en pixels
      gen 9                                     // Le nombre de générations
      X=F[@.5+++++++++X]-F[@.4----------!X]@.6X  // Une règle de production
}
```

FIG. 2 – Fichier descriptif et rendu 2D produit par l'interprète de L-Système.

2.3 Les véhicules

Les aéronefs sont placés et évoluent dans les paysages virtuels générés par les algorithmes fractals. Nous allons étudier maintenant comment ces véhicules sont représentés et se meuvent. Nous implémentons un module d'importation d'objets 3D réalisés par modeleur. Ce module permet une représentation tridimensionnelle paramétrable pour chaque type de véhicule, actuel ou à venir. Il créé l'objet *OpenGL* [WNDS99] décrit par des fichiers *A.S.E.* — « Ascii Scene Export » — et le place dans le paysage. Pour un hélicoptère, nous utilisons quatre fichiers *A.S.E.*, chacun décrit respectivement le cockpit, la queue, le disque rotor et le rotor anti-couple. Cette subdivision aide à la construction de l'enveloppe convexe contenant l'ensemble ; nous utilisons cette enveloppe dans la détection de collisions.
Pour obtenir une simulation de la dynamique de vol d'un hélicoptère, nous proposons un modèle physique simplifié du véhicule. Dans notre modèle, la sustentation et la propulsion du véhicule sont assurées par le disque rotor qui produit deux types de mouvements : le tangage et le roulis. Le rotor anti-couple produit une force contraire à la direction de rotation du disque rotor ; elle assure le mouvement en lacet. Seule la variation de cette force est considérée dans notre modèle. La dynamique de vol de l'aéronef est montrée sur la figure 3, elle obéit à :

$$\left\{ \begin{array}{l} \overrightarrow{F} + m \times \overrightarrow{g} = m \times \overrightarrow{a} \text{ projetée sur chaque axe :} \\ \overrightarrow{F} = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix}, \quad \overrightarrow{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} : \left\{ \begin{array}{lcl} a_x & = & F_x \\ a_y & = & F_y - g \\ a_z & = & F_z \end{array} \right. \Rightarrow \left\{ \begin{array}{lcll} a_x & = & \frac{\cos\theta \times ||\overrightarrow{F}||}{m} & (x) \\ a_y & = & \frac{\sin\rho \times ||\overrightarrow{F}||}{m} - g & (y) \\ a_z & = & \frac{\cos\rho \times ||\overrightarrow{F}||}{m} & (z) \end{array} \right. \end{array} \right.$$

Avec $\overrightarrow{F}$ la force produite par le disque rotor, $\overrightarrow{g}$ la gravité, m la masse du véhicule et $\overrightarrow{a}$ son accélération.

Enfin, les différentes projections du vecteur vitesse sont corrigées par l'ajout d'une force de résistance à l'air. Nous obtenons :

$$v_i = v_i - (k \times v_i^2)$$

où k est le coefficient aérodynamique, déterminé pour une vitesse maximale donnée, environ $90m.s^{-1}$ pour ce type d'aéronefs, par :

$$k = \frac{m \times a}{v_{max}^2}$$

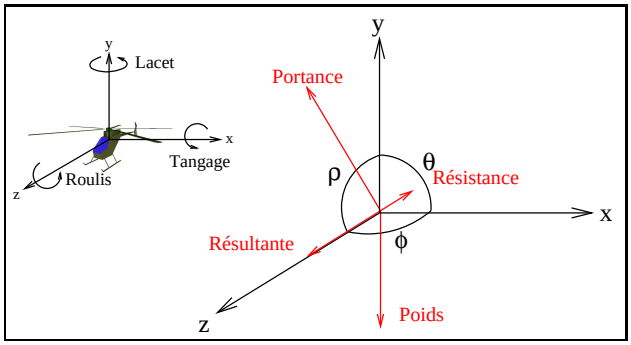

FIG. 3 – Dynamique de vol de l'aéronef.

3 Les éléments de perception et de commande

Chaque programme (pilote automatique) accède via notre librairie à une série de fonctions lui permettant de piloter un ou plusieurs hélicoptères ainsi que de percevoir l'univers virtuel local ou global. Nous décrivons ici ces outils de perception et le processus de commande [LLS+01].

3.1 La perception globale : le *plus sûr chemin*

Notre algorithme du *plus sûr chemin*, noté *P.S.C.*, est une méthode rapide de calcul du chemin discret reliant deux points quelconques sur une carte. Le chemin résultant minimise les altitudes empruntées réalisant le meilleur compromis entre altitude et distance. L'utilisateur de l'interface sélectionne une coordonnée d'arrivée pour chaque véhicule ; le module *P.S.C.* calcule les chemins aux buts.
Cet algorithme est basé sur la méthode de calcul du plus court chemin dans un graphe [Dij59, LS95]. Le graphe des chemins possibles est donné par la matrice, en 8-connexité, des données topographiques du terrain. Les arètes sont pondérées par l'altitude en chaque point ou nœud du graphe. Nous introduisons des modifications de l'algorithme afin d'améliorer les temps de calcul des *P.S.C.* Pour cela, nous découpons la carte des altitudes en sous-régions rectangulaires de dimensions égales. Une moyenne des altitudes est calculée pour chaque sous-région et un *P.S.C. grossier* est produit pour relier l'ensemble des sous-régions empruntées. À partir de là nous raffinons le résultat obtenu en calculant un *P.S.C.* à l'intérieur de chaque sous-région empruntée par le chemin *grossier*. La complexité des graphes représentant chaque sous-région est bien moindre comparée à l'ensemble.
Nous obtenons par cette méthode un *P.S.C.* équivalent à une version non optimisée dans des délais inférieur à la seconde. Un ratio de temps de calcul supérieur à 10^3 a été mesuré entre l'algorithme classique et le nôtre. Cette mesure est donnée pour une carte topographique de 1024×1024 nœuds.

3.2 La perception locale

Tout aéronef créé possède sa propre configuration d'outils de perception. Ces outils sont implémentés sous la forme d'un ensemble paramétrable d'instruments de bord. Nous pouvons les sélectionner dans la liste contenant : des capteurs de distance, des instruments de mesure de l'angle de direction, l'angle de tangage, l'angle de roulis et la mesure de la vitesse. Cette dernière est donnée en *unité-terrain* par seconde, notée $ut.s^{-1}$, et ut est la distance

horizontale séparant, dans le quadrillage du terrain, deux sommets voisins. Enfin, un système radar permet de localiser les différents véhicules présents à proximité de l'appareil.
Nous utilisons, dans nos implémentations de pilotes automatiques, un instrument de mesure de l'angle de direction ainsi que six capteurs de distance. Ces derniers renvoient la distance, en ut, les séparant d'un obstacle. Leur portée maximale est de $10ut$. Quand à l'instrument de mesure de l'angle de direction, il renvoie l'angle permettant à l'aéronef de suivre le chemin discret donné par l'utilisateur via le module du *P.S.C.*

3.3 Le processus de commande

Dans un hélicoptère, les commandes correspondent au *manche de pas cyclique*, il incline le disque rotor afin de modifier la direction de la portance, le *palonnier* contrôle le rotor anti-couple, le *levier de pas collectif* contrôle l'inclinaison des pales et la *manette de gaz* modifie la vitesse de rotation du disque rotor.
Nous émulons les commandes réelles d'un hélicoptère par l'intermédiaire d'un automate à états qui produit une modification des paramètres de l'hélicoptère et génère la dynamique de vol correspondante. De cette manière, toute action produite par le pilote devient accessible en *lecture / écriture*. Nous pouvons ainsi, à tout moment, interroger l'interface sur l'état de chaque commande.
Nous obtenons alors une représentation numérique des actions effectuées par le pilote. Cette représentation nous permet d'introduire la notion *d'apprentissage* par observation. Elle est utilisée pour l'apprentissage du *Perceptron Multi-Couches*. L'observateur apprend à piloter en *copiant* les réaction du superviseur (cf. figure 4).

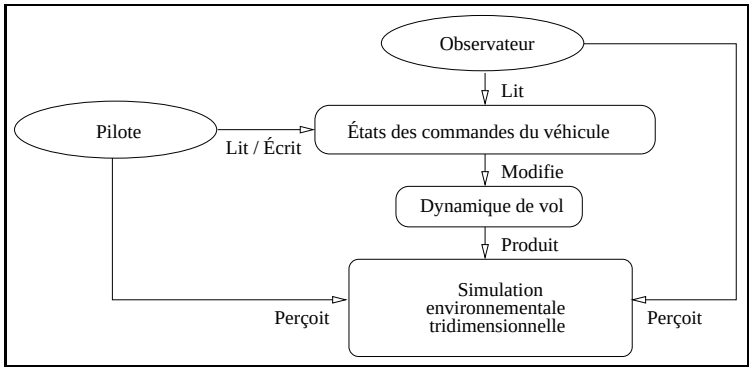

FIG. 4 – Le processus d'apprentissage par observation.

4 Les pilotes automatiques

Comme application de notre librairie de gestion d'environnements virtuels, nous implémentons deux pilotes automatiques, un automate déterministe et un *Perceptron Multi-Couches*, et nous comparons leurs résultats. Pour cette évaluation, la configuration de la perception des hélicoptères est la suivante :

- six capteurs de distance configurés comme suit :
  - cinq capteurs en position avant-centre et orientés respectivement $(\rho = \frac{\pi}{15}, \phi = -\frac{\pi}{15})$, $(\rho = \frac{\pi}{15}, \phi = \frac{\pi}{15})$, $(\rho = -\frac{\pi}{15}, \phi = \frac{\pi}{15})$, $(\rho = -\frac{\pi}{15}, \phi = -\frac{\pi}{15})$, $(\rho = 0, \phi = 0)$;
  - un capteur en position centre-bas et orienté $(\rho = -\frac{\pi}{2}, \phi = 0)$;
- un instrument de mesure de l'angle de direction : il renvoie l'angle ϕ permettant à l'aéronef de rejoindre la prochaine position discrète donné par le *plus sûr chemin* ;
- un instrument de mesure de la vitesse au sol.

4.1 L'automate volant

Ce pilote doit, à partir des informations de perception locale (instruments de mesures et capteurs de distance) et globale (le *plus sûr chemin*), s'orienter dans l'environnement virtuel et arriver jusqu'aux coordonnées spécifiées par l'utilisateur de l'interface. Il est implémenté sous la forme d'un automate déterministe pour lequel, d'une part,

les capteurs définissent les entrées de l'automate et d'autre part, ses différents états donnent les commandes correspondantes pour l'hélicoptère (cf. figure 5). Ce pilote adopte un comportement de *précaution* : il vole à des altitudes moyennes en prenant ses distances par rapport aux obstacles.

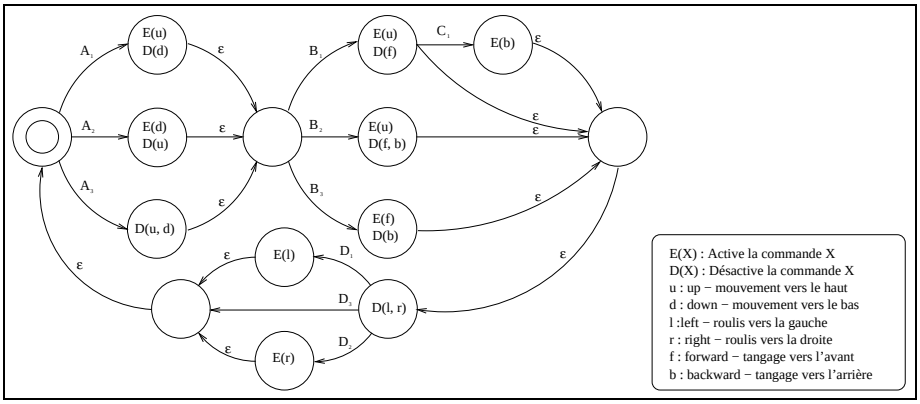

FIG. 5 – La réaction aux événements (perception via les capteurs de distance, de vitesse et d'orientation) de l'automate volant produit un changement dans le comportement de l'hélicoptère.

Nous donnons les conditions de passage d'états sur la figure 6, où $D(cpt_n)$ est la distance, en *unité-terrain*, renvoyée par le capteur n et vH est la projection de la vitesse sur le plan horizontal.

$$
\begin{array}{lcl}
A_1 & \Longleftrightarrow & (D(cpt_6) < 1\ ut) \vee (D(cpt_3) < 3\ ut) \vee (D(cpt_4) < 3\ ut) \\
A_2 & \Longleftrightarrow & \neg A_1 \wedge (D(cpt_6) > 2\ ut) \wedge (D(cpt_3) > 6\ ut) \wedge (D(cpt_4) > 6\ ut) \\
A_3 & \Longleftrightarrow & (\neg A_1 \wedge \neg A_2) \\
B_1 & \Longleftrightarrow & (D(cpt_5) < 3\ ut) \vee (D(cpt_1) < 3\ ut) \vee (D(cpt_2) < 3\ ut) \\
B_2 & \Longleftrightarrow & \neg B_1 \wedge (\ (D(cpt_5) < 6\ ut) \vee (D(cpt_1) < 6\ ut) \vee (D(cpt_2) < 6\ ut)\ ) \\
B_3 & \Longleftrightarrow & (\neg B_1 \wedge \neg B_2) \\
C_1 & \Longleftrightarrow & (vH > 0) \\
D_1 & \Longleftrightarrow & (\delta\phi > 0.5\ rad) \\
D_2 & \Longleftrightarrow & (\neg D_1 \wedge (\delta\phi < -0.5\ rad)) \\
D_3 & \Longleftrightarrow & (\neg D_1 \wedge \neg D_2) \\
\varepsilon & \Longleftrightarrow & \text{vrai}
\end{array}
$$

FIG. 6 – Définition des conditions de changement d'état de l'automate volant.

4.2 Le *Perceptron Multi-Couches*

La dynamique de vol de l'automate, ainsi défini, suit un comportement de précaution. Notre but est d'obtenir un nouveau pilote qui adopte une attitude *dangereuse* mais efficace. Celui-ci devra voller en rase-mottes, à une vitesse soutenue et sans collision. Pour ce faire, nous utilisons l'automate volant à des fins d'entraînement d'un *Perceptron Multi-Couches* noté *P.M.C.*

Le *P.M.C.* est un réseau de neurones [Ros58, DMS+02, MP90] à apprentissage supervisé. Notre implémentation est constituée de trois couches de neurones avec huit neurones en entrée pour une compatibilité avec l'automate et six neurones en sortie pour les six commandes de l'hélicoptère. Lors de la **première phase** d'apprentissage, nous interfaçons l'automate (i.e. le superviseur) aux commande de dix hélicoptères. Le *P.M.C.* apprend par observation (cf. figure 4) et converge vers le même comportement.

En **deuxième phase**, l'utilisateur *humain* prend, via l'interface clavier, les commandes d'un hélicoptère et adopte un comportement plus dynamique et vole au plus près du sol. Comme dans le cas précédent, le *P.M.C.* apprend par observation : cette phase d'apprentissage dure quelques minutes. L'interface permet à l'utilisateur de *redonner la main* au *P.M.C.* et ainsi vérifier l'amélioration de son comportement.

Nous obtenons à la suite des deux phases d'apprentissage un nouveau pilote au comportement dit **amélioré**. Nous comparons les résultats obtenus par l'automate volant aux résultats du *P.M.C.* après chaque phase d'apprentissage (cf. tableau 1). Ces tests sont effectués dans les mêmes conditions, sur un échantillon de dix terrains et pour les mêmes couples de coordonnées (point de départ et point d'arrivée). Nous pouvons voir sur la figure 7 une réduction considérable de la distance par rapport au sol. Le *P.M.C.* **amélioré** effectue un vol en rase-mottes et sa vitesse horizontale est globalement constante.

	Vitesse horizontale	Distance par rapport au sol
Automate volant	$1,344\ ut/s$	$44,784\ ut$
P.M.C.	$1,501\ ut/s$	$49,3375\ ut$
P.M.C. **amélioré**	$1.2522\ ut/s$	$19,3815\ ut$

TAB. 1 – Comparatif entre l'automate volant, le *P.M.C* et le *P.M.C* amélioré. La distance par rapport au sol est donnée est *unité-terrain* et la vitesse horizontale moyenne en *unité-terrain* par seconde. La fréquence de l'échantillon est de 40 itérations par seconde.

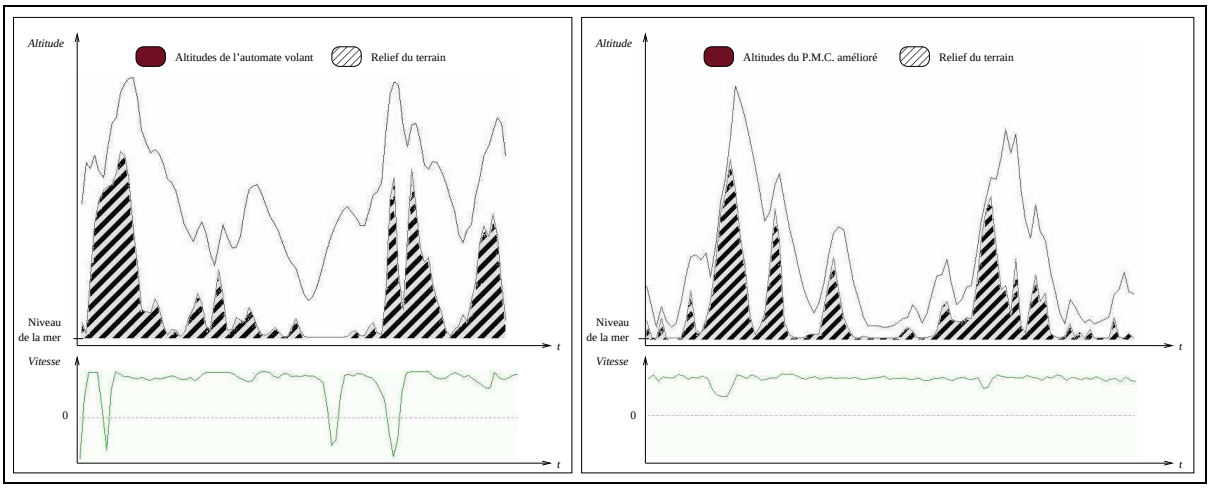

FIG. 7 – Comparaison des courbes des altitudes et de vitesse horizontale de l'automate volant et du *Percepton Multi-Couches* **amélioré**.

4.3 Conclusion et perspectives

Nous avons développé, sous la forme d'une librairie, un simulateur temps réel d'environnement d'entraînement pour drones. L'interface nous a permis de réaliser plusieurs phases d'apprentissage, de visualiser et de comparer les comportements obtenus. Une modélisation plus complexe et plus interactive de l'interface peut être produite en étendant les résultats obtenus par cette première approche.
Dans cette optique, la création et l'intégration à l'environnement d'un modeleur pour *véhicules et capteurs virtuels* permettrait d'élargir le champ applicatif de l'interface. Ce module prendrait à sa charge la dynamique de mouvement des véhicules créés. Une autre perspective est la possibilité de mettre en concurrence plusieurs programmes autonomes et d'observer leur évolution comportementale. Ces programmes communiqueraient entre-eux via l'environnement virtuel et feraient émerger des notions de concurrences ou d'entre aide.

Références

[Aud97] P. Audibert. *Algorithmes et Programmation*. Université Paris 8, 1995-1997.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[DMS+02] G. Dreyfus, J.-M. Martinez, M. Samuelides, M.B. Gordon, F. Badran, S.Thiria, and L. Hérault Sous la direction de Gérard Dreyfus. *Réseaux de neurones : Méthodologie et applications*. Eyrolles, 2002.

[DRC+00] T. Duval, J. Regincós, A. Chauffaut, D. Margery, and B. Arnaldi. Interactions collectives locales en immersion dans des univers virtuels 3d avec gasp. Actes de la conférence ERGO-IHM 2000, Biarritz, France, Octobre 2000.

[LLS+01] J.-P. Laumond, Florent Lamiraux, Sepanta Sekhavat, Pascal Morin, Claude Samson, Patrick Rives, Michel Devy, Malik Ghallab, Bernard Espiau, and Frank Génot sous la direction de Jean-Paul Laumond. *La robotique mobile*. Hermès, 2001.

[LS95] C.A. Lazere and D.E. Shasha. *Out of Their Minds*. Copernicus Books, 1995.

[Man95] B. Mandelbrot. *Les Objets Fractals*. Flammarion, quatrième edition, 1975-1995.

[MP90] M.L. Minsky and S.A. Papert. Perceptrons. 1990.

[PH92] P. Prusinkiewicz and J. Hanan. *Lindenmayer Systems, Fractals, and Plants*. Springer Verlag, 1992.

[PL89] P. Prusinkiewicz and A. Lindenmayer. Developmental models of multicellular organisms : A computer graphics perspective. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Addison Wesley, 1989.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *Algorithmic Beauty of Plants*. Springer Verlag, 1990.

[Ros58] F. Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological Review 65 : 386-408*, 1958.

[Sed98] R. Sedgewick. *Algorithms in C*. Addison-Wesley, third edition, 1998.

[Ste90] R.T. Stevens. *Advanced Fractal Programming in C*. M&T, 1990.

[Tho98] G. Thomas. Représentation d'environnement urbains pour l'animation de piétons. *AFIG'98*, 1998.

[WNDS99] M. Woo, J. Neider, T. Davis, and D. Shreiner. *Le guide officiel à l'apprentissage d'OpenGL, version 1.2*. CompusPress France, 1999.